

SÉCURITÉ DES APPLICATIONS WEB

Animé par :
Hamza Sarraj

PLAN

- 1- INTRODUCTION
- 2- LE PROTOCOLE HTTP EN DÉTAIL
- 3- LES VULNÉRABILITÉS DES APPLICATIONS WEB
- 4- LE FIREWALL RÉSEAU DANS LA PROTECTION D'APPLICATIONS HTTP
- 5- SÉCURISATION DES FLUX AVEC SSL/TLS
- 6- PRINCIPE DU DÉVELOPPEMENT SÉCURISÉ
- 7- L'AUTHENTIFICATION DES UTILISATEURS
- 8- CONCLUSION



INTRODUCTION

- Plus de 90 % des applications utilisées par les entreprises et les particuliers reposent sur des technologies Web et chaque jour, des millions de tentatives d'intrusion sont enregistrées à travers le monde. Les attaques ciblent aussi bien les failles technique que les erreurs humaines.

LE MONDE DES ATTAQUANTS : QUI, POURQUOI, COMMENT ?



Cybercriminels
(majoritaires), groupes
étatiques, hacktivistes,
insiders. Acteurs
externes ≈ 62 % des
brèches



Financières en
tête (≈ 88 %), puis
espionnage



Identifiants volés
exploitation de
vulnérabilités connues ;
l'usage d'identifiants
valides reste une porte
d'entrée clé

ARCHITECTURE CLASSIQUE D'UNE APPLICATION WEB



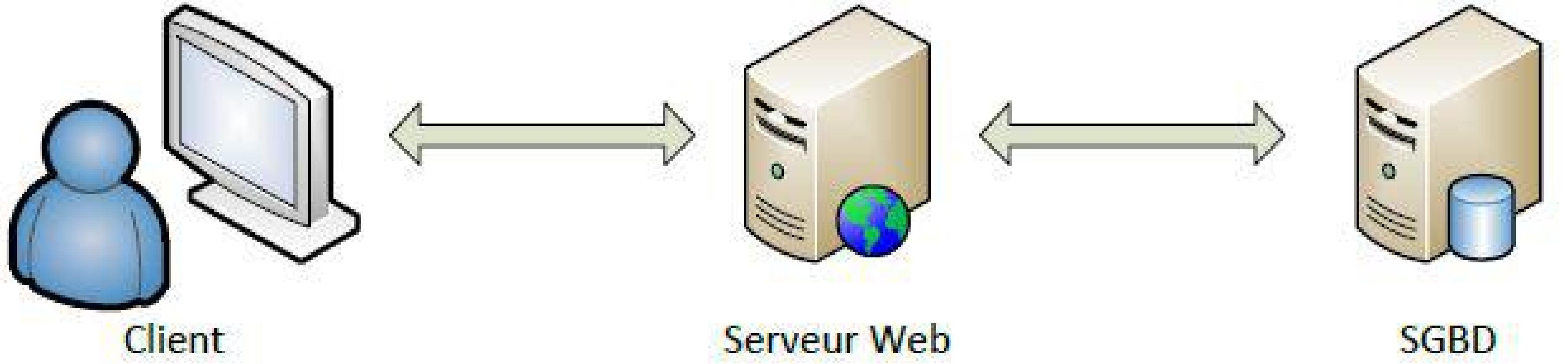
- Client (navigateur / mobile)
Interface utilisateur, formulaires, cookies, scripts.



- Serveur applicatif (back-end)
Gère la logique métier (Java, PHP, Python, Node.js).



- Base de données
Contient les informations sensibles



Vulnérable aux
attaques XSS, CSRF.

Vulnérable aux
injections, mauvaises
configurations

Vulnérable aux
attaques SQL
Injection, exfiltration
de données



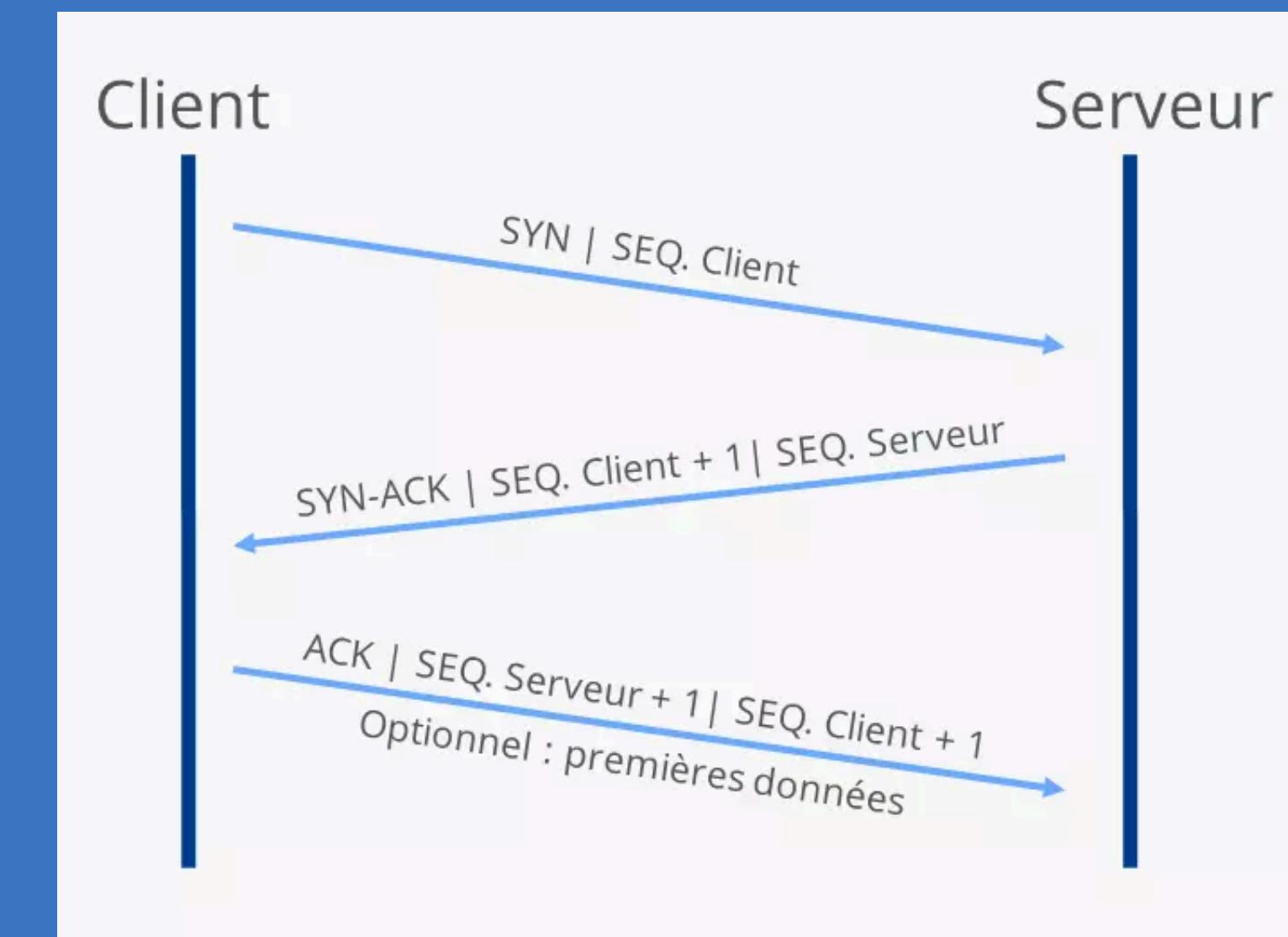
LE PROTOCOLE HTTP EN DÉTAIL

HTTP = HyperText Transfer Protocol
Fonctionne sur TCP (port 80) ou TLS/SSL
(HTTPS, port 443)
Protocole client-serveur basé sur des
requêtes/réponses textuelles
Sans état (stateless), complété par
cookies/session



TCP & HTTP

- **TCP** : connexion fiable, orientée flux
- **HTTP 1.0** : une requête = une connexion (non persistant)
- **HTTP 1.1** : persistance des connexions (Keep-Alive)
- **Pipelining** : plusieurs requêtes envoyées sans attendre les réponses (peu utilisé, remplacé par HTTP/2 multiplexé)

