Exploitation web

INF600C

Sécurité des logiciels et exploitation de vulnérabilités

Auteur: Jean Privat, Adapté par: Philippe Pépos Petitclerc

Hiver 2023

Université du Québec à Montréal

Exploitation web

Table des matières

Cybersécurité et Web

Pourquoi

- Ubiquitaire: Presque tout et tout le monde sont sur le web
- Exposé: Les informations et services sont faciles d'accès
- Complexe: Multicouches et interactions entre couches
- Imparfait: Produits, outils, développeurs

Objectifs de l'exploitation web

- Dénis de service/vandalisme
- Obtenir de l'information du serveur $\longrightarrow$ elle est normalement privilégiée
- Contrôler le serveur $\longrightarrow$ Pour exécuter du code arbitraire
- Obtenir de l'information des clients $\longrightarrow$ car ils utilisent le serveur compromis
- Pivot, point d'entrée $\longrightarrow$ pour attaquer d'autres machines ou services

OWASP

Promotion des bonnes pratiques en sécurité des application web

- Top 10 Security Risks
- Testing Guide (WSTG)
- Testing Cheat Sheet

Web: Modèle de sécurité

Naviguer sur le web 101

Le client

Un humain qui utilise un navigateur sur son ordinateur

- Affiche des pages web
- Clique sur des liens et boutons

Le serveur

Une suite de logiciels sur une machine

- Accepte les demandes des pages web
- Répond avec la page web à afficher

Protocoles & langages

- Langages du web (client)
  - HTML: structure et contenu
  - JavaScript: code et comportement
  - CSS, médias: rendu
- Protocole du web
  - HTTP: hyper-texte
  - TLS (HTTPS): communication sécurisée
- Couche réseau et système
  - TCP/IP/etc.
  - DNS

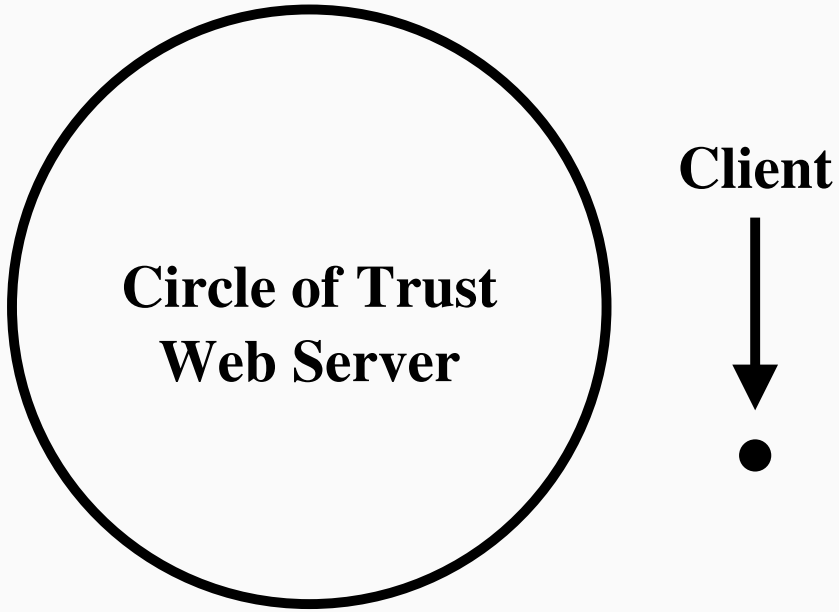
Démo: Confiance

http://confiance.kaa/

Objectif: devenir admin

Web: principe de sécurité

- Le client peut demander n'importe quoi
- Le serveur ne doit jamais faire confiance



OWASP A7:2013 – Contrôle d'accès manquant

Fusionné dans A5:2017 – Contrôle d'accès brisé

Un client peut accéder à des ressources qui ne lui sont pas destinées

Contres-mesures

- Authentifier les clients
- Contrôler l'accès aux ressources sensibles

Fuite d'information

De l'information, possiblement critique, est accessible alors qu'elle ne devrait pas.

- Interfaces d'administration
- Identifiants, clés et mots de passe en clair
- Fichiers de configuration
- *Backups*

Contre-mesures

- Ne pas mettre sur le site des choses privées: `.git`, `.bak`, `.rsa`…
- Désactiver les fuites d'information: messages d'erreurs, contenu des répertoires, sous-sites en développement, etc.

Outils d'analyse

Objectif: parcourir furieusement un site web pour trouver de l'information *non sécurisée* et des problèmes de permissions.

- Wfuzz https://github.com/xmendez/wfuzz
- ffuf https://github.com/ffuf/ffuf
- Dirsearch https://github.com/maurosoria/dirsearch

Voir aussi

- WSTG-INFO-005 Webpage comments and metadata for information leakage
- WSTG-CONFIG-004 Review Old, Backup and Unreferenced Files for Sensitive Information

Protocole HTTP

Protocole HTTP (en bref)

- RFC 7230 et suivantes
- Port 80 (443 pour https) en TCP
- Sans état
- Échange de messages
- Asymétrique
- Client (UA) – Serveur (requête–réponse)
- Le client demande des ressources (URI/URL)

Historique

- 1996 HTTP/1.0
- 1997 HTTP/1.1 connexion persistante, etc.
- 2015 HTTP/2 format binaire, etc.
- 2018 HTTP/3 sur UDP (QUIC), etc.

Request for Comments (RFC)

Publications de l'*Internet Engineering Task Force* (IETF)

- Fait par des ingénieurs et des scientifiques
- Documents avant tout techniques
- Certains sont des standards (TCP, HTTP)

Principe de robustesse

«Soyez conservateur dans ce que vous faites, soyez libéral dans ce que vous acceptez des autres» — Loi de Postel (TCP)

Beaucoup de standards et d'implémentations respectent ce principe.

Problèmes:

- Comportement et compatibilité mal spécifiés
- Cas limites exploitables

URI/URL

- RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax
- Spec URL de WHATWG

Schéma

- `scheme:[//[creds@]host[:port]]/path[?query][#fragment]`

Exemple

- `https://www.example.com/hello.txt?page=1#top`

Requêtes et réponses

Contenu des messages HTTP

- Ligne d'entête (*start-line*)
- Métadonnées (*headers*)
- Donnée (*body*)

Format des messages HTTP

- Texte ASCII: format email (RFC 5322)
- Entêtes: `cle: valeur<CRLF>` (extensible)
- CRLF entre les champs et à la fin de la requête

Exemple de messages

Requête client

```
1  GET /hello.txt?page=1 HTTP/1.1
2  User-Agent: curl/7.16.3 libcurl/7.16.3
3  Host: www.example.com
4  Accept-Language: fr, en
```

Réponse serveur

```
1  HTTP/1.1 200 OK
2  Date: Wed, 24 Jan 2018 09:47:13 -0500
3  Server: Apache
4  Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
5  Content-Length: 51
6  Content-Type: text/plain
7
8  Bonjour le monde, avec un CRLF!
```

Causer directement le HTTP?

- Directement en TCP

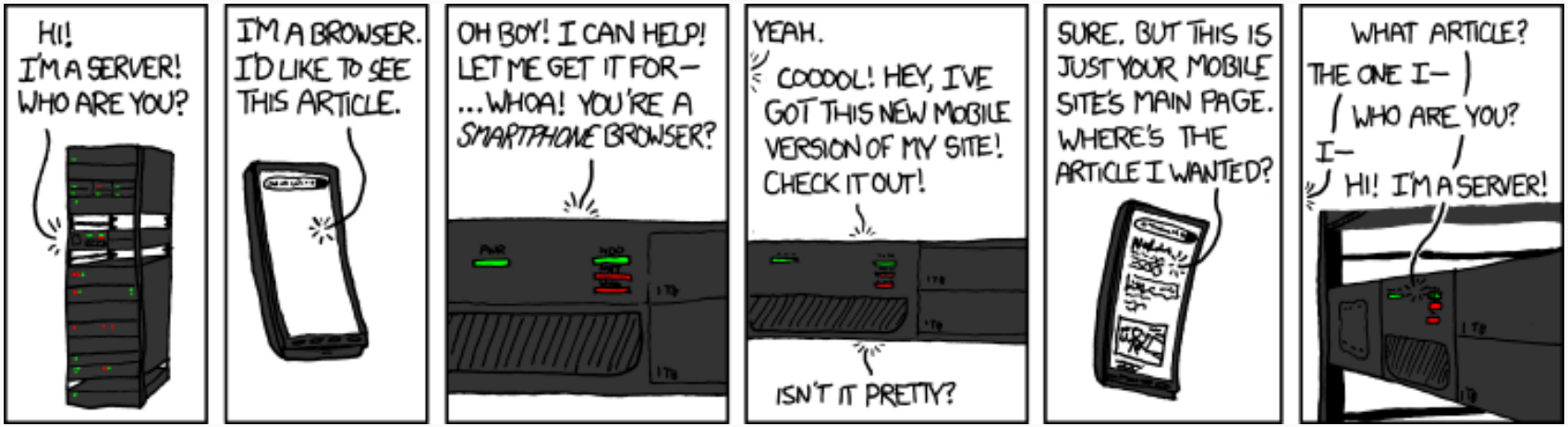
```
$ nc confiance.kaa 80
GET /admin/ HTTP/1.1
Host: confiance.kaa
```

- Outil

```
$ curl http://confiance.kaa/admin/
$ curl http://confiance.kaa/admin/ -v
```

- Navigateur (*dev tools*)

Déficit d'attention



Source http://xkcd.com/869/

Demo: Verbes

http://verbes.kaa/

Verbes HTTP (*Request Methods*)

- GET transfère une représentation de la ressource Pour HTML: les liens normaux `<a>`
- HEAD comme GET mais seulement l'entête
- POST effectue un traitement sur la ressource Pour HTML: la plupart des formulaires `<form>`
- PUT initialise ou remplace une ressource
- et 35 autres verbes enregistrés

Testez les méthodes

OWASP - Test HTTP Methods

Attention aux comportements par défaut de serveurs et frameworks

Démo: Paquet

Confirmation d'expédition

Votre colis n°70053 a été expédié. Vous pouvez le suivre en temps réel grâce à notre service en ligne disponible 6/24h et 4/7j.

http://paquet.kaa/

Pile logicielle

Coté client

- Humain
- Navigateur
- HTML, JavaScript, CSS

$\longrightarrow$ Ne faire confiance à aucun d'entre eux!

Coté serveur

- Du code (serveur web, programmes dédiés)
- Des données (et des bases de données)

Injection de requêtes de base de données

OWASP 2017:A1 - Injection

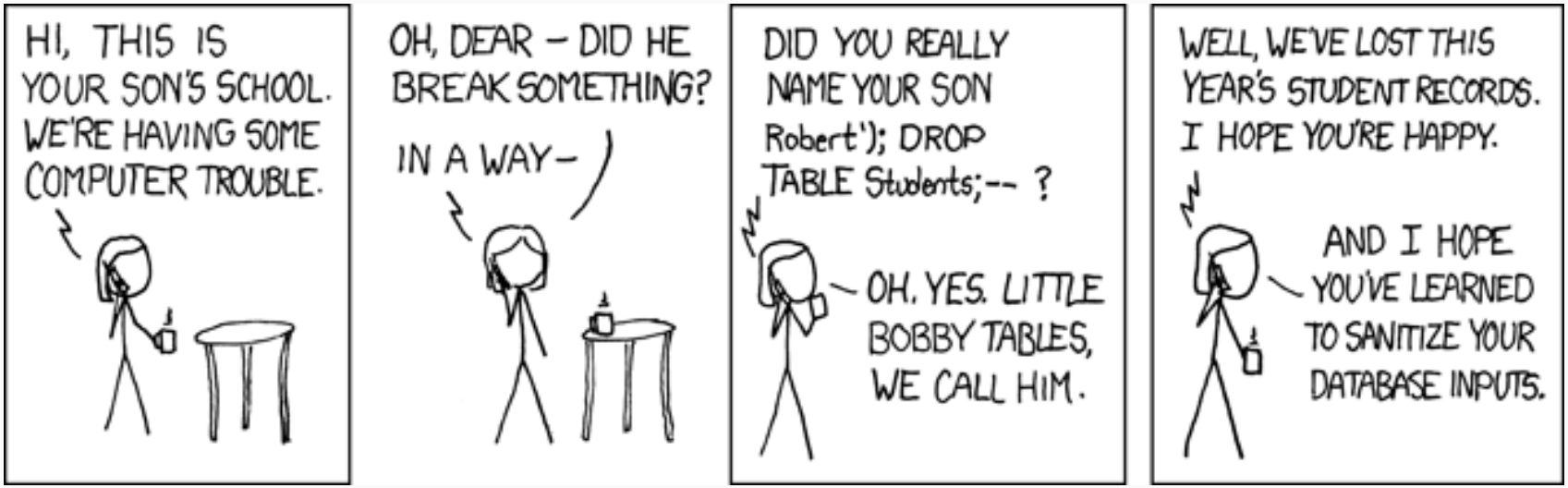
Le client contrôle une partie de la requête SQL

Exemples

```
"SELECT * FROM PAQUETS WHERE ID=" + userid
```

- id = `1 OR 1=1`
- id = `1; DROP TABLE PAQUETS`

Injection SQL



Source: https://xkcd.com/327

Attention

Ne pas faire fuiter de l'information sensible

http://paquet.kaa/database.db

Démo: Biscuit

http://biscuit.kaa/

Cookies et Session

Témoin de connexion (*Cookie*)

- Donné au client par le serveur
- Retransmis par le client à chaque requête

Session

- Séquence d'interaction d'un utilisateur avec un système
- HTTP le fait pas (sans état)

$\longrightarrow$ On doit les programmer (bugs?)

$\longrightarrow$ Les cookies sont une bonne solution

Danger

- Vol de cookie et de session, c.f. plus tard
- Pistage et vie privée (2009/136/EC), c.f. INF4471 (Sécurité) et INM6000 (Informatique et société)

Démo: Commandes de vêtements

http://vetements.kaa/

- Utilisateur jdoe:hunter2

OWASP 2013-A4 – Références d'objet direct non sécurisées

- Fusionné en A5:2017 – Contrôle d'accès brisé
- Promu en A1:2021 – Contrôle d'accès brisé

«Les restrictions sur ce que les utilisateurs authentifiés sont autorisés à faire ne sont souvent pas correctement appliquées. Les attaquants peuvent exploiter ces failles pour accéder à des fonctionnalités et/ou des données non autorisées, telles que l'accès aux comptes d'autres utilisateurs, l'affichage de fichiers sensibles, la modification des données d'autres utilisateurs, la modification des droits d'accès, etc.»

Données des requêtes

Généralement sous forme de paires « cle $\longrightarrow$ valeur »

- dans l'URL (principalement pour GET) urlencodé
- dans le message (principalement pour POST et PUT) urlencodé ou multipart (attribut **enctype**) ou format *ad hoc*

Les frameworks et langages dédiés font (en pratique) le travail de récupération et de décodage des données.

application/x-www-form-urlencoded

- Standard WHATWG URL

« Le format `application/x-www-form-urlencoded` est à bien des égards une aberrante monstruosité, résultat de nombreuses années d'accidents d'implémentation et de compromis conduisant à un ensemble d'exigences nécessaires à l'interopérabilité, mais nullement représentatives des bonnes pratiques de conception [...] »

- `nom=Doe&prenom=John`
- `juron=%25%23%26%2A%21%2B+%C3%A0+gaufres`

Utilisé pour le GET et le POST.

multipart/form-data

- Standard W3C HTML
- RFC 7578 – Returning Values from Forms: multipart/form-data

Avantages sur *urlencode*:

- Précise le codage des éléments
- Permet d'inclure des fichiers (*upload*)
- Permet de préciser les types MIME des fichiers
- Pas besoin de coder/décoder les binaires

L’Attaque des Serveurs

Démo: Croustillant (niveau 1)

http://croustillant.kaa/

Démo: Croustillant (niveau 1)

http://croustillant.kaa/

http://croustillant.kaa/index.php

Démo: Croustillant (niveau 1)

http://croustillant.kaa/

http://croustillant.kaa/index.php

http://croustillant.kaa/index.php.bak

Pile serveur

LAMP

- Linux: Système d'exploitation
- Apache: Serveur web/frontal
- MySQL/MariaDB: Base de données
- PHP/Perl/Python: Langage de programmation

Composants très interchangeables

Pile serveur

Serveur frontal, style Apache

Écoute les requêtes HTTP des clients

- Ressource inconnue/interdite $\longrightarrow$ 404 (ou autre)
- Ressource statique (fichier) $\longrightarrow$ 200 (ou autre)
- Ressource dynamique $\longrightarrow$ on délègue à un langage serveur

Langage serveur, style PHP

- S'exécute avec les droits du serveur web (d'habitude)
- Programmes classiques qui font des calculs
- Programmes classiques qui font des entrées/sorties
- Code *ad hoc*, propice aux bugs

croustillant/index.php.bak

```php
<?php
require('util.php');

$user = null;
if(isset($_GET["name"])) {
  $name = $_GET["name"];
  $pass = $_GET["pass"];
  $user = db_get_user($name);
  if ($user == null) {
    echo "<h1>Mauvais utilisateur</h1>";
    exit;
  }
  if (strcmp($pass, $user->pass)) {
    echo "<h1>Mauvais mot de passe</h1>";
    exit;
  }
```

Objectif: devenir bob

PHP

- Technologie commune du web, coté serveur
- 1994: *Personal Home Page*
- 1997: *PHP: Hypertext Preprocessor*
- 2018: 79% des sites web (source w3techs.com)
- Pas de spécification formelle (wip depuis 2014)
- PHP: a fractal of bad design
- PHP Sadness

OWASP-2017-A2 Broken Authentication

CWE-287 Improper Authentication

Encore lui?

Quelques ressources

- ASVS Authentication
- OWASP Testing for authentication

Credentials (*creds*)

- Identifiants et mots de passe

Du latin *credentia* (confiance, croyance) qui a donné «avoir du crédit», «crédule» ou «mécréant».

- Aie confiance que ce soit moi car je connais un secret que moi seul connaît

Pas de bonne traduction en français

- « Identifiants ». Ambigu.
- « Accréditation ». C'est le processus ou le résultat, mais pas les entrées.
- « Justificatifs d'identité ». Un peu long.
- « Lettre de créance ». Un peu long et vieillot.

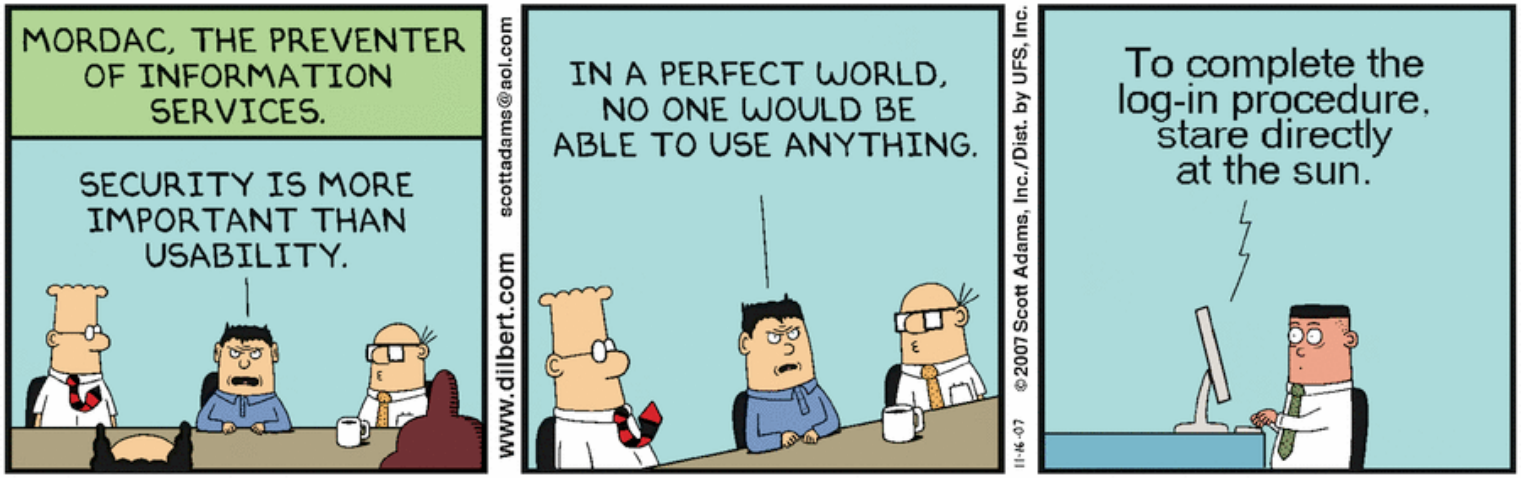
Propositions: «crédentiels», «créditiels» voire «accréditiels» :)

Politique de mots de passe

« Le mécanisme d'authentification le plus répandu et le plus facile à gérer est un mot de passe statique. Le mot de passe représente les clés du royaume, mais il est souvent détourné par les utilisateurs au nom de la facilité d'utilisation. Dans chacune des récentes attaques qui ont révélé les informations d'identification des utilisateurs, les mots de passe les plus courants sont toujours: 123456, password et qwerty. »
Source: WSTG-AