

# Sécurité web

Mohamed Akheraz & Ahmed Lounis

## Quelques rappels

- stockage des informations côté client
- durée de vie longue
- envoyés dans les entêtes HTTP

## Côté client (javascript) :

```
function setCookie(nom, val, nbJours) {  
    var fin = new Date();  
    fin.setTime(fin.getTime() + (nbJours * 24 * 60 * 60 * 1000));  
    document.cookie = nom + "=" + val + "; expires=" + fin.toGMTString() + "; path="/;  
}  
  
function getCookie(nomCookie) {  
    var mesCookies = document.cookie.split(';');  
    for(var i=0; i<mesCookies.length; i++) {  
        var unCookie = mesCookies[i].trim();  
        if (unCookie.indexOf(nomCookie + "=") == 0)  
            return unCookie.substring(nomCookie.length+1, unCookie.length);  
    }  
    return null;  
}
```

# Cookies côté serveur (ex. node)

167

- npm install cookie-parser

```
var express = require('express')
var cookieParser = require('cookie-parser')
var app = express()
app.use(cookieParser())
app.get('/', function (req, res) {
  // Cookies that have not been signed
  console.log('Cookies: ', req.cookies)

  // Cookies that have been signed
  console.log('Signed Cookies: ', req.signedCookies);

  //Delete the saved cookies
  res.clearCookie();

  //Set a cookie
  res.cookie(`attribute name`, `value`);
})
```

- stockage des informations sur le serveur
- durée de vie courte
- un identifiant de session est échangé entre le serveur et le client

# Sessions (node)

169

```
// créer une session
app.post('/user',(req,res) => {
  if(req.body.email == myemail && req.body.password == mypassword){
    var session=req.session;
    session.userid = req.body.email;

    console.log(req.session)
    res.send(`click to logout`);
  }
  else{
    res.send('Invalid email or password');
  }
})
```

```
// utiliser la session
app.get('/',(req,res) => {
  session=req.session;
  if(session.userid){
    res.send("Welcome User click to logout");
  }else
    res.redirect('/login.html');
});
```

```
// fermer la session
app.get('/logout',(req,res) => {
  req.session.destroy();
  res.redirect('/');
});
```

# Types d'attaques/vulnérabilités fréquentes

170

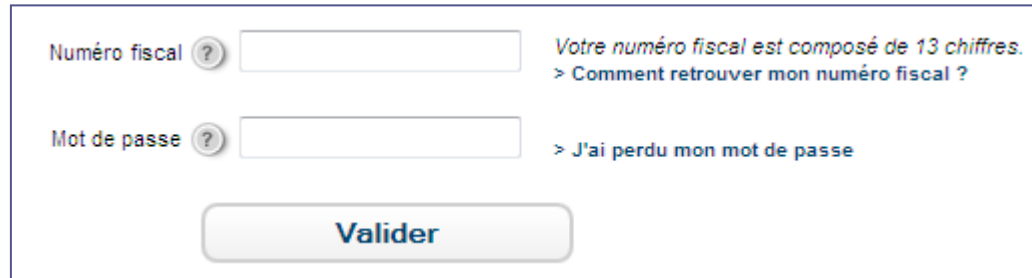
- ❑ Injection
- ❑ Violation de gestion d'authentification
- ❑ Violation de gestion de session
- ❑ Cross-Site Scripting (XSS)
- ❑ Violation de contrôle d'accès
- ❑ Mauvaise configuration de sécurité
- ❑ Exposition de données sensibles
- ❑ Falsification de requête intersites (CSRF)
- ❑ Utilisation de composants avec des vulnérabilités connues
- ❑ Redirections et renvois non validés



*Une faille d'injection, telle l'injection SQL, OS et LDAP, se produit quand une donnée non fiable est envoyée à un interpréteur en tant qu'élément d'une commande ou d'une requête. Les données hostiles de l'attaquant peuvent duper l'interpréteur afin de l'amener à exécuter des commandes fortuites ou accéder à des données non autorisées.*

- Exploitabilité : **facile**
- Prévalence : **commune**
- Détection : **moyenne**
- Impact : **sévère**

## Exemple d'injection SQL



The image shows a login form with two input fields and a button. The first field is labeled 'Numéro fiscal' with a question mark icon and a hint: 'Votre numéro fiscal est composé de 13 chiffres. > Comment retrouver mon numéro fiscal ?'. The second field is labeled 'Mot de passe' with a question mark icon and a hint: '> J'ai perdu mon mot de passe'. Below the fields is a button labeled 'Valider'.

➤ SQL attendu :

```
select nom from usagers where numero='123456789' and password='abcdef'
```

➤ Code malicieux dans le champ password :

```
xxxx' or 'a'='a
```

➤ SQL frauduleux :

```
select nom from usagers where numero='987456' and password='xxxx' or 'a'='a'
```

- Le même procédé peut-être utilisé avec une requête LDAP :
  - Critère de recherche : (& (numero=12345) (pwd=abcd))
  - Code malicieux : (& (numero=\*) (pwd=\*))
  
- Ou encore dans un formulaire qui manipule des noms de fichier pour supprimer ou afficher le fichier :
  - Paramètre soumis par le formulaire : filename=toto.pdf
  - Code malicieux : filename=/rep1/rep2/fichier\_sensible.txt

# Violation de gestion d'authentification

175

*Les fonctions applicatives relatives à l'authentification ne sont parfois pas mises en œuvre correctement, permettant aux attaquants de compromettre les mots de passe ou d'exploiter d'autres failles d'implémentation pour s'approprier les identités d'autres utilisateurs.*

# Violation de gestion d'authentification

176

- Exploitabilité : **moyenne**
- Prévalence : **répandue**
- Détection : **moyenne**
- Impact : **grave**

# Violation de gestion d'authentification

177

Le contournement de l'authentification se fait le plus souvent par :

- vol des identifiants d'un utilisateur étourdi ou négligeant
- tromperie d'un utilisateur (mail, fausse hotline,...)
- utilisation de comptes par défaut de certaines applications
- détournement appli de changement de mot de passe
- génération aléatoire et massive de couple d'identifiants

# Violation de gestion de session

178

*Les fonctions applicatives relatives à la gestion de session ne sont pas mises en œuvre correctement, permettant aux attaquants de compromettre les jetons de session, ou d'exploiter d'autres failles d'implémentation pour s'approprier les identités d'autres utilisateurs.*



# Violation de gestion de session

179

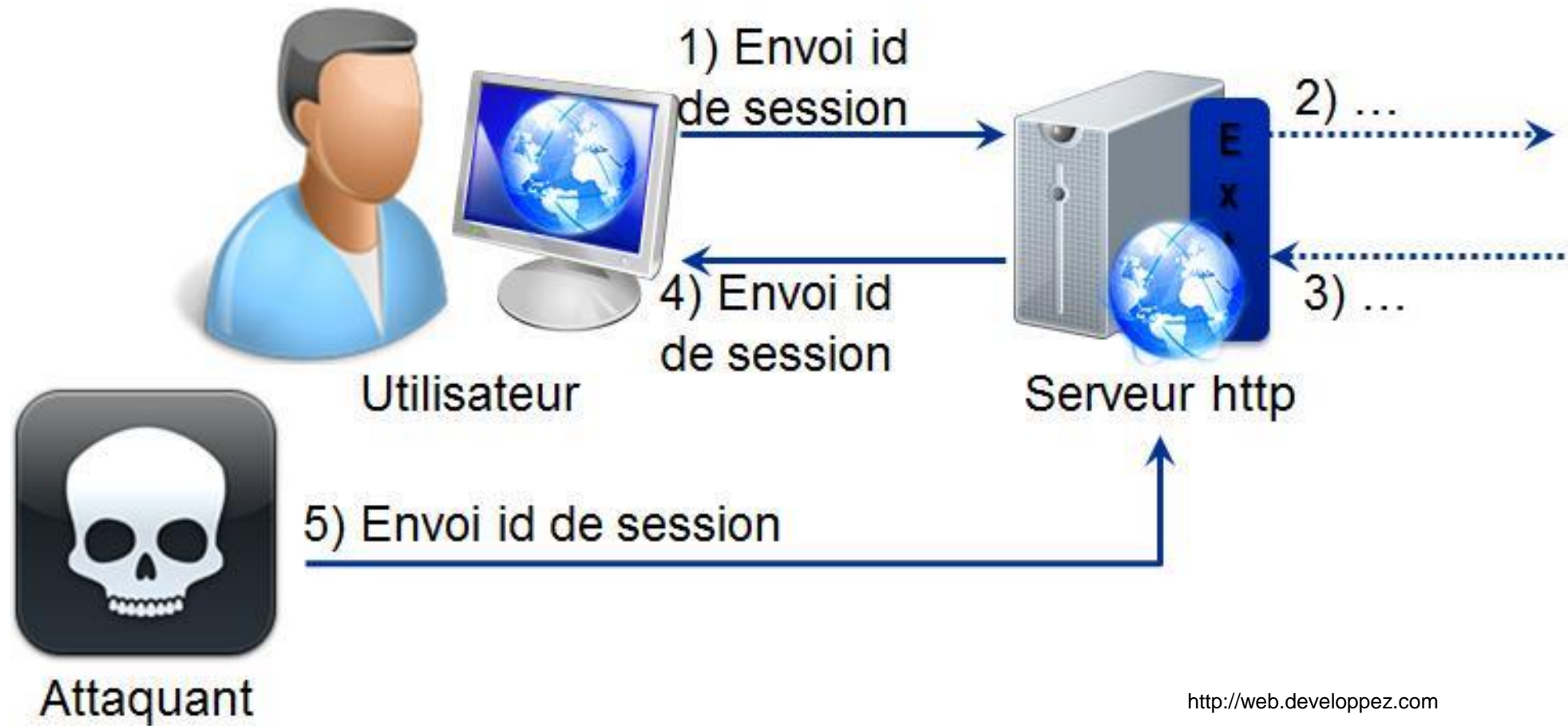
- Exploitabilité : **moyenne**
- Prévalence : **répandue**
- Détection : **moyenne**
- Impact : **grave**

L'usurpation de session se fait de plusieurs manières :

- génération aléatoire et massive d'identifiants de session
- récupération d'un identifiant valide par attaque XSS
- fixation d'un identifiant de session à un utilisateur

# Violation de gestion de session

181



# Cross-Site Scripting (XSS)

182

*Les failles XSS se produisent chaque fois qu'une application accepte des données non fiables et les envoie à un browser web sans validation appropriée. XSS permet à des attaquants d'exécuter du script dans le navigateur de la victime afin de détourner des sessions utilisateur, défigurer des sites web, ou rediriger l'utilisateur vers des sites malveillants.*

# Cross-Site Scripting (XSS)

183

- Exploitabilité : **moyenne**
- Prévalence : **très répandue**
- Détection : **facile**
- Impact : **modéré**

## XSS réfléchi

- Attaque dite non permanente
- Se base sur la crédulité de l'utilisateur
- Par exemple, un mail contenant un lien frauduleux :

`http://www.sitevulnerable.fr/page?parametre=<script>window.open('www.sitepirate.com');</script>`

## XSS stocké

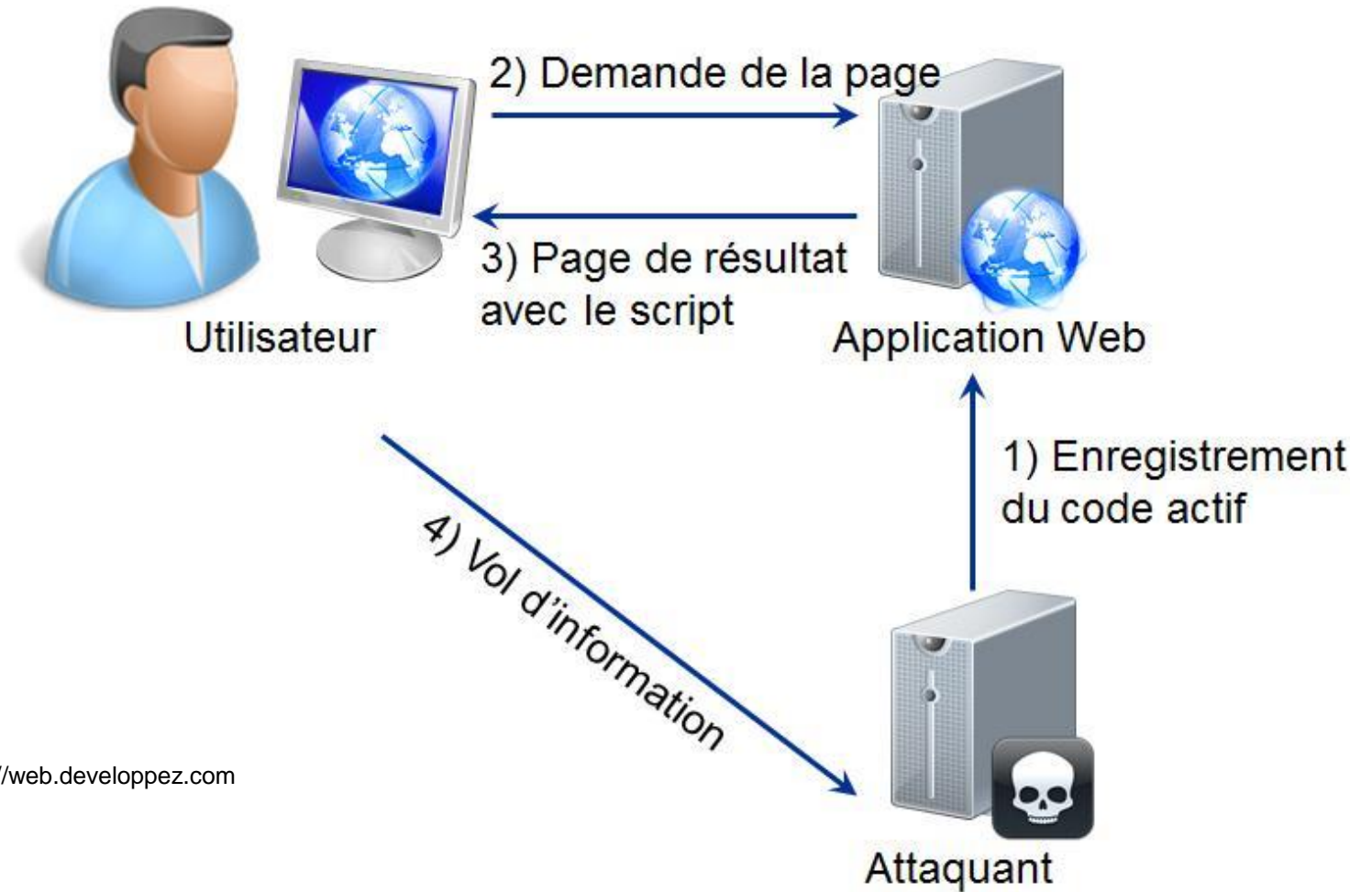
- Attaque dite permanente
- Profite de la faille du site vulnérable pour toucher un grand nombre d'utilisateurs
- Par exemple, un message dans un forum par le hacker et retransmis tel quel à tous les autres visiteurs du forum :

Bonjour, voici un message posté sur un forum vulnérable, les autres utilisateurs ne voient pas ceci :  
`<script>alert('attaque XSS');</script>`

# Cross-Site Scripting (XSS)

186

## Schéma XSS stocké :





# Violation de contrôle d'accès

187

- *Référence directe non sécurisée : un développeur expose une référence à un objet d'exécution interne, tel un fichier, un dossier, un enregistrement de base de données ou une clé de base de données. Sans un contrôle d'accès ou autre protection, les attaquants peuvent manipuler ces références pour accéder à des données non autorisées.*
- *Manque de contrôle d'accès au niveau fonctionnel : les applications doivent vérifier les droits d'accès au niveau fonctionnel sur le serveur lors de l'accès à chaque fonction. Si les demandes ne sont pas vérifiées, les attaquants seront en mesure de forger des demandes afin d'accéder à une fonctionnalité non autorisée.*

# Violation de contrôle d'accès

188

- Exploitabilité : **facile**
- Prévalence : **commune**
- Détection : **facile**
- Impact : **modéré**

## Référence directe non sécurisée

- exemple : récupérer un document confidentiel

➤ URL permettant d'accéder à un dossier autorisé :

`www.monsite.fr/afficherFicheDePaye?id_employe=12345`

➤ URL modifiée pour accéder à un autre dossier :

`www.monsite.fr/afficherFicheDePaye?id_employe=12388`

## Manque de contrôle d'accès au niveau fonctionnel

- Exemple 1 : accéder à une fonctionnalité réservée aux admin
  - URL présentée à un utilisateur sans privilege :  
`www.monsite.fr/accueil`
  - URL modifiée pour accéder à un autre dossier :  
`www.monsite.fr/admin/accueil`
- exemple 2 : supprimer des données
  - URL de consultation d'un dossier :  
`www.monsite.fr/gestionPers?id=1234&action=CONSULT`
  - URL modifiée pour supprimer le dossier :  
`www.monsite.fr/gestionPers?id=1234&action=DELETE`

# Mauvaise configuration sécurité

191

*Une bonne sécurité nécessite de disposer d'une configuration sécurisée définie et déployée pour l'application, contextes, serveur d'application, serveur web, serveur de base de données et la plate-forme. Tous ces paramètres doivent être définis, mis en œuvre et maintenus, car beaucoup ne sont pas livrés sécurisés par défaut. Cela implique de tenir tous les logiciels à jour.*

# Mauvaise configuration sécurité

192

- Exploitabilité : **simple**
- Prévalence : **commune**
- Détection : **facile**
- Impact : **modéré**

- Exemple 1 : console d'administration du serveur d'application non désactivée avec mot de passe par défaut
- Exemple 2 : listage des répertoires non désactivé

*Beaucoup d'applications web ne protègent pas correctement les données sensibles telles que les cartes de crédit, identifiants d'impôt et informations d'authentification. Les pirates peuvent voler ou modifier ces données faiblement protégées pour effectuer un vol d'identité, de la fraude à la carte de crédit ou autres crimes. Les données sensibles méritent une protection supplémentaire tel un chiffrement statique ou en transit, ainsi que des précautions particulières lors de l'échange avec le navigateur.*



# Exposition de données sensibles

195

- Exploitabilité : **difficile**
- Prévalence : **peu commune**
- Détection : **moyenne**
- Impact : **sévère**

- Toutes les briques de l'architecture de l'application WEB sont concernées : client, serveur WEB, serveur d'application, SGBD, etc....
- Il est impératif de sécuriser tous les composants, au risque de rendre vulnérable l'ensemble de l'application
- Il faut sécuriser toutes les pages concernées par le transport d'informations; sécuriser la page d'authentification par exemple ne suffit pas

# Falsification de requête intersites (CSRF)

197

*Une attaque CSRF (Cross Site Request Forgery) force le navigateur d'une victime authentifiée à envoyer une requête HTTP forgée, comprenant le cookie de session de la victime ainsi que toute autre information automatiquement incluse, à une application web vulnérable. Ceci permet à l'attaquant de forcer le navigateur de la victime à générer des requêtes dont l'application vulnérable pense qu'elles émanent légitimement de la victime.*

# Falsification de requête intersites (CSRF)

198

- Exploitabilité : **moyenne**
- Prévalence : **courant**
- Détection : **facile**
- Impact : **modéré**

# Falsification de requête intersites (CSRF)

199

## Exemple : attaque CSRF contre un site de publication de news

1. l'attaquant réussit à connaître la commande et les paramètres pour publier des news, et compose alors un script de publication de news malveillante :

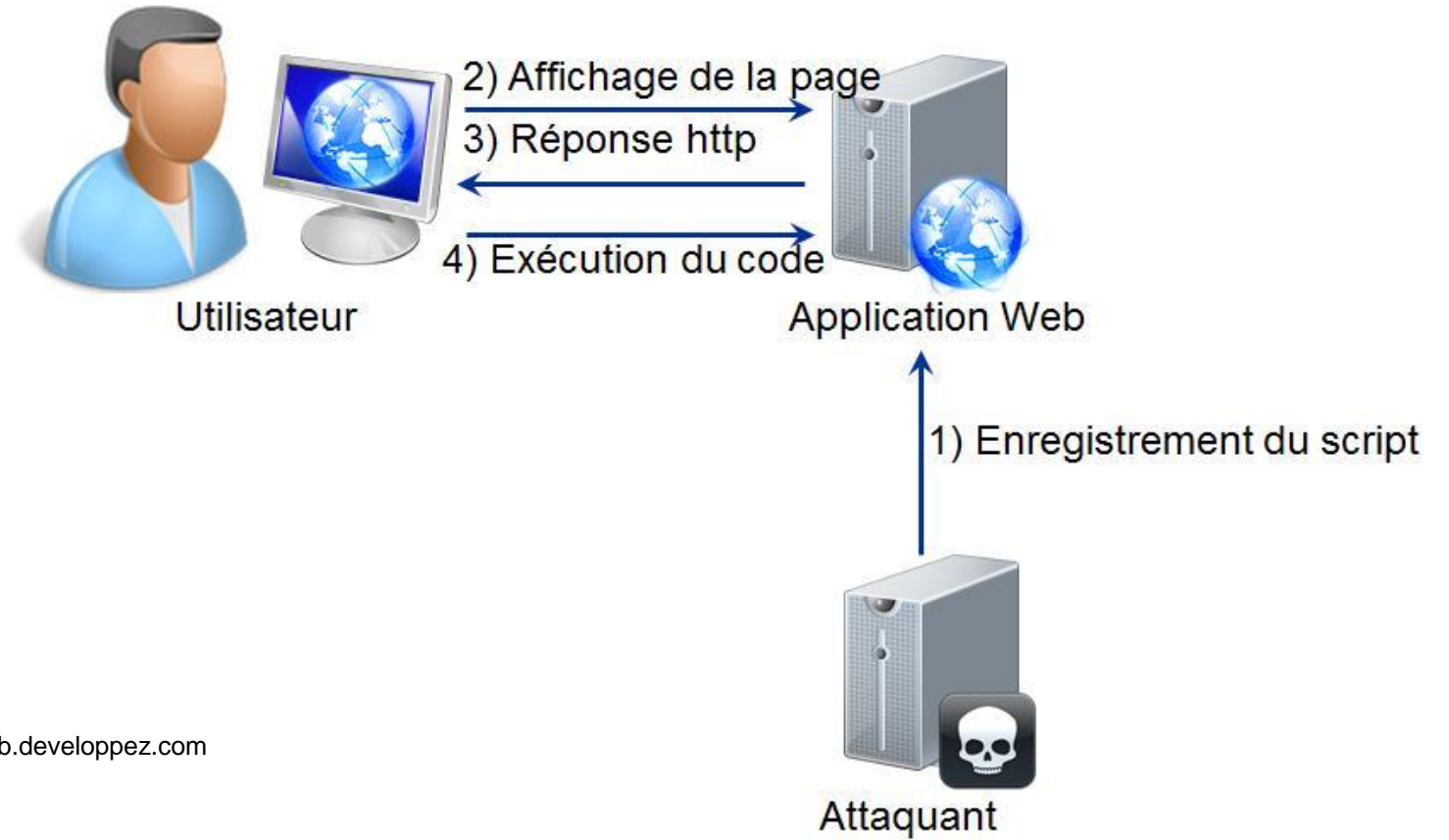
```
<form method="GET" name="attaque CSRF" action="ajouter_news.php">  
  <input type="hidden" name="auteur" value="hackerCSRF">  
  <input type="hidden" name="message" value="Attaque CSRF réussie !">  
</form>  
<script>document.attaque_CSRF.submit()</script>
```

2. il envoie un mail ou présente un site au frauduleux à un utilisateur qui peut se connecter au site
3. l'utilisateur clique sur le lien du mail ou se rend sur le site frauduleux, s'il est connecté au site de news, la news malveillante est alors publiée

# Falsification de requête intersites (CSRF)

200

## Schéma CSRF stocké



*Les composants vulnérables, tels que bibliothèques, contextes et autres modules logiciels fonctionnent presque toujours avec des privilèges maximum. Ainsi, si exploités, ils peuvent causer des pertes de données sérieuses ou une prise de contrôle du serveur. Les applications utilisant ces composants vulnérables peuvent compromettre leurs défenses et permettre une série d'attaques et d'impacts potentiels.*

- Exploitabilité : **moyenne**
- Prévalence : **courant**
- Détection : **difficile**
- Impact : **modéré**



On trouve par exemple comme cas de vulnérabilité :

- des comptes par défaut dont les mots de passe ne sont pas changés
- des anomalies connues du grand public et qui nécessitent un correctif
- des failles découvertes par les hackers dans les produits open-source

# Redirections et renvois non validés

204

*Les applications web réorientent et redirigent fréquemment les utilisateurs vers d'autres pages et sites internet, et utilisent des données non fiables pour déterminer les pages de destination. Sans validation appropriée, les attaquants peuvent réorienter les victimes vers des sites de phishing ou de malware, ou utiliser les renvois pour accéder à des pages non autorisées.*

# Redirections et renvois non validés

205

- Exploitabilité : **moyenne**
- Prévalence : **rare**
- Détection : **facile**
- Impact : **modéré**

# Redirections et renvois non validés

206

- un script de redirection appelé fwd.php:

```
<?php  
echo '<html><head>';  
echo '<meta http-equiv="refresh" content="0; url=' . $_GET['target'] . '" />';  
echo '</head>body>Merci de patienter...</body></html>';  
?>
```

- un lien utilisé pour tromper l'utilisateur et le rediriger vers un site malveillant :

<https://www.utc.fr/fwd.php?target=www.hijacking.com>

## Bonnes pratiques

# Bonnes pratiques

208

- ☐ Contrôler les données envoyées et reçues
- ☐ Requêtes paramétrées
- ☐ Gestion des mots de passe
- ☐ Authentification rigide
- ☐ Gestion des sessions
- ☐ Chiffrement des données sensibles
- ☐ Protection des cookies
- ☐ Protection des opérations sensibles
- ☐ Contrôle des redirections
- ☐ Bien configurer ses composants

## Contrôle des champs envoyés (client)

- Valeurs attendues
  - listes déroulantes, boutons option, etc...
  - contrôle javascript (longueur, type, intervalle de valeur, format spécifique)
- Caractères interdits
  - expressions régulières
  - contrôle javascript

## Contrôle des champs reçus (serveur)

- Echappement des caractères autorisés avant stockage et avant l'affichage
- Supprimer les caractères interdits



# Requêtes paramétrées

211

## □ Exemple

```
let sql = "SELECT * FROM users WHERE username = ? And password = ?";
con.query(sql, [userName, password], function (err, result) {
    if (err) throw err;
    console.log(result);
});
```

- Politique de mots de passe forts
  - longueur
  - complexité
- Processus de récupération de mot de passe sécurisé = être sûr que c'est la bonne personne qui récupère le mot de passe

Pour gêner les attaques par essais aléatoires :

- verrouillage après N tentatives
- message d'erreur d'authentification non explicite

Assurer la pérennité de l'authentification :

- pouvoir savoir à n'importe quel moment qui est authentifié
- toutes les pages doivent être soumises à authentification

La gestion des habilitations :

- chaque utilisateur ne doit pouvoir accéder qu'à un périmètre précis de l'application
- contrôler les habilitations à chaque page

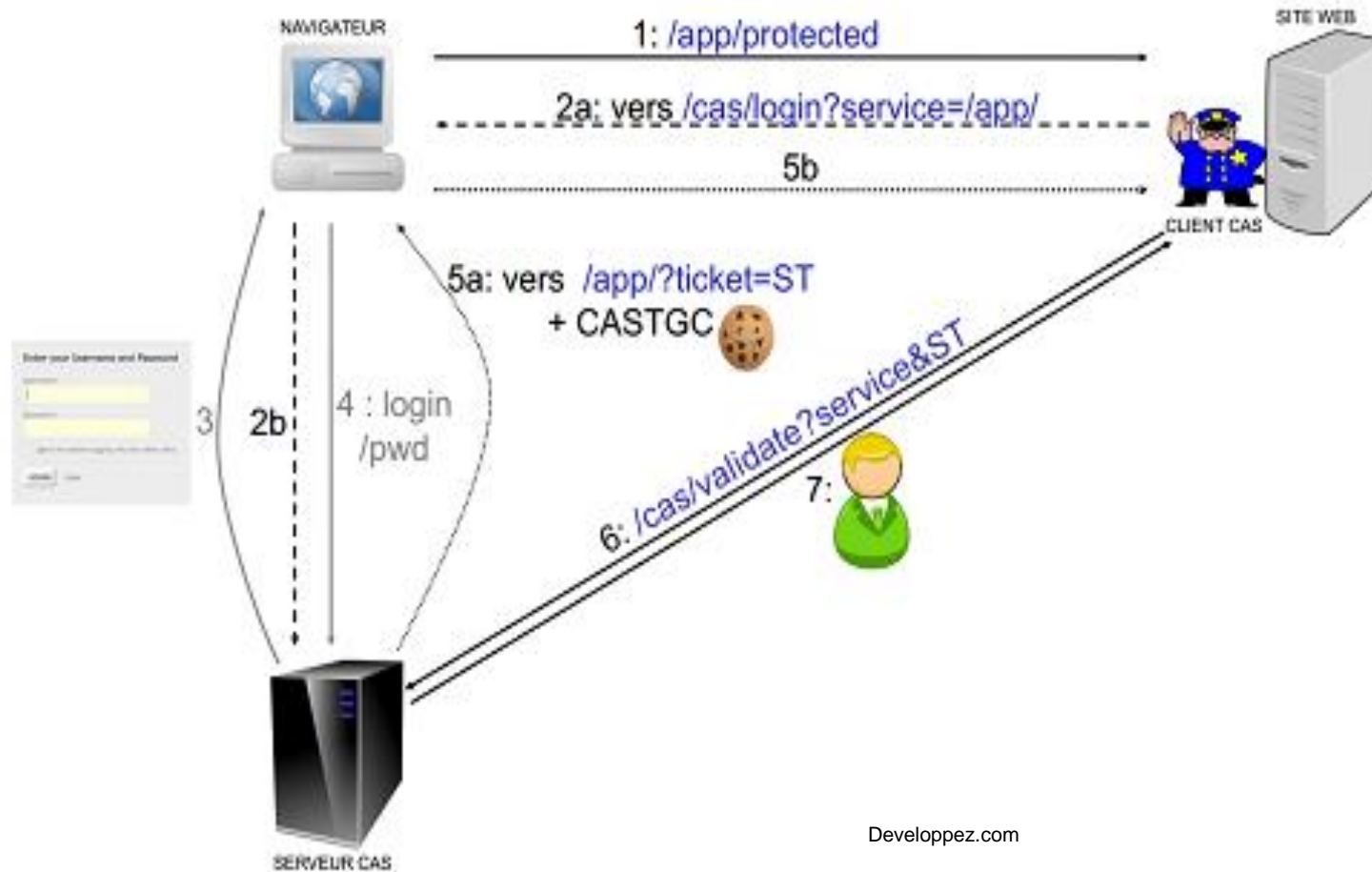
## L'authentification unique (SSO) :

- L'utilisateur n'a qu'un seul mot de passe à retenir
- L'utilisateur ne saisit qu'une fois ses identifiants
- Outil d'authentification unique
- Simplification de la généralisation de l'authentification

## A noter tout de même au sujet du SSO :

- en cas de vols d'identifiants, les risques sont multipliés
- SSO ne dispense pas des mesures de sécurité pour protéger chacun des sites qui y recourt

## Exemple de SSO : le CAS



Developpez.com

- diminuer le temps d'inactivité au minimum
- détruire la session de manière systématique
- pour les fonctionnalités critiques, ajouter un jeton unique pour l'utilisateur (stocké en session ou même persisté)

- cryptage au niveau du stockage :
  - base de données
  - annuaire
  - fichier
- cryptage pendant le transport (SSL)
- utilisation d'algorithmes forts et reconnus

## Utiliser la directive HTTPOnly

### Exemple node :

- ```
app.use(session({  
  secret: "notagoodsecretnoreallydontusethisone",  
  resave: false,  
  saveUninitialized: true,  
  cookie: {httpOnly: true, secure: true}  
}));
```
- ```
res.cookie("secureCookie","donnée secrete", {  
  secure: true, // true si on utilise TLS/SSL sinon false  
  httpOnly: true,  
  expires: dayjs().add(30, "days").toDate(),  
});
```



Redemander à l'utilisateur de s'authentifier s'il doit effectuer une opération considérée come particulièrement sensible :

- effectuer un ordre de virement bancaire
- changer un mot de passe administrateur
- etc...

## Les redirections doivent être contrôlées :

- Pas de construction dynamique de la redirection
- Liste exhaustive dans le programme

- Mises à jour régulières
- Modifier tous les mots de passe
- Désactiver les fonctionnalités non utilisées
- Bien gérer ses paramètres : durée de session, taille max de l'upload, écriture du SESSID dans l'URL, etc...

# Bonnes pratiques node.js

223

<https://nodejs.org/en/docs/guides/security>

- ❑ Denial of Service of HTTP server (CWE-400)
  - ▣ Use a reverse proxy to receive and forward requests to the Node.js application
  - ▣ Correctly configure the server timeouts
  - ▣ Limit the number of open sockets per host and in total.
- ❑ DNS Rebinding (CWE-346)
- ❑ HTTP Request Smuggling (CWE-444)
- ❑ Information Exposure through Timing Attacks (CWE-208)
- ❑ Etc

- Nécessité de veille et de formation permanente
- OWASP : un projet qui facilite le travail du développeur