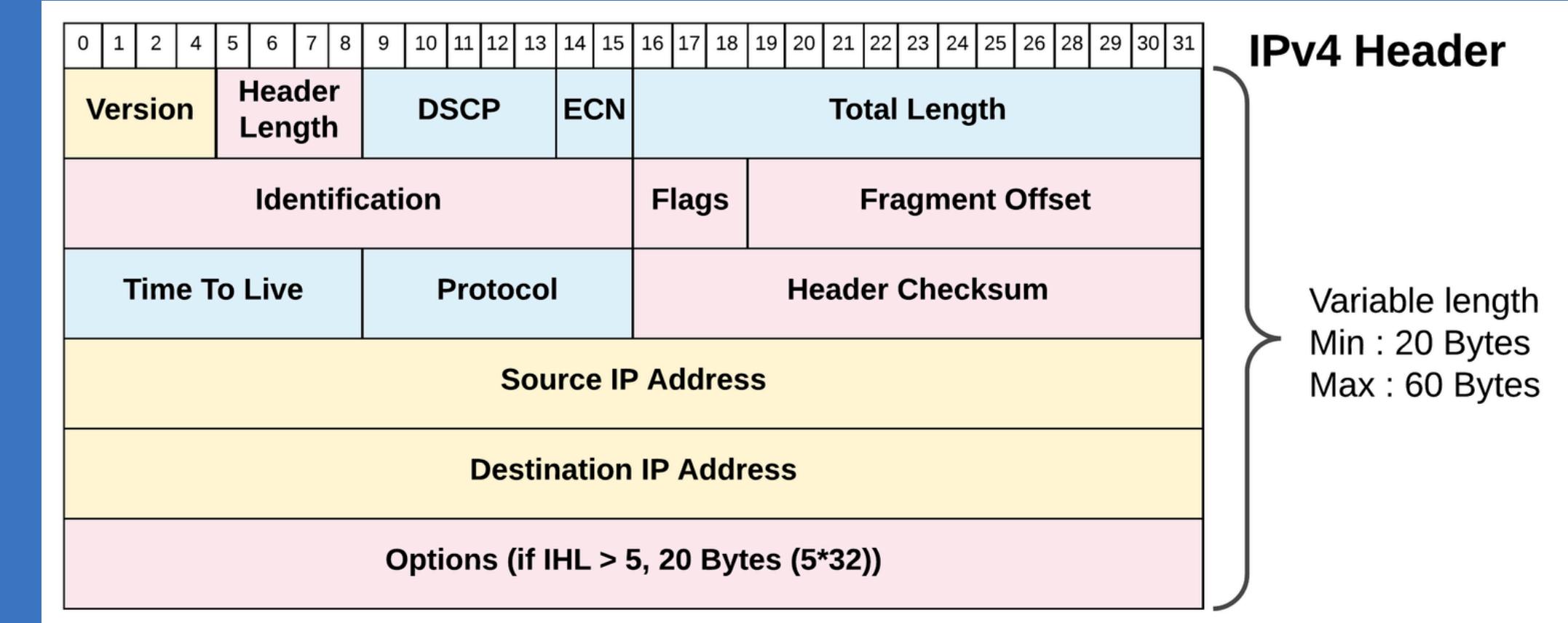


LES PDU (MÉTHODES HTTP)

- **GET** : lecture de ressource
- **POST** : envoi de données (formulaires, upload)
- **PUT** : création/remplacement de ressource
- **DELETE** : suppression d'une ressource
- **HEAD** : récupération des métadonnées (en-têtes uniquement)
- **TRACE** : diagnostic (boucle la requête → peut être abusé)

EN-TÊTES ET CODES HTTP

En-têtes (headers) :
infos envoyées avec les
requêtes/réponses



- 1xx : infos → (ex. 100 Continue)
- 2xx : succès → (200 OK)
- Codes de statut :**
- 3xx : redirection → (301, 302)
 - 4xx : erreur client → (404 Not Found)
 - 5xx : erreur serveur → (500 Internal Error)

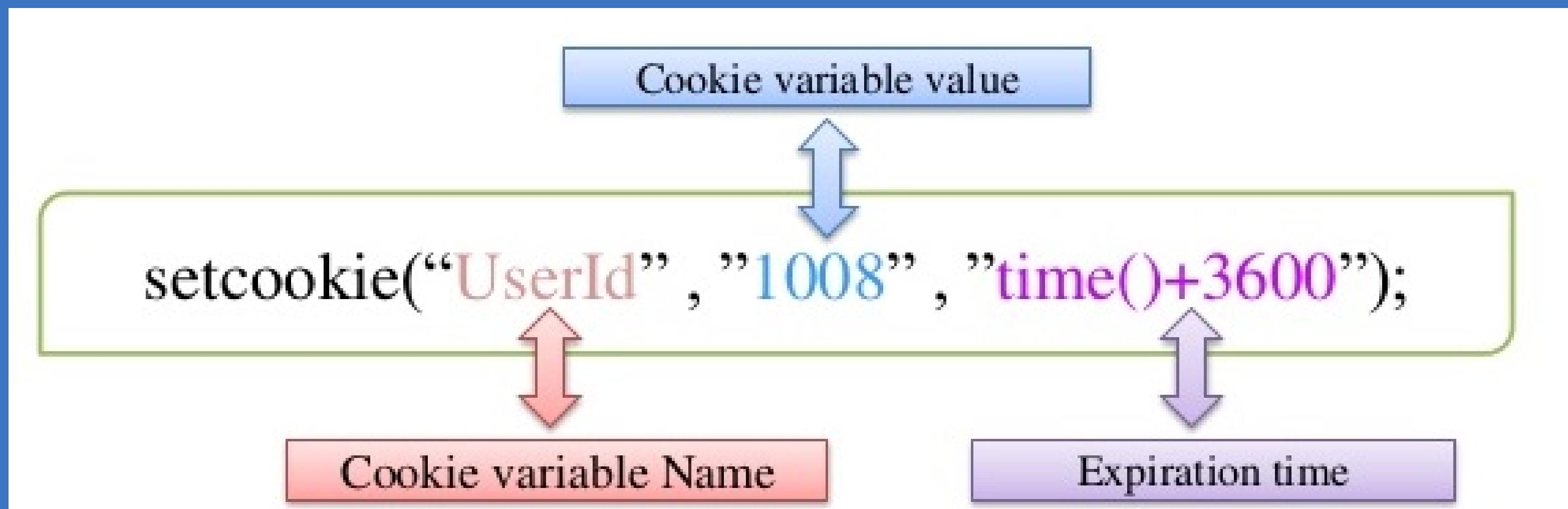


FONCTIONNALITÉS AVANCÉES

- **Redirection** : renvoie vers une autre page (301 permanent, 302 temporaire).
- **Hôtes virtuels** : plusieurs sites sur une seule IP.
- **Proxy cache** : stocke les pages pour les charger plus vite.
- **Tunneling** : permet de passer du trafic chiffré (HTTPS) via un proxy.

COOKIES ET SESSIONS

- **Cookies** = petits fichiers dans le navigateur pour garder la session.



AUTHENTIFICATION HTTP

- **Basic** : login + mot de passe → pas sécurisé.
- **Digest** : un peu mieux (mot de passe haché), mais dépassé.
- **Bearer/JWT** : jetons modernes pour les API.
- **OAuth2** : utilisé pour SSO (connexion Google, Facebook).

JWT Authorization Grant (RFC 7523 2.1)

grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer

assertion=JWT

scope=...

```
{  
    "access_token": "access token",  
    "token_type": "...",  
    "expires_in": ...,  
    "scope": "..."  
}
```

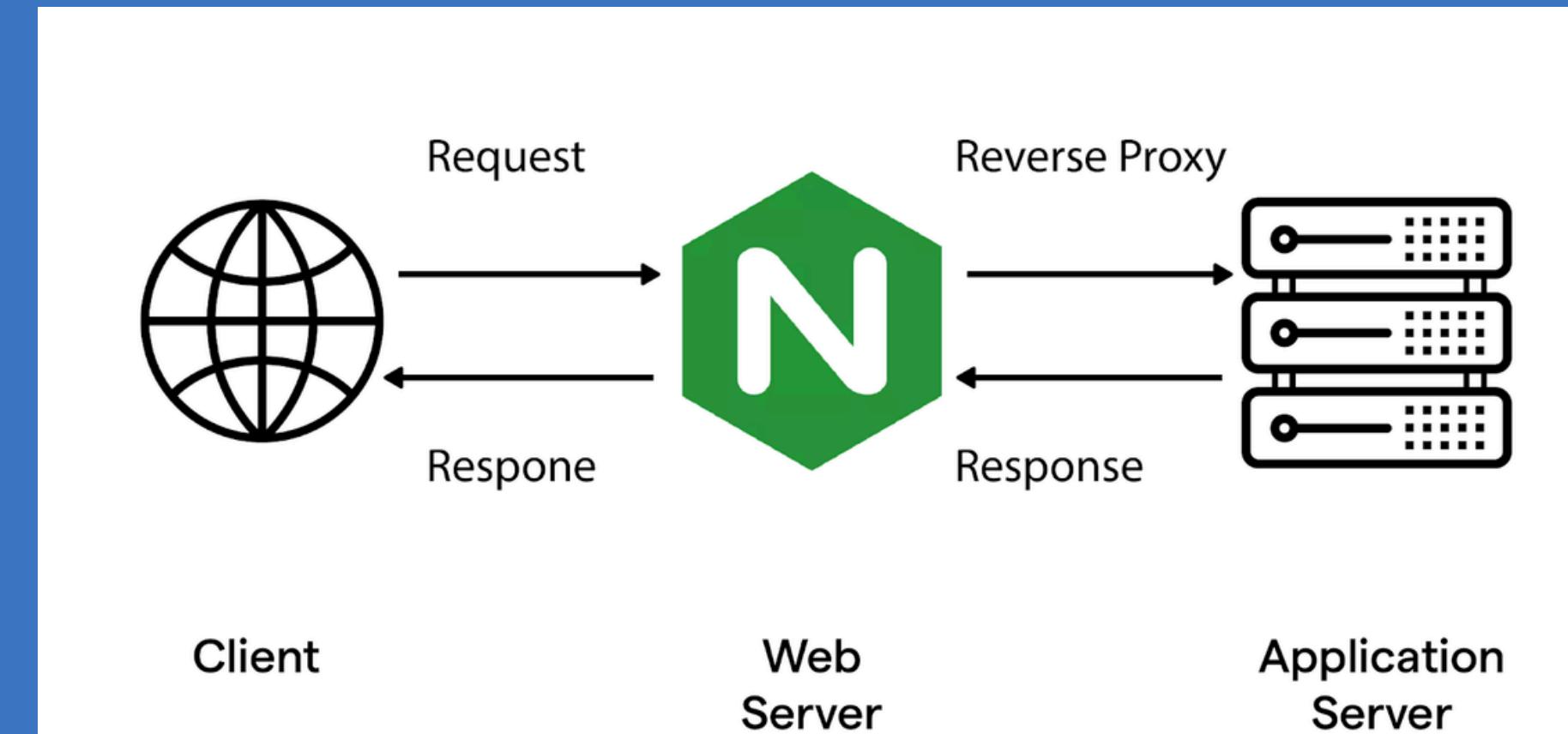
Authorization
Server

Token
Endpoint



OPTIMISATION ET ÉQUILIBRE

- **Compression (gzip, Brotli)** → pages plus rapides.
- **Proxy** → intermédiaire pour filtrer/accélérer.
- **Load Balancing** → répartir les connexions entre plusieurs serveurs.



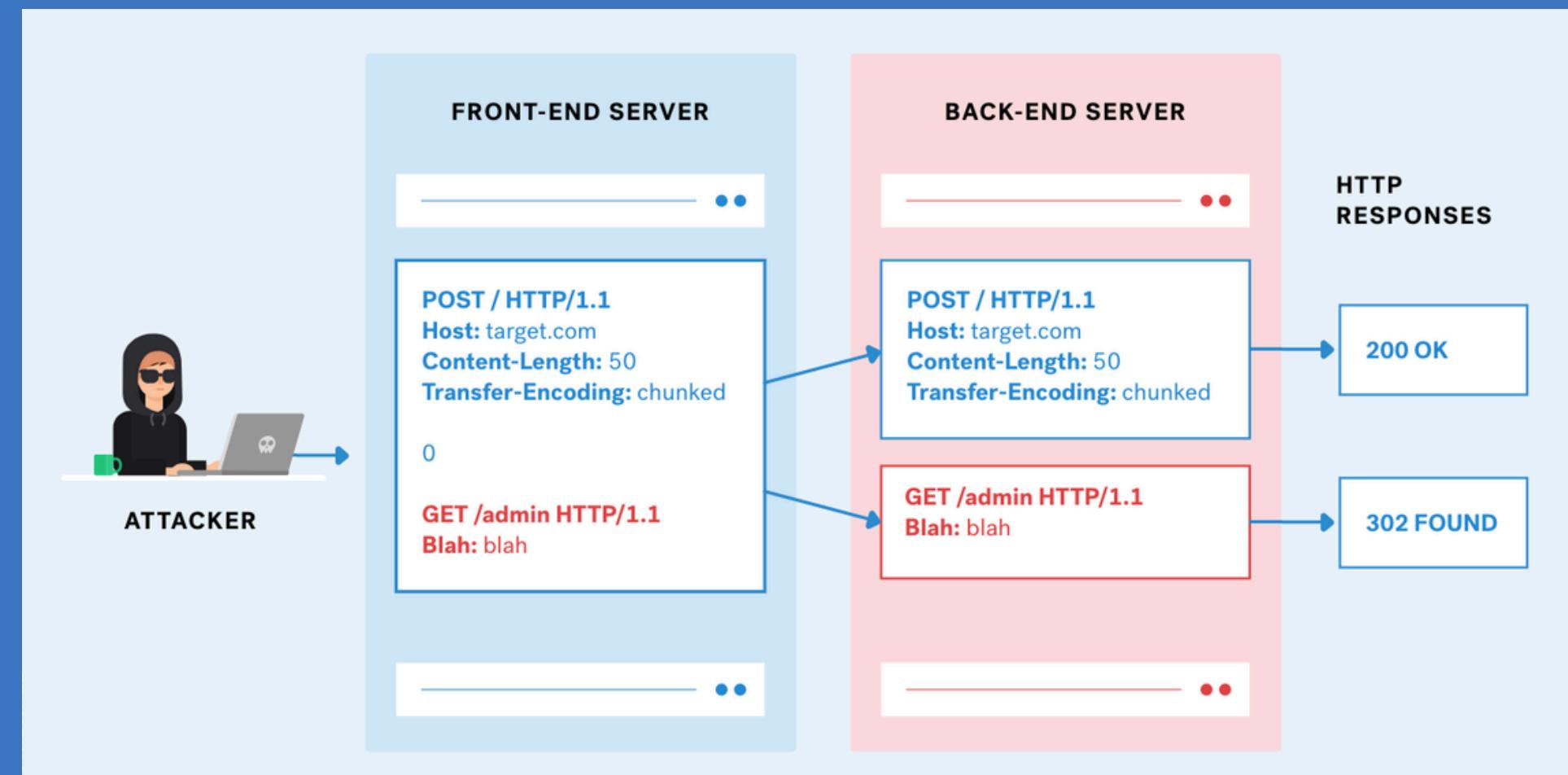


LES VULNÉRABILITÉS DES APPLICATIONS WEB

ATTAQUES PROTOCOLAIRES

HTTP Request Smuggling

- Profite d'erreurs de lecture entre serveurs proxy.
- Permet d'injecter des requêtes cachées.



ATTAQUES PROTOCOLAIRES

HTTP Response Splitting

- Forcer un serveur à renvoyer plusieurs réponses.
- Sert à rediriger l'utilisateur ou injecter du contenu malveillant.

```
HTTP/1.1 302 Found
Content-Type: text/plain
Location: \r\n
Content-Type: text/html \r\n\r\n

<html><h1>hacked!</h1></html>
Content-Type: text/plain
Date: Thu, 13 Jun 2019 16:12:20 GMT
```



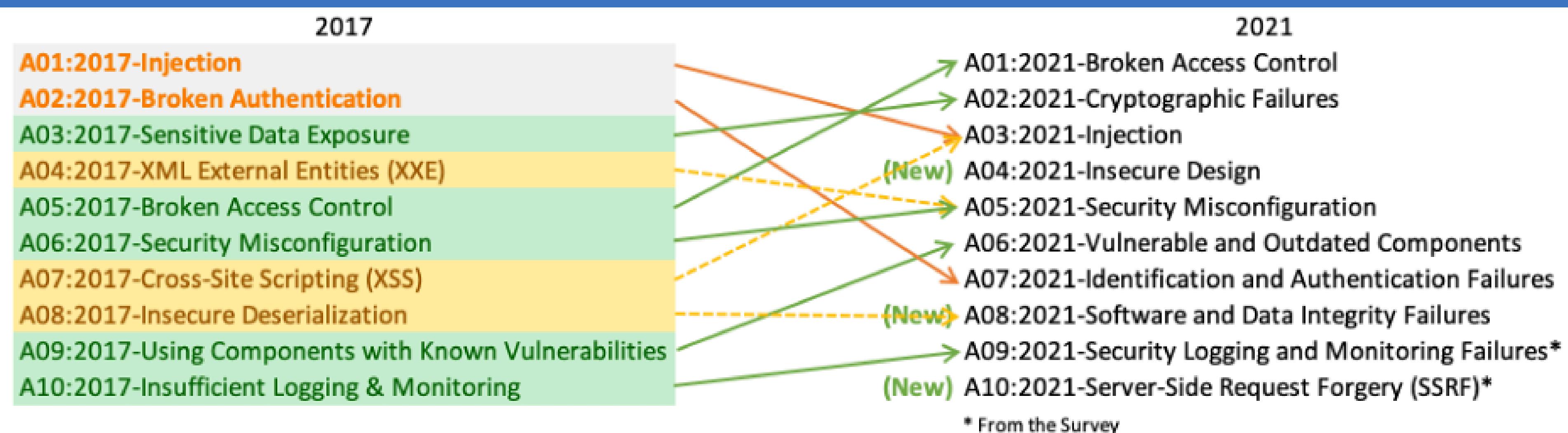
Pourquoi les applications Web sont exposées ?

- Accessibles 24/7 depuis Internet
- Complexité croissante (API, microservices, frameworks)
- Données sensibles en jeu (paiement, santé, identités)
- Utilisateurs non formés → erreurs fréquentes (mots de passe faibles, phishing)
- Les failles découvertes sont vite exploitées par des attaquants automatisés (bots, scans).



OWASP (Open Web Application Security Project) est une organisation internationale à but non lucratif qui regroupe une communauté d'experts en cybersécurité. Son objectif est de sensibiliser, former et fournir des outils gratuits pour améliorer la sécurité des applications.

OWASP est surtout connu pour son **Top 10**, une liste des dix failles de sécurité Web les plus critiques, qui sert de référence mondiale pour les développeurs, architectes et auditeurs.



ATTAQUES XSS (CROSS-SITE SCRIPTING)

Principe : injection de script malveillant dans une page Web.

```
<form action="" method="GET">
  <input type="text" name="name" placeholder="Votre nom">
  <input type="submit" value="Envoyer">
</form>

<?php
if (isset($_GET['name'])) {
    echo "Bonjour " . $_GET['name'];
}
?>
```



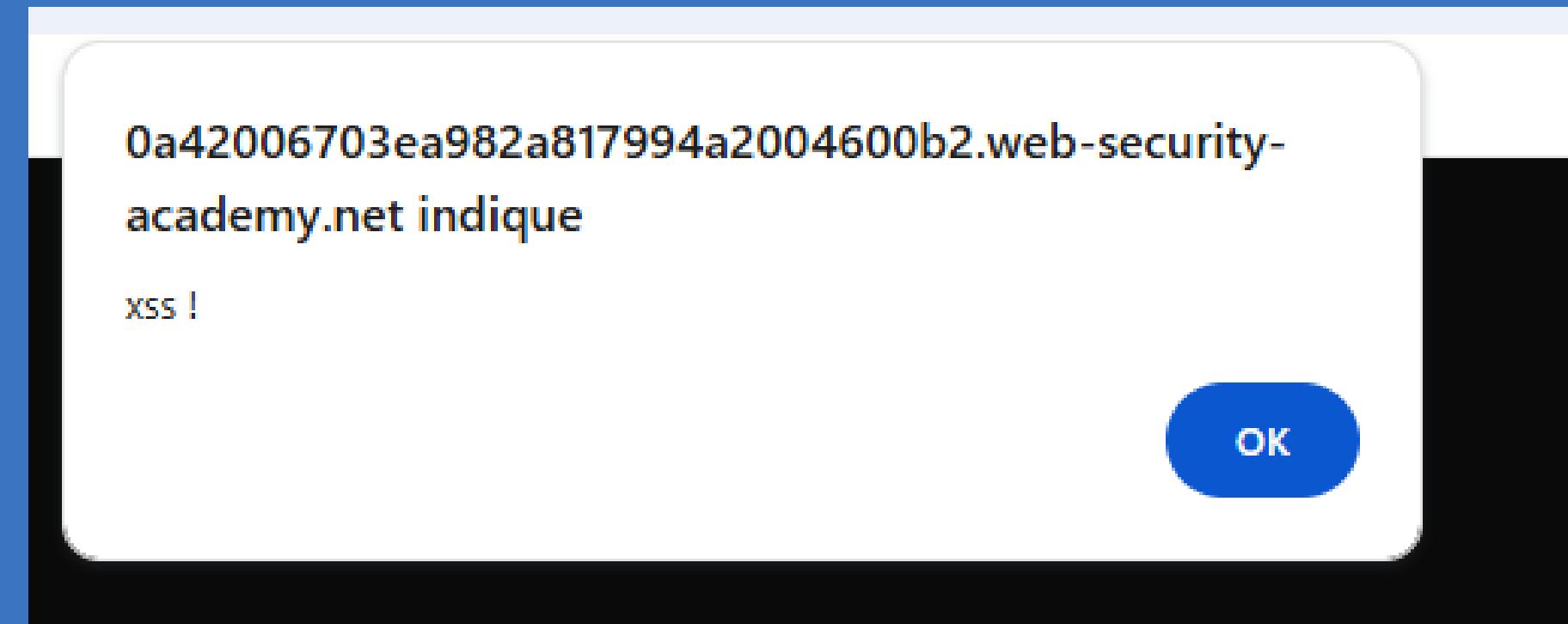
la valeur de name est renvoyée sans filtre dans la page.

ATTAQUES XSS (CROSS-SITE SCRIPTING)

Si l'attaquant envoie dans le champ texte

```
<script>alert('XSS !');</script>
```

- La page affichera une fenêtre d'alerte JavaScript.
Cela prouve que du code malveillant a été injecté et exécuté dans le navigateur.



ATTAQUES XSS (CROSS-SITE SCRIPTING)



- Échapper les entrées
(encode/sanitize)
- Cookies sécurisés (HttpOnly,
SameSite)
- Utiliser CSP (Content Security Policy)

ATTAQUES PAR INJECTION

Types courants :

- SQL Injection → accès/modification de la base de données

```
$query = "SELECT * FROM users WHERE email = '" . $_GET['email'] . "'";
```

Si l'utilisateur entre :

" OR '1'='1

La requête devient :

```
SELECT * FROM users WHERE email = '' OR '1'='1';
```

ATTAQUES PAR INJECTION

Types courants :

- Command Injection → exécution de commandes système

```
$ip = $_GET['ip'];
system("ping -c 4 " . $ip);
```

Si l'utilisateur entre :

```
127.0.0.1; ls
```

La commande exécutée sera :

```
ping -c 4 127.0.0.1; ls
```

ATTAQUES PAR INJECTION

Types courants :

- LDAP Injection (annuaire Active Directory) : accès à des annuaires (Active Directory)

```
String filter = "(uid=" + userInput + ")";  
ldap.search("ou=users,dc=example,dc=com", filter);
```

Si l'utilisateur entre :

)(|(uid=)(password=*))

Le filtre LDAP devient :

(uid=*)(|(uid=*)(password=*))

ATTAQUES PAR INJECTION



- Utiliser des requêtes préparées
- Validation stricte des entrées utilisateur
- Droits minimaux sur la base de données

ATTAQUES SUR LES SESSIONS

Types courants :

- Cookie Poisoning : modification des cookies pour usurper un compte.

Le site stocke dans un cookie les droits d'accès :

```
user=Hamza; role=Admin
```

L'attaquant modifie le cookie en :

```
user=Hamza; role=user
```



Si l'application ne vérifie pas côté serveur, l'attaquant devient administrateur.

ATTAQUES SUR LES SESSIONS

Types courants :

- Session Hijacking : vol du cookie de session → connexion en tant que victime.

L'utilisateur se connecte → reçoit un cookie de session :

```
PHPSESSID=ab12cd34
```

L'attaquant vole ce cookie (via XSS, sniffing HTTP non chiffré, malware).

Il l'injecte dans son navigateur → il devient l'utilisateur **légitime**.

ATTAQUES SUR LES SESSIONS

Types courants :

- Fixation de session : forcer l'utilisateur à utiliser un cookie contrôlé par l'attaquant.

L'attaquant génère un cookie valide :

```
| PHPSESSID=attacker123
```

Il force la victime à l'utiliser (lien piégé, redirection).

Quand la victime se connecte, la session reste la même.

L'attaquant utilise le même ID pour accéder au compte

ATTAQUES SUR LES SESSION



- Cookies sécurisés : HttpOnly, Secure, SameSite.
- Renouveler l'ID de session après connexion.
- Utiliser toujours HTTPS.
- Détruire la session à la déconnexion.



LE FIREWALL RÉSEAU DANS LA PROTECTION D'APPLICATIONS HTTP

C'QUOI UN FIREWALL RÉSEAU ?

Un dispositif qui permet de filtrer le trafic réseau selon des règles qui sert à :

- Bloquer/autoriser des flux (IP, ports, protocoles).
- Séparer les zones réseau (LAN, DMZ, Internet).

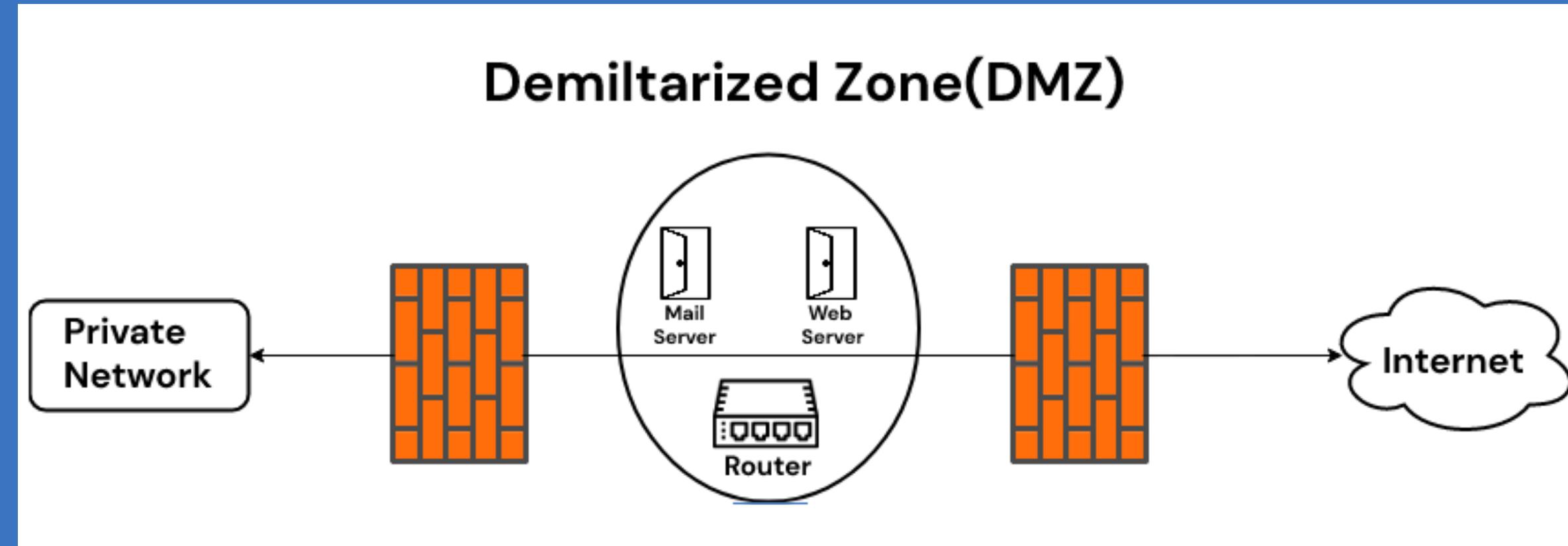
Exemple : autoriser le port 443 (HTTPS), bloquer le reste.



FONCTIONS D'UN FIREWALL RÉSEAU

- Filtrage statique : règles IP/port (ex. bloquer tel service).
- Inspection dynamique (stateful) : suit les connexions établies.
- NAT (Network Address Translation) : masque les adresses internes.
- Segmentation réseau : cloisonner les zones sensibles.

LES DMZ DANS UNE ARCHITECTURE N-TIERS



La DMZ (Demilitarized Zone) est une zone réseau intermédiaire placée entre Internet et le réseau interne d'une organisation. Elle sert de zone tampon pour héberger les serveurs exposés au public, tout en protégeant les systèmes critiques du LAN.

LES DMZ DANS UNE ARCHITECTURE N-TIERS

Nombre de DMZ selon l'architecture :

2-Tiers (Client ↔ Serveur) : 1 DMZ (héberge le serveur web).

3-Tiers (Client ↔ Serveur web ↔ BDD) : 2 DMZ

- DMZ 1 : serveur web (front-end).
- DMZ 2 : serveur applicatif.

La BDD reste en interne (LAN).

LIMITES DU FIREWALL RÉSEAU FACE AUX APPLIS WEB



- Filtrage basé sur IP/port, pas sur le contenu HTTP.
- Laisse passer le trafic HTTP(S) → mais ne voit pas les failles applicatives : Injection SQL, XSS, CSRF...
- Chiffrement HTTPS = boîte noire pour le firewall classique.
- Attaques Web → doivent être bloquées par des mécanismes supplémentaires.

Firewall

Travaille sur les **couches basses** (IP, ports, protocoles).
Règles basées sur adresses IP, ports (80, 443, etc.), protocoles (TCP, UDP).

Mais:
il laisse passer tout le trafic HTTP/HTTPS, même s'il contient une attaque.

WAF

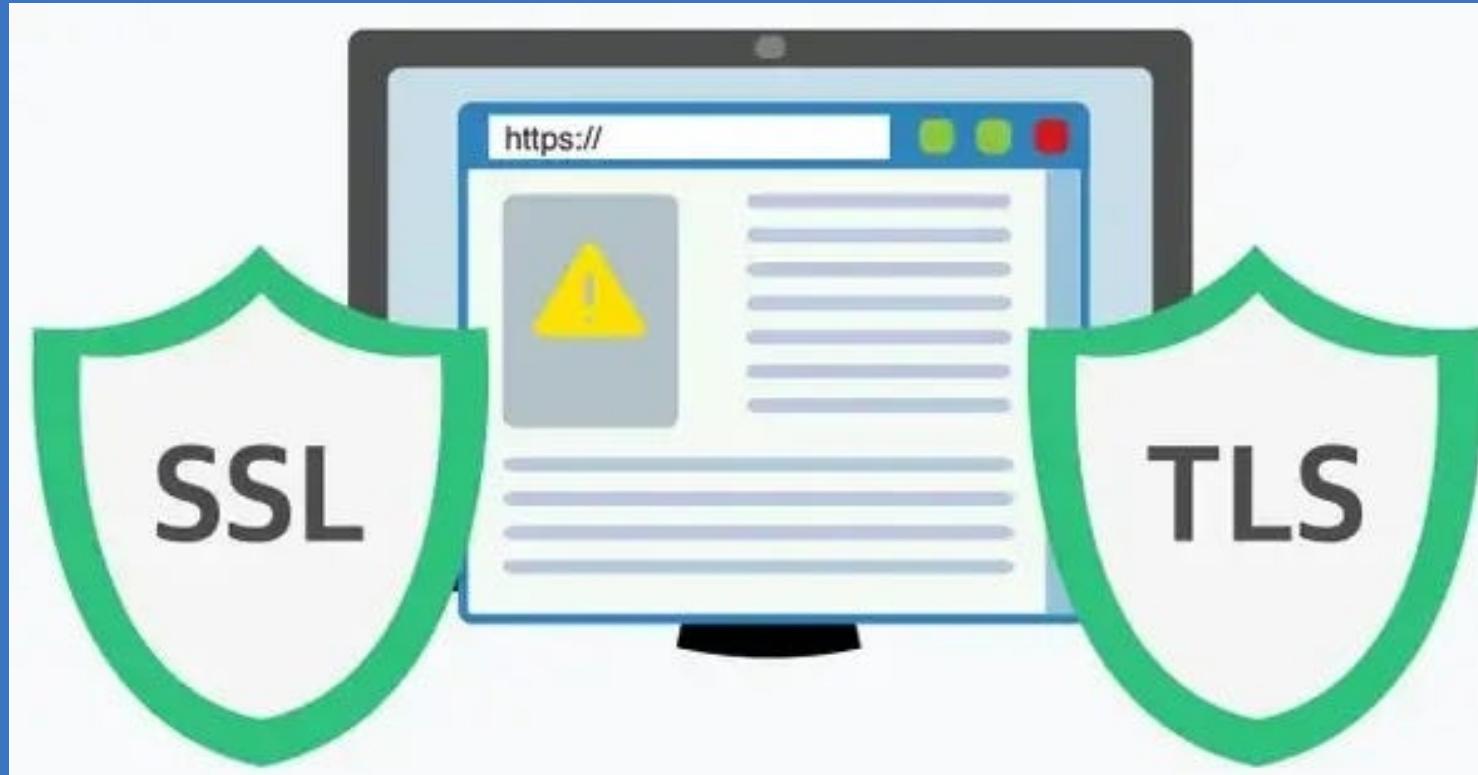
Travaille sur la couche applicative (HTTP/HTTPS).
Analyse le contenu des requêtes et réponses Web, capable de détecter SQL Injection , XSS, CSRF , Attaques sur sessions, cookies...



SSL/TLS

SÉCURISATION DES FLUX AVEC SSL/TLS

SÉCURISATION DES FLUX AVEC SSL/TLS



SSL/TLS : protocoles qui assurent la confidentialité, l'intégrité et l'authenticité des échanges.

Principaux objectifs :
Chiffrement des données
Authentification des serveurs (et parfois des clients)
Prévention de l'espionnage (Man-in-the-Middle)



RAPPELS CRYPTOGRAPHIQUES

- **Chiffrement asymétrique (RSA, ECC)** → échange des clés.
- **Chiffrement symétrique (AES)** → rapide pour chiffrer les données.
- **Fonctions de hachage (SHA-256, SHA-3)** → intégrité.
- **MAC / HMAC** → vérification des messages.
- **Handshake TLS** : négociation des algorithmes + génération clé de session.



GESTION DES CERTIFICATS X.509

Un certificat X.509 est comme une carte d'identité numérique utilisée pour sécuriser les communications entre un navigateur (client) et un serveur (site web), il contient plusieurs informations

importantes :

- Clé publique du serveur
- Nom du serveur (CN ou SAN)
- Autorité de certification (CA)



Les certificats ne sont valables que pour une période limitée (ex. 1 an).

Ils doivent être renouvelés régulièrement.



GESTION DES CERTIFICATS X.509

Quand tu te connectes à un site en HTTPS :

Le navigateur reçoit le certificat X.509.

Il vérifie :

- La validité (pas expiré).
- Le nom de domaine correspond.
- L'autorité de certification est reconnue.

Si tout est bon → tu vois le cadenas vert dans la barre d'adresse.

Émis par

Nom commun (CN)
Organisation (O)
Unité d'organisation (OU)

WE1
Google Trust Services
<Ne fait pas partie du certificat>

Durée de validité

Émis le
Expire le

mercredi 30 juillet 2025 à 03:20:29
mardi 28 octobre 2025 à 03:20:27



PRINCIPE DU DÉVELOPPEMENT SÉCURISÉ

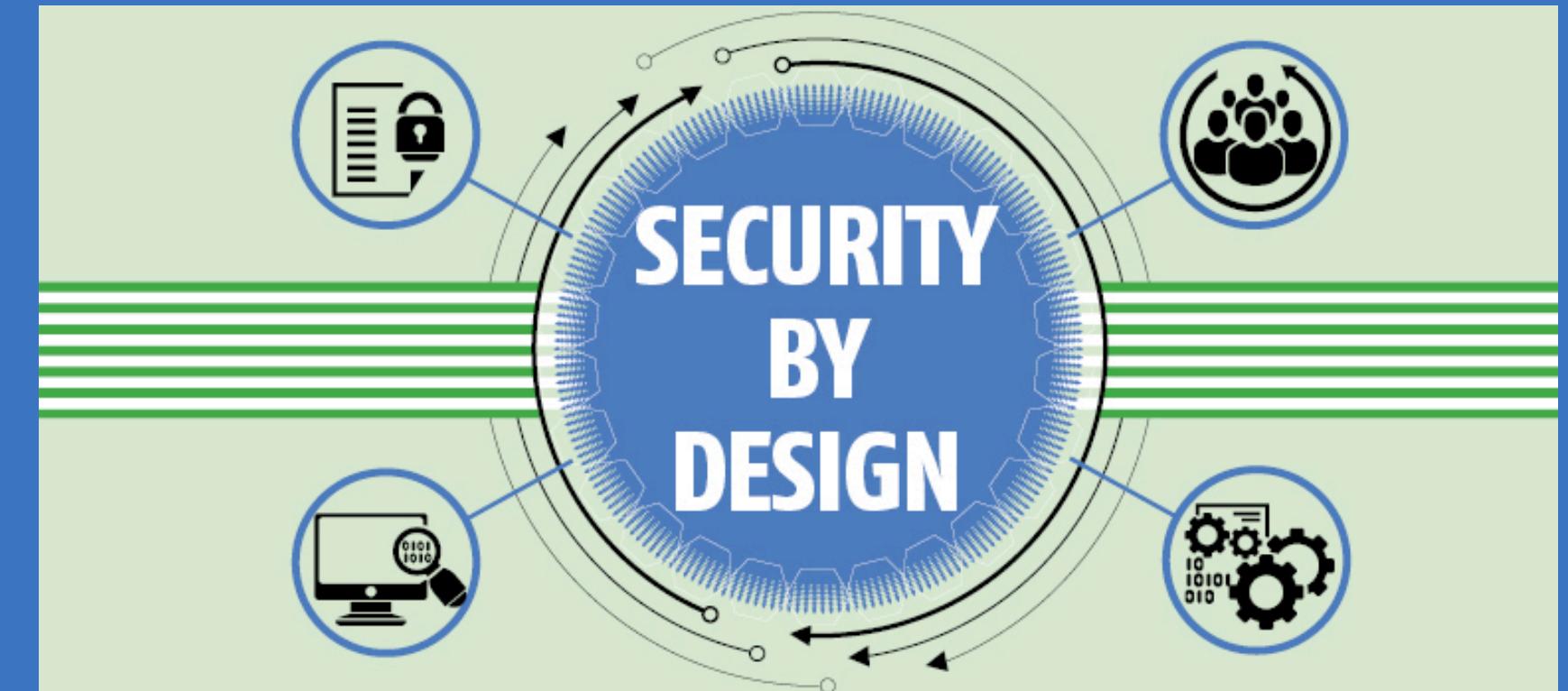
SÉCURITÉ DU DÉVELOPPEMENT : QUEL BUDGET ?



Corriger une faille en production coûte 10 à 30 fois plus cher que de la prévenir dès la conception.

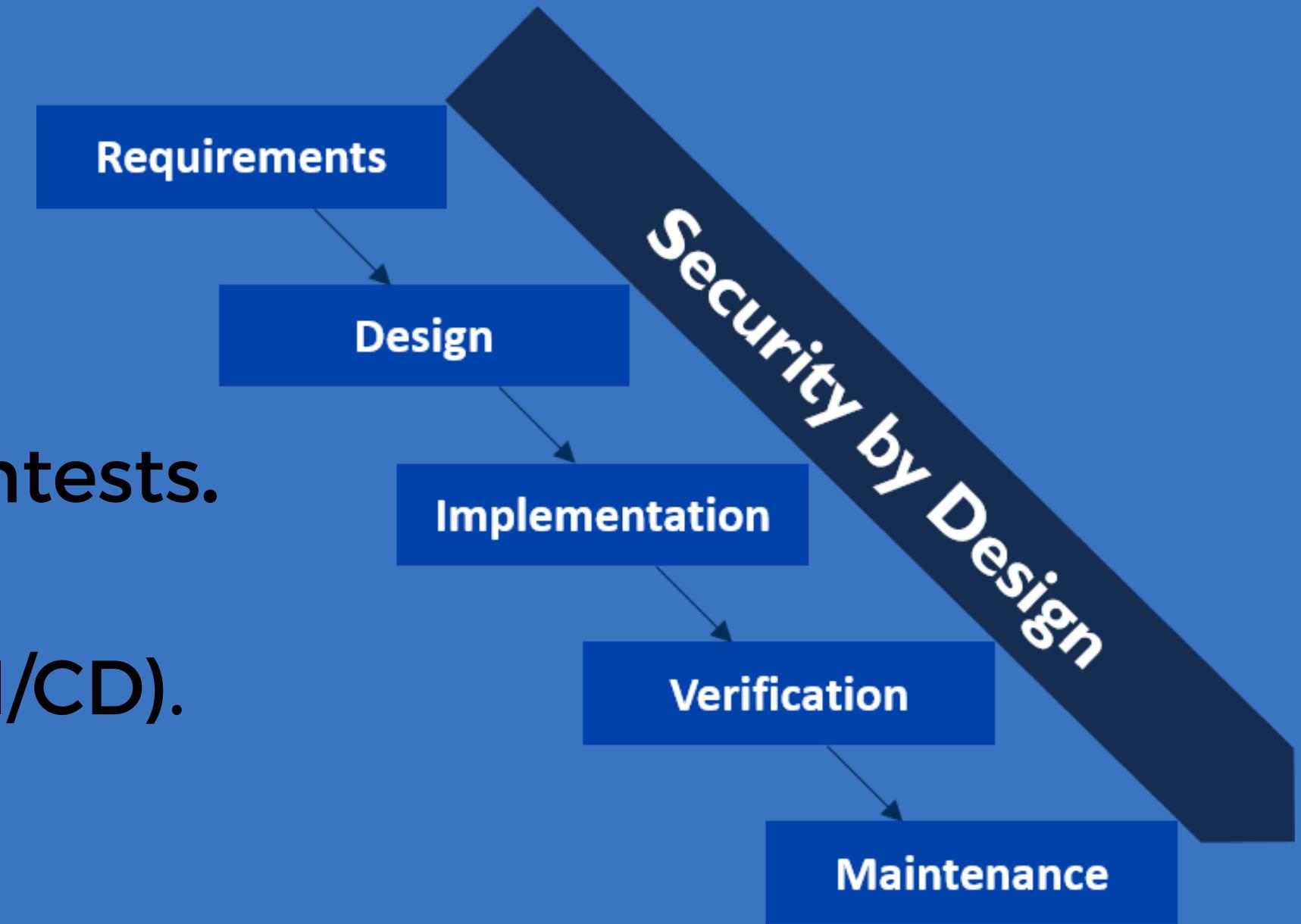
Investissement recommandé :

- Formations développeurs (sécurité applicative, OWASP).
- Intégration d'outils SAST/DAST (analyse de code, tests dynamiques).
- Audits et revues de code réguliers.



LA SÉCURITÉ DANS LE CYCLE DE DÉVELOPPEMENT

- **Design** : identification des menaces (Threat Modeling).
- **Développement** : règles de codage sécurisé.
- **Tests** : tests unitaires + fuzzing + pentests.
- **Déploiement (DevSecOps)** : scans automatiques à chaque livraison (CI/CD).
- **Exploitation** : supervision, patch management, logs et SIEM.



LE RÔLE DU CODE CÔTÉ CLIENT : SÉCURITÉ OU ERGONOMIE ?

Le code client (JavaScript/HTML/CSS) est visible et modifiable par l'utilisateur.

Il sert surtout à l'ergonomie (UX, rapidité).

Toute logique de sécurité doit être côté serveur (authentification, contrôle d'accès, validation définitive).



Règle d'or : "Ne jamais faire confiance au client."

Contrôle des données envoyées par le client

Les entrées utilisateur sont la porte d'entrée principale des attaques (XSS, SQLi, LDAPi).



Vérifier le type (int, string, email).

Définir des tailles maximales (ex. mot de passe \leq 64 caractères).

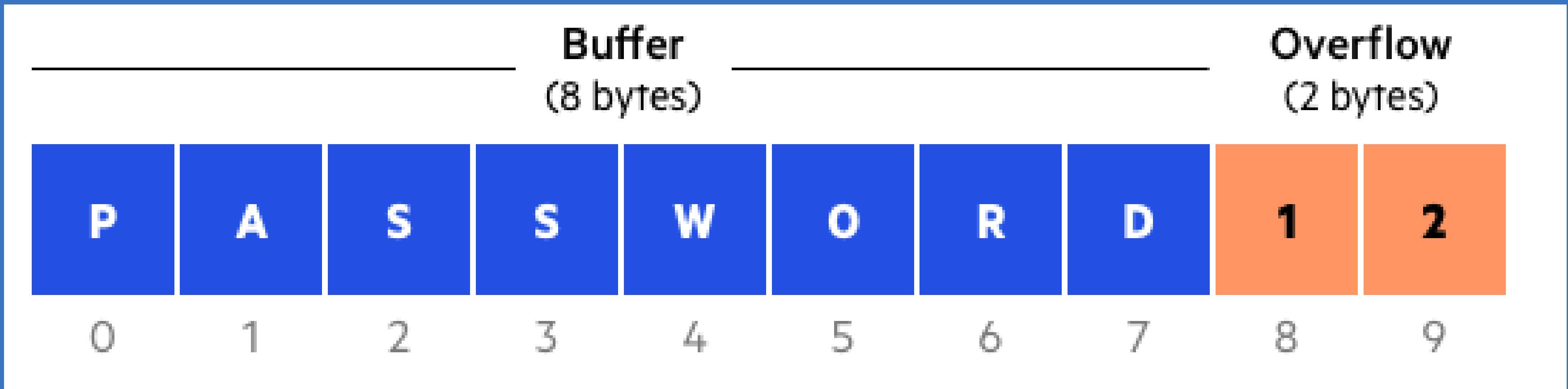


Utiliser des listes blanches (caractères autorisés).

Exemple : champ “âge” \rightarrow n’accepter que des chiffres de 0 à 120.

LUTTER CONTRE LES ATTAQUES DE TYPE BUFFER OVERFLOW

Principe : écrire plus de données que prévu → écrase la mémoire → permet injection de code.



LUTTER CONTRE LES ATTAQUES DE TYPE BUFFER OVERFLOW

Prévention :



- Utiliser des fonctions sécurisées (strncpy au lieu de strcpy en C).
- Limiter la taille des entrées.
- Langages modernes protégés par défaut (Java, Python).
- Activer protections système (ASLR, DEP).

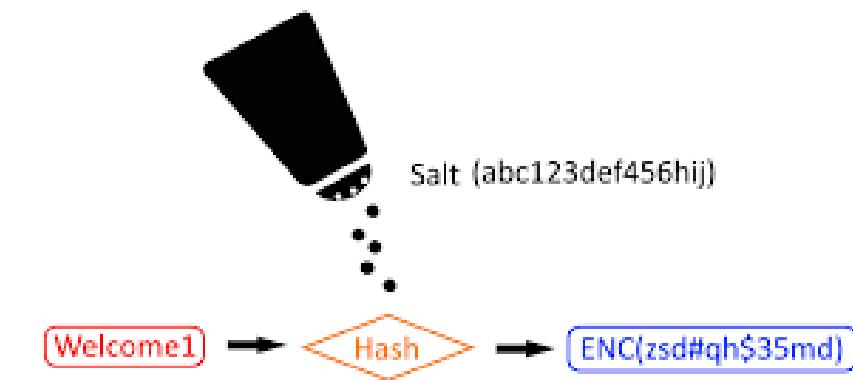
LES RÈGLES DE DÉVELOPPEMENT À RESPECTER

ZERO TRUST
ARCHITECTURE

NEVER TRUST,
ALWAYS VERIFY

OWASP/ASVS

Application Security Verification Standard



- Valider toutes les entrées (never trust, always verify).
- Ne jamais stocker de mots de passe en clair → hash + salt.
- Limiter les messages d'erreur (ne pas divulguer d'infos techniques).
- Séparer les environnements (dev/test/prod).
- Respecter les guides : OWASP ASVS, CERT Secure Coding.

COMMENT LUTTER CONTRE LES RISQUES RÉSIDUELS ?

Headers HTTP sécurisés :

- CSP (Content Security Policy) → bloque scripts non autorisés.
- HSTS → force l'utilisation de HTTPS.
- X-Frame-Options → bloque le clickjacking.
- URLs malformées : validation stricte côté serveur.
- Cookie Poisoning : cookies signés et protégés par HttpOnly, Secure, SameSite.



L'AUTHENTIFICATION DES UTILISATEURS

AUTHENTIFICATION VIA HTTP

Basic Authentication

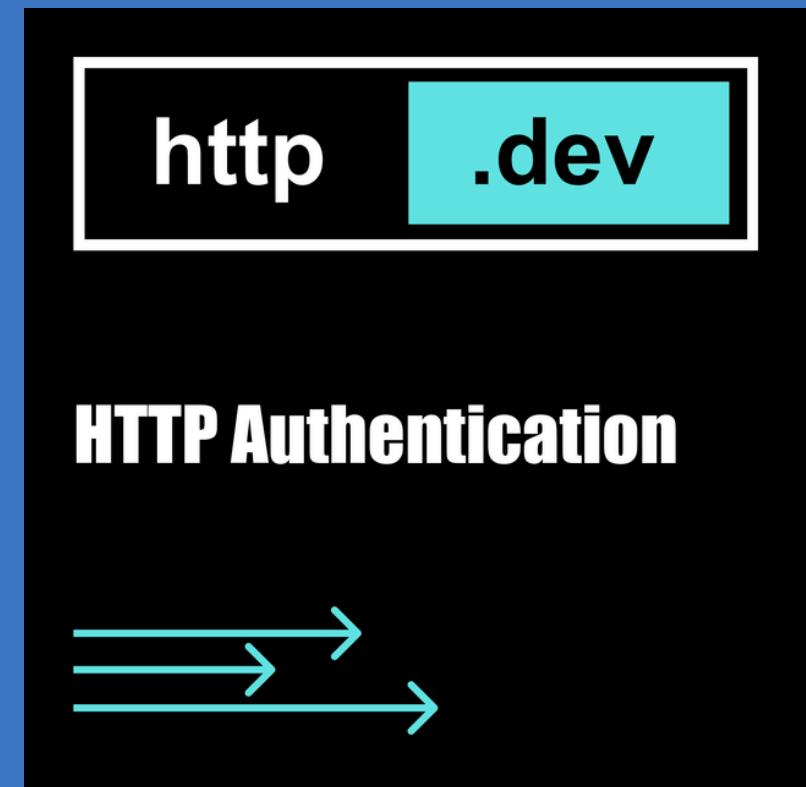
- Login + mot de passe envoyés en Base64 dans l'en-tête HTTP.
- Très faible sécurité → doit être utilisé uniquement en HTTPS.

Digest Authentication

- Amélioration du Basic : le mot de passe est haché avant envoi.
- Plus sûr que Basic, mais obsolète aujourd'hui.

Formulaire HTML (login/password)

- Le plus courant (sites web, e-commerce, réseaux sociaux).
- La sécurité dépend du protocole utilisé (jamais en HTTP simple).



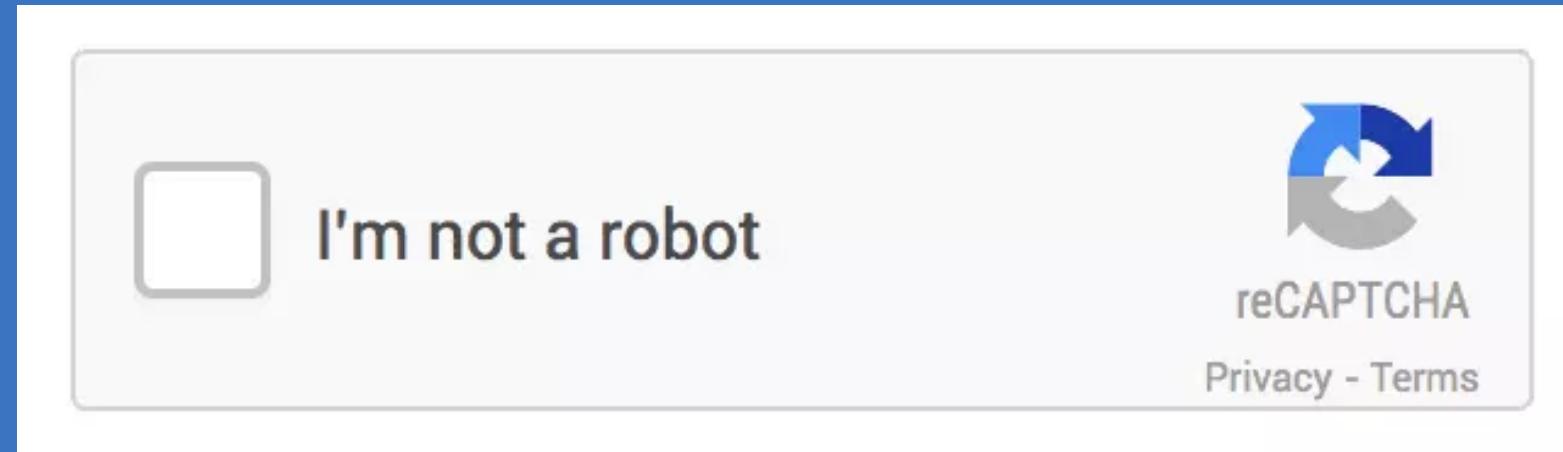
AUTHENTIFICATION FORTE (2FA / MFA)

- **Certificat X.509 client** : carte d'identité numérique installée côté utilisateur.
- **Token matériel (RSA SecurID)** : code généré toutes les 60 secondes.
- **Biométrie / ADN digital Mobilegov** : empreintes, reconnaissance faciale ou mobile ID.



AUTRES TECHNIQUES LOGICIELLES

- **CAPTCHA** : prouve qu'on est humain (bloque les bots).



- **OTP (One Time Password)** : mot de passe unique valable quelques secondes (Google Authenticator).

An interface for entering a One Time Password (OTP). At the top, a red horizontal bar displays the error message "Invalid OTP!". Below this, the text "Enter OTP" is centered in a large, bold, dark blue font. Underneath, a green note says "Check your email for the OTP". At the bottom, there is a light blue rectangular button with the text "One Time Password" and a blue rectangular button with the text "Submit".

ATTAQUES SUR LES MOTS DE PASSE

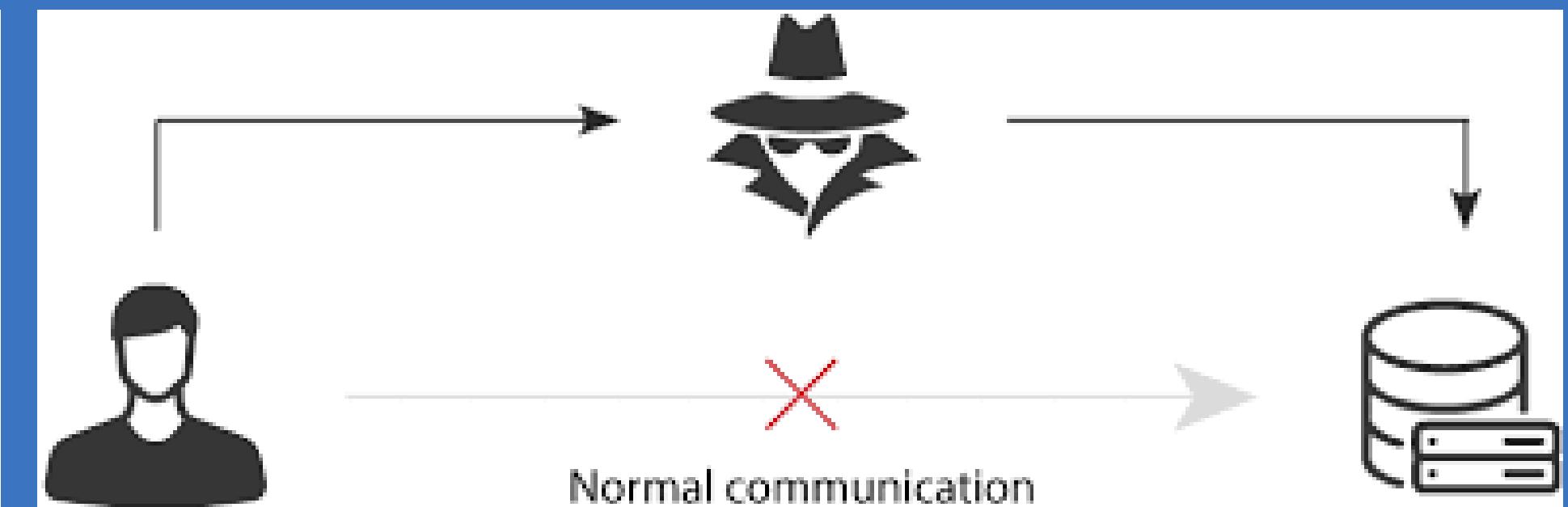
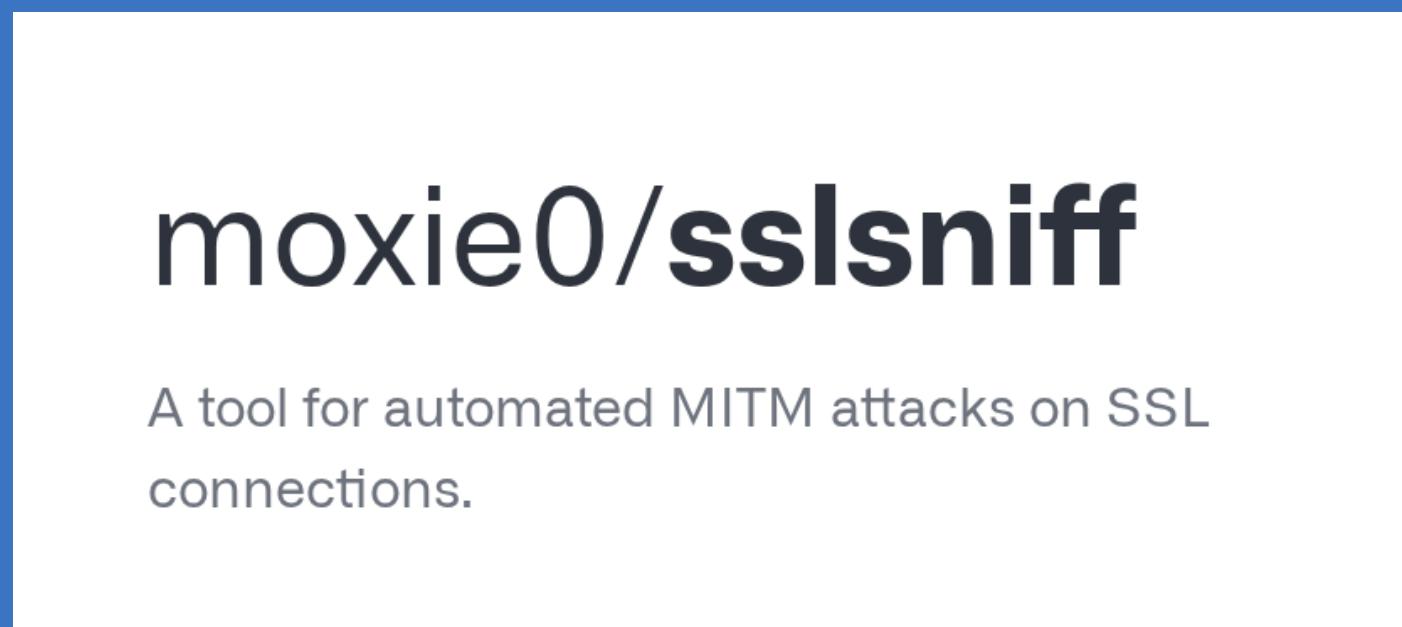
- **Sniffing** : capture du trafic réseau si pas de HTTPS.
- **Brute force** : tester automatiquement toutes les combinaisons.
- **Phishing** : tromper l'utilisateur avec une fausse page de login.
- **Keylogger** : logiciel espion qui enregistre les frappes clavier.



```
[80][http-get-form] host: 192.168.100.155 login: admin password: password
[80][http-get-form] host: 192.168.100.155 login: admin password: p@ssword
[80][http-get-form] host: 192.168.100.155 login: admin password: 12345
[80][http-get-form] host: 192.168.100.155 login: admin password: 1234567890
[80][http-get-form] host: 192.168.100.155 login: admin password: Password
[80][http-get-form] host: 192.168.100.155 login: admin password: 123456
[80][http-get-form] host: 192.168.100.155 login: admin password: 1234567
[80][http-get-form] host: 192.168.100.155 login: admin password: 12345678
[80][http-get-form] host: 192.168.100.155 login: admin password: 1q2w3e4r
[80][http-get-form] host: 192.168.100.155 login: admin password: 123
[80][http-get-form] host: 192.168.100.155 login: admin password: 1
[80][http-get-form] host: 192.168.100.155 login: admin password: 12
1 of 1 target successfully completed, 12 valid passwords found
Hydra (http://www.thc.org/thc-hydra) finished at 2017-07-27 15:28:24
```

ATTAQUES SUR LES AUTHENTIFICATIONS HTTPS

- **Fake server** : un attaquant se fait passer pour le vrai serveur (attaque Man-in-the-Middle).
- **sslsniff** : outil qui intercepte et manipule les connexions SSL/TLS.
- **Exploits sur certificats X.509** : certificats compromis ou émis par une autorité non fiable (ex. DigiNotar hack 2011).





CONCLUSION

La sécurité n'est pas une option, mais un processus continu qui doit être intégré dès la conception, testé régulièrement, et renforcé en production.

Une application Web n'est sécurisée que si tous les maillons (réseau, serveur, code, utilisateur) sont protégés et surveillés.