Programmation orientée sécurité : introduction au contexte web

Olivier Levillain

M2 FIIL 2018-2019

Avant-propos

- Les supports de cours seront disponibles sur `http://paperstreet.picty.org/POS/`
- En cas de question, n'hésitez-pas
  - pendant le cours, à m'interrompre,
  - plus tard, à m'envoyer un courrier électronique à `cours-POS@picty.org`

Contexte

Aujourd'hui, HTTP est partout. On pense naturellement à l'ensemble des usages classiques du web depuis un navigateur :

- sites d'information
- commerce en ligne
- réseaux sociaux

Contexte

Aujourd'hui, HTTP est partout. On pense naturellement à l'ensemble des usages classiques du web depuis un navigateur :

- sites d'information
- commerce en ligne
- réseaux sociaux

Le rêve du client léger universel a fait fleurir de nombreux usages

- utilisation de *web services* pour les applications mobiles
- interfaces d'administration sur de nombreux équipements...
- jusque dans les CPU (technologie AMT)

Contexte

Aujourd'hui, HTTP est partout. On pense naturellement à l'ensemble des usages classiques du web depuis un navigateur :

- sites d'information
- commerce en ligne
- réseaux sociaux

Le rêve du client léger universel a fait fleurir de nombreux usages

- utilisation de *web services* pour les applications mobiles
- interfaces d'administration sur de nombreux équipements...
- jusque dans les CPU (technologie AMT)

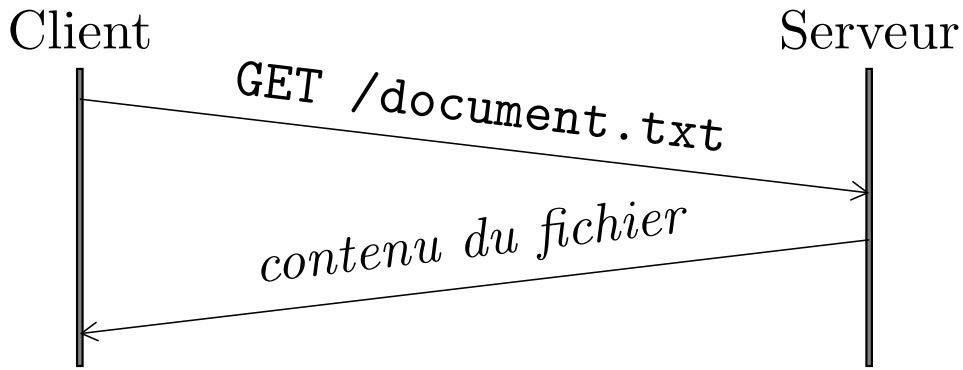
Il semble donc pertinent de s'interroger sur la sécurité des développements liés au web

- attention : ceci n'est qu'une introduction à un sujet vaste

Origine du web au CERN

- Créé en 1989 par des chercheurs du CERN (Tim Berners-Lee)
- HTTP : *HyperText Transfer Protocol* pour échanger des fichiers en utilisant par défaut le port TCP 80
- HTML : *HyperText Markup Language* pour écrire des documents enrichis avec liens hypertextes

- Exemple d'échange HTTP v0.9 :



Normalisation d'HTTP par l'IETF

- HTTP 1.0 (RFC 1945)
- HTTP 1.1 (RFC 7230 à 7235)
- HTTP 2.0 (RFC 7540)

- le langage HTML maintenu à part par le W3C (avec les difficultés qu'on connaît pour suivre la course à la fonctionnalité des navigateurs)

- Ces révisions d'HTTP ont apporté respectivement
  - un format plus clair des requêtes et des réponses
  - la possibilité d'enchaîner plusieurs requêtes dans une même connexion TCP (*pipeline*)
  - le multiplexage des connexions et la compression des en-têtes

Exemple de connexion

```
GET / HTTP/1.1
User-Agent: Wget/1.14 (linux-gnu)
Accept: */*
Host: www.google.com
Connection: Keep-Alive
```

Exemple de connexion

```
GET / HTTP/1.1
User-Agent: Wget/1.14 (linux-gnu)
Accept: */*
Host: www.google.com
Connection: Keep-Alive
```

```
HTTP/1.1 200 OK
Date: Wed, 16 Jan 2013 17:03:09 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=[...]; expires=Fri, 16-Jan-2015 17:03:09 GMT;
    path=/; domain=.google.fr
Set-Cookie: NID=[...]; expires=Thu, 18-Jul-2013 17:03:09 GMT;
    path=/; domain=.google.fr; HttpOnly
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Transfer-Encoding: chunked
```

Description d'une requête

```
method path HTTP/version
header: value
...
header: value
(ligne vide)
(données optionnelles)
```

```
GET / HTTP/1.1
Host : www.google.com
...
User-Agent : Wget/1.14 (linux-gnu)
```

Description d'une réponse

`HTTP/version status info`	HTTP/1.1 200 OK
`header: value`	Date : Wed, 16 Jan 2013 17 :03 :09 GM
`...`	...
`header: value`	Set-Cookie : [...]
`(ligne vide)`	
`(données optionnelles)`	<html>[...]</html>

HTTP 2.0 : motivation et historique

Limitations de HTTP 1.1

- le *pipelining* ne marche pas
- de manière générale, problèmes de latence (établissement TCP, répétition des en-têtes)
- format texte complexe à *parser*
- le serveur ne peut *pousser* de données simplement
- *toute évolution du standard doit rester compatible avec HTTP 1.1 et permettre une cohabitation*

Historique

- Google met au point le protocole SPDY en 2009
- SPDY v3 est proposé à l'IETF en 2012
- entre 2012 et 2015, SPDY sert de base à HTTP 2
- RFC 7540 en mai 2015

HTTP 2.0 : fonctionnalités et difficultés

- Mise en forme
  - Format binaire (*plus dur à tester/analyser, mais moins complexe à implémenter*)
  - Compression des en-têtes avec HPACK (*pour contrer CRIME*)
- Fonctionnalités HTTP
  - HTTP2 est surtout une couche de transport pour HTTP/1.1
  - Multiplexage des connexions (avec priorités)
  - Ajout de la possibilité pour le serveur de pousser des données
  - *Une grande complexité dans la machine à état*
- Sélection de la version 2 grâce à l'extension ALPN de TLS (il n'y pas actuellement d'implémentation HTTP 2 sans TLS)

Contenu d'une « page » web

Classiquement, une page web se compose

- d'un contenu hypertexte (documents HTML)
- de feuilles de style (fichiers CSS)
- de scripts (fichiers JavaScript)

Contenu d'une « page » web

Classiquement, une page web se compose

- d'un contenu hypertexte (documents HTML)
  - HTML 4 : tentative de figer un standard après une guerre des navigateurs
  - HTML 5 : *a rolling standard*...
  - en pratique, le HTML peut embarquer du style et des scripts
- de feuilles de style (fichiers CSS)
- de scripts (fichiers JavaScript)

Contenu d'une « page » web

Classiquement, une page web se compose

- d'un contenu hypertexte (documents HTML)
  - HTML 4 : tentative de figer un standard après une guerre des navigateurs
  - HTML 5 : *a rolling standard*...
  - en pratique, le HTML peut embarquer du style et des scripts
- de feuilles de style (fichiers CSS)
  - un langage riche pour disposer les éléments
  - possibilité de tromper l'utilisateur sur la marchandise
  - possibilité d'exfiltrer des données sous conditions
- de scripts (fichiers JavaScript)

Contenu d'une « page » web

Classiquement, une page web se compose

- d'un contenu hypertexte (documents HTML)
  - HTML 4 : tentative de figer un standard après une guerre des navigateurs
  - HTML 5 : *a rolling standard*...
  - en pratique, le HTML peut embarquer du style et des scripts
- de feuilles de style (fichiers CSS)
  - un langage riche pour disposer les éléments
  - possibilité de tromper l'utilisateur sur la marchandise
  - possibilité d'exfiltrer des données sous conditions
- de scripts (fichiers JavaScript)
  - à l'origine du web 2.0 : `XMLHttpRequest` (AJAX)
  - langage devenu très riche (et incontrôlable)
  - code exécuté par le client

Web 2.0

Plusieurs façon de décrire le web d'aujourd'hui

Web 2.0

Plusieurs façon de décrire le web d'aujourd'hui

- par ses technologies
  - HTML5 + CSS + JS

Web 2.0

Plusieurs façon de décrire le web d'aujourd'hui

- par ses technologies
  - HTML5 + CSS + JS
- par ses usages
  - sites dynamiques
  - participation de l'utilisateur au contenu (commentaires, etc.)
  - sites *patchwork (mash-up)*

Web 2.0

Plusieurs façon de décrire le web d'aujourd'hui

- par ses technologies
  - HTML5 + CSS + JS
- par ses usages
  - sites dynamiques
  - participation de l'utilisateur au contenu (commentaires, etc.)
  - sites *patchwork (mash-up)*
- par son architecture
  - clients variés *(responsive design)*
  - notion de *web services*

Caractéristiques du web

- Modèle client-serveur
- Pas d'état a priori (mais il y a des *cookies*)
- Pas de point d'entrée clairement identifié
- Communication en clair par défaut (mais il y a TLS)
- Grande complexité
- Milieu hostile
- Frontières de confiances floues

Caractéristiques du web

- Pas d'état a priori
- Pas de point d'entrée clairement identifié

- Milieu hostile
- Frontières de confiances floues

Remarque en passant : certaines caractéristiques se retrouvent dans le monde des applications mobiles

Une grande complexité

- De nombreux langages et technologies à comprendre
  - HTML+CSS pour le contenu et sa présentation
  - PHP (ou Java, Ruby, etc.) pour le code côté serveur
  - SQL pour la gestion des données
  - JavaScript pour le code côté client
  - la configuration du serveur web (en particulier `.htaccess`)

Une grande complexité

- De nombreux langages et technologies à comprendre
  - HTML+CSS pour le contenu et sa présentation
  - PHP (ou Java, Ruby, etc.) pour le code côté serveur
  - SQL pour la gestion des données
  - JavaScript pour le code côté client
  - la configuration du serveur web (en particulier `.htaccess`)
- Une architecture complexe
  - le serveur web
  - le serveur de base de données
  - le client
  - le *load balancer*
  - tous les services tiers dont vous dépendez...

Un milieu hostile

Plusieurs éléments nécessitent que le développeur soit méfiant

- les entrées utilisateurs sont partout
  - paramètres de l'URL
  - contenu d'une méthode POST
  - cookies
  - ...

Un milieu hostile

Plusieurs éléments nécessitent que le développeur soit méfiant

- les entrées utilisateurs sont partout
  - paramètres de l'URL
  - contenu d'une méthode POST
  - cookies
  - ...
- en mode normal, beaucoup se passe sur le client
  - scripts JS
  - stockage local
  - modes autonomes
  - comment contrôler de manière fiable ce qu'il s'y passe ?

Un milieu hostile

Plusieurs éléments nécessitent que le développeur soit méfiant

- les entrées utilisateurs sont partout
  - paramètres de l'URL
  - contenu d'une méthode POST
  - cookies
  - ...
- en mode normal, beaucoup se passe sur le client
  - scripts JS
  - stockage local
  - modes autonomes
  - comment contrôler de manière fiable ce qu'il s'y passe ?
- un flot de contrôle incontrôlable
  - pas d'état a priori
  - pas de point d'entrée ni de parcours bien identifié
  - tout doit être contraint par le développeur !

Frontières de confiance floues

Un site web récent contient des centaines de ressources :

- des morceaux venant du site visité (`monsite.com`)

Frontières de confiance floues

Un site web récent contient des centaines de ressources :

- des morceaux venant du site visité (`monsite.com`)
- des morceaux venant de sites dépendant a priori de la même entité administrative (`static.monsite.com` ou `monsite-static.com`)

Frontières de confiance floues

Un site web récent contient des centaines de ressources :

- des morceaux venant du site visité (`monsite.com`)
- des morceaux venant de sites dépendant a priori de la même entité administrative (`static.monsite.com` ou `monsite-static.com`)
- des scripts tiers courants (`jquery`)

Frontières de confiance floues

Un site web récent contient des centaines de ressources :

- des morceaux venant du site visité (`monsite.com`)
- des morceaux venant de sites dépendant a priori de la même entité administrative (`static.monsite.com` ou `monsite-static.com`)
- des scripts tiers courants (`jquery`)
- des publicités (régies telles que Google)

Frontières de confiance floues

Un site web récent contient des centaines de ressources :

- des morceaux venant du site visité (`monsite.com`)
- des morceaux venant de sites dépendant a priori de la même entité administrative (`static.monsite.com` ou `monsite-static.com`)
- des scripts tiers courants (`jquery`)
- des publicités (régies telles que Google)
- des liens vers les réseaux sociaux

Frontières de confiance floues

Un site web récent contient des centaines de ressources :

- des morceaux venant du site visité (`monsite.com`)
- des morceaux venant de sites dépendant a priori de la même entité administrative (`static.monsite.com` ou `monsite-static.com`)
- des scripts tiers courants (`jquery`)
- des publicités (régies telles que Google)
- des liens vers les réseaux sociaux

Pour éviter une trop grande porosité entre les contenus issus de ces différents acteurs, la SOP *(Same Origin Policy)*, qui définit les interactions entre « origines »

Authentification HTTP

- HTTP n'a pas d'état !

Authentification HTTP

- HTTP n'a pas d'état !
- Si une authentification est nécessaire, erreur 401 (`Unauthorized`) en indiquant un *domaine* (*realm*)

Authentification HTTP

- HTTP n'a pas d'état !
- Si une authentification est nécessaire, erreur 401 (`Unauthorized`) en indiquant un *domaine* (*realm*)
- Plusieurs modes d'authentification
  - `Basic`
  - `Digest`

Authentification HTTP

- HTTP n'a pas d'état !
- Si une authentification est nécessaire, erreur 401 (`Unauthorized`) en indiquant un *domaine* (*realm*)
- Plusieurs modes d'authentification
  - `Basic`
  - `Digest`
- Ces modes d'authentification sont peu utilisés en pratique
  - Pas de gestion de la déconnexion
  - Problème de couches entre le serveur HTTP et l'application web

Authentification à une application web

- L'application web demande à l'utilisateur de transmettre son identifiant et son mot de passe via un formulaire
- Une fois les authentifiants validés, un *cookie* est envoyé au client
- Ce *cookie* de session est généralement associé à des informations conservées côté serveur

Authentification à une application web

- L'application web demande à l'utilisateur de transmettre son identifiant et son mot de passe via un formulaire
- Une fois les authentifiants validés, un *cookie* est envoyé au client
- Ce *cookie* de session est généralement associé à des informations conservées côté serveur
- Premier problème : stockage des mots de passe côté serveur
  - en clair
  - chiffrés
  - hachés avec une fonction de hachage rapide

Authentification à une application web

- L'application web demande à l'utilisateur de transmettre son identifiant et son mot de passe via un formulaire
- Une fois les authentifiants validés, un *cookie* est envoyé au client
- Ce *cookie* de session est généralement associé à des informations conservées côté serveur
- Premier problème : stockage des mots de passe côté serveur
  - en clair
  - chiffrés
  - hachés avec une fonction de hachage rapide
  - **hachés avec une fonction de hachage dédiée**

Authentification à une application web

- L'application web demande à l'utilisateur de transmettre son identifiant et son mot de passe via un formulaire
- Une fois les authentifiants validés, un *cookie* est envoyé au client
- Ce *cookie* de session est généralement associé à des informations conservées côté serveur
- Premier problème : stockage des mots de passe côté serveur
  - en clair
  - chiffrés
  - hachés avec une fonction de hachage rapide
  - **hachés avec une fonction de hachage dédiée**
- Second problème : a priori tout passe en clair !

Protection du canal applicatif

- Avec HTTP, tout passe en clair, y compris les mots de passe (authentifcation basique/applicative) et les cookies
- on peut utiliser HTTPS (SSL + HTTP)

Protection du canal applicatif

- Avec HTTP, tout passe en clair, y compris les mots de passe (authentifcation basique/applicative) et les cookies
- on peut utiliser HTTPS (SSL + HTTP)
- Attention à bien vérifier le certificat !
  - digression : qu'est-ce qu'un certificat ?
  - et les autorités de certification ?
  - cas particulier d'une application mobile *(certificate pinning)*

Protection du canal applicatif

- Avec HTTP, tout passe en clair, y compris les mots de passe (authentifcation basique/applicative) et les cookies
- on peut utiliser HTTPS (SSL + HTTP)
- Attention à bien vérifier le certificat !
  - digression : qu'est-ce qu'un certificat ?
  - et les autorités de certification ?
  - cas particulier d'une application mobile *(certificate pinning)*
- Problème de la première connexion en HTTP

Protection du canal applicatif

- Avec HTTP, tout passe en clair, y compris les mots de passe (authentifcation basique/applicative) et les cookies
- on peut utiliser HTTPS (SSL + HTTP)
- Attention à bien vérifier le certificat !
  - digression : qu'est-ce qu'un certificat ?
  - et les autorités de certification ?
  - cas particulier d'une application mobile *(certificate pinning)*
- Problème de la première connexion en HTTP
  - HSTS (pour forcer HTTPS) + listes blanches

Protection du canal applicatif

- Avec HTTP, tout passe en clair, y compris les mots de passe (authentifcation basique/applicative) et les cookies
- on peut utiliser HTTPS (SSL + HTTP)
- Attention à bien vérifier le certificat !
  - digression : qu'est-ce qu'un certificat ?
  - et les autorités de certification ?
  - cas particulier d'une application mobile *(certificate pinning)*
- Problème de la première connexion en HTTP
  - HSTS (pour forcer HTTPS) + listes blanches
- Problème des mélanges de contenus (*mixed content*)
  - scripts depuis un site tiers en HTTP
  - images depuis un site tiers en HTTP

Protection du canal applicatif

- Avec HTTP, tout passe en clair, y compris les mots de passe (authentifcation basique/applicative) et les cookies
- on peut utiliser HTTPS (SSL + HTTP)
- Attention à bien vérifier le certificat !
  - digression : qu'est-ce qu'un certificat ?
  - et les autorités de certification ?
  - cas particulier d'une application mobile *(certificate pinning)*
- Problème de la première connexion en HTTP
  - HSTS (pour forcer HTTPS) + listes blanches
- Problème des mélanges de contenus (*mixed content*)
  - scripts depuis un site tiers en HTTP
  - images depuis un site tiers en HTTP
- Et l'authentification par certificat client ?

Protection du canal applicatif

- Avec HTTP, tout passe en clair, y compris les mots de passe (authentifcation basique/applicative) et les cookies
- on peut utiliser HTTPS (SSL + HTTP)
- Attention à bien vérifier le certificat !
  - digression : qu'est-ce qu'un certificat ?
  - et les autorités de certification ?
  - cas particulier d'une application mobile *(certificate pinning)*
- Problème de la première connexion en HTTP
  - HSTS (pour forcer HTTPS) + listes blanches
- Problème des mélanges de contenus (*mixed content*)
  - scripts depuis un site tiers en HTTP
  - images depuis un site tiers en HTTP
- Et l'authentification par certificat client ?
  - en pratique, mêmes problèmes qu'avec les authentification au niveau HTTP

Retour sur les *cookies*

- Par défaut, ils sont récupérables depuis le code JavaScript

Retour sur les *cookies*

- Par défaut, ils sont récupérables depuis le code JavaScript
  - On peut utiliser l'option `httpOnly`

Retour sur les *cookies*

- Par défaut, ils sont récupérables depuis le code JavaScript
  - On peut utiliser l'option `httpOnly`
- Ils sont envoyés en clair en HTTP, même s'ils ont été créés par une page récupérée en HTTPS

Retour sur les *cookies*

- Par défaut, ils sont récupérables depuis le code JavaScript
  - On peut utiliser l'option `httpOnly`
- Ils sont envoyés en clair en HTTP, même s'ils ont été créés par une page récupérée en HTTPS
  - option `secure` et HSTS

Retour sur les *cookies*

- Par défaut, ils sont récupérables depuis le code JavaScript
  - On peut utiliser l'option `httpOnly`
- Ils sont envoyés en clair en HTTP, même s'ils ont été créés par une page récupérée en HTTPS
  - option `secure` et HSTS
- Ils peuvent être écrits au préalable par un attaquant (sans `secure`) pour forcer un identifiant de session (*session fixation attacks*)

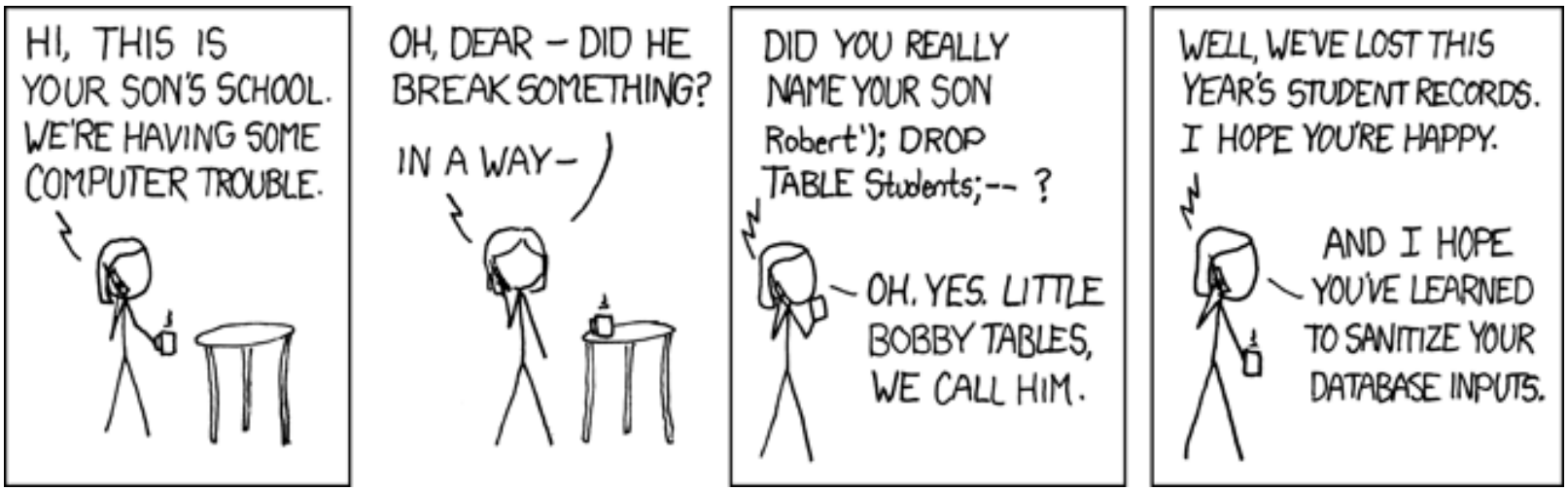
Retour sur les *cookies*

- Par défaut, ils sont récupérables depuis le code JavaScript
  - On peut utiliser l'option `httpOnly`
- Ils sont envoyés en clair en HTTP, même s'ils ont été créés par une page récupérée en HTTPS
  - option `secure` et HSTS
- Ils peuvent être écrits au préalable par un attaquant (sans `secure`) pour forcer un identifiant de session (*session fixation attacks*)
  - Changement régulier d'identifiant nécessaire

Retour sur les *cookies*

- Par défaut, ils sont récupérables depuis le code JavaScript
  - On peut utiliser l'option `httpOnly`
- Ils sont envoyés en clair en HTTP, même s'ils ont été créés par une page récupérée en HTTPS
  - option `secure` et HSTS
- Ils peuvent être écrits au préalable par un attaquant (sans `secure`) pour forcer un identifiant de session (*session fixation attacks*)
  - Changement régulier d'identifiant nécessaire
- Le diable est dans les détails : les cookies de session doivent être imprédictibles
  - il faut aller au-delà des compteurs et des *timestamp*s
  - et utiliser des cookies de taille raisonnable

Failles liées aux injections (1/4) : injections SQL (SQLi)



Source : `http://xkcd.com/327`

Faille liées aux injections (2/4) : XSS

Les attaques de type *cross-site scripting* consistent à injecter des contenus HTML ou JavaScript via des entrées utilisateurs non filtrées

- encore un nouveau type d'injection liée à une interprétation dans un contexte différent

Faille liées aux injections (2/4) : XSS

Les attaques de type *cross-site scripting* consistent à injecter des contenus HTML ou JavaScript via des entrées utilisateurs non filtrées

- encore un nouveau type d'injection liée à une interprétation dans un contexte différent
- *Reflected XSS* : inclusion dans une page d'un paramètre reçu dans l'URL

Faille liées aux injections (2/4) : XSS

Les attaques de type *cross-site scripting* consistent à injecter des contenus HTML ou JavaScript via des entrées utilisateurs non filtrées

- encore un nouveau type d'injection liée à une interprétation dans un contexte différent
- *Reflected XSS* : inclusion dans une page d'un paramètre reçu dans l'URL
- *Stored XSS* : stockage d'une entrée utilisateur dans une base de données, puis affichage non filtré de ces données
  - exemple : commentaires dans un blog
  - exemple : champs dans un certificat X.509 utilisé pour se connecter à un point d'accès Wifi, ensuite affiché dans une console d'administration

Faille liées aux injections (3/4) : LFI

Exemple issu de la page Wikipédia :

```
<?php
    if ( isset( $_GET['language'] ) ) {
        include( $_GET['language'] . '.php' );
    }
?>
```

- C'est une bête injection PHP à partir d'une variable d'URL
- Le développeur fait l'hypothèse (plutôt hardie) que `language` ne peut être définie que par les valeurs du formulaire...

Faille liées aux injections (4/4)

Éléments de réponse

- lorsque c'est possible, ne pas utiliser directement les entrées utilisateur
- utiliser les *prepared statements* ou des outils préservant la structure
- *échapper* les chaînes de caractères avant utilisation
  - l'action à effectuer **dépend du contexte d'utilisation**, qui peut être multiple pour une même chaîne !
- utiliser pour cela un moteur de *templates* fourni par votre *framework*

Faille liées aux injections (4/4)

Éléments de réponse

- lorsque c'est possible, ne pas utiliser directement les entrées utilisateur
- utiliser les *prepared statements* ou des outils préservant la structure
- *échapper* les chaînes de caractères avant utilisation
  - l'action à effectuer **dépend du contexte d'utilisation**, qui peut être multiple pour une même chaîne !
- utiliser pour cela un moteur de *templates* fourni par votre *framework*

Défense en profondeur

- interdire ou retreindre les scripts *(Content Security Policies)*
- protéger les cookies (une cible de choix des XSS)

Problèmes de point d'entrée (1/2)

Exemple de site en carton

- `index.php` force l'utilisateur à s'authentifier et donne ensuite accès à `admin.php`, après avoir renseigné un cookie de session
- `admin.php` ne vérifie pas le cookie de session

- Aucune hypothèse ne peut être faite sur l'ordre dans lequel des pages vont être visitées
- Les contraintes doivent toutes être vérifiées explicitement !

Problèmes de point d'entrée (2/2) : CSRF

Considérons

- un site avec des pages nécessitant une authentification
- un utilisateur connecté
- une page P nécessitant une authentification permettant de faire une action à l'aide d'un formulaire

Problèmes de point d'entrée (2/2) : CSRF

Considérons

- un site avec des pages nécessitant une authentification
- un utilisateur connecté
- une page P nécessitant une authentification permettant de faire une action à l'aide d'un formulaire

Un attaquant met en place avec un formulaire

- préparant les champs permettant de réaliser l'action
- indique que la page pour poster le formulaire est P
- amène un utilisateur *authentifié* sur le site visé à poster le formulaire (possible via un script embarqué invisible).

Problèmes de point d'entrée (2/2) : CSRF

Considérons

- un site avec des pages nécessitant une authentification
- un utilisateur connecté
- une page P nécessitant une authentification permettant de faire une action à l'aide d'un formulaire

Un attaquant met en place avec un formulaire

- préparant les champs permettant de réaliser l'action
- indique que la page pour poster le formulaire est P
- amène un utilisateur *authentifié* sur le site visé à poster le formulaire (possible via un script embarqué invisible).

- Cela marche, malgré la SOP !
- Pour s'en protéger, on ajoute des jetons anti-CSRF dans les formulaires (pour s'assurer que le client est bien passé par le formulaire avant de le poster)
- ces jetons doivent être imprédictibles

Top 10 OWASP (2013)

- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control
- Cross-Site Request Forgery
- Using Components with Known Vulnerabilities
- Unvalidated Redirects and Forwards

Pour aller plus loin

Attention, **ce cours n'est qu'une introduction à un vaste sujet**

- configuration du serveur (par exemple : visibilité des fichiers, *directory traversal*)
- *The Tangled Web* : problème de typage des ressources
  - HTTPv0.9 : le document est rendu directement
  - Qui décide du type d'un contenu ?
- *click-jacking* et prise en compte du DOM
- notion d'origine parfois floue, souvent incohérente
- quid des *plugins* (Java, Flash...)
- HTTPS
- SPDY / HTTP 2.0
- WebSockets (RFC 6455)
- ...

Conclusion

Le web, c'est compliqué

- contexte complexe
- le client web, un milieu franchement hostile
- des contours (origines, points d'entrées) mal définis

Conclusion

Le web, c'est compliqué

- contexte complexe
- le client web, un milieu franchement hostile
- des contours (origines, points d'entrées) mal définis

Quelques conseils

- en général, il est utile de travailler avec un *framework* éprouvé et régulièrement mis à jour
- il ne faut pas négliger l'architecture (utilisation de différents noms de domaine, restriction des privilèges, configuration système des serveurs, etc.)
- faire de la veille sur les composants utilisés
- (faire) auditer votre code
- mettre en place des moyens de détection en cas de souci
- avoir un environnement de développement/production adapté à la mise à jour, y compris en urgence

Questions ?

Merci de votre attention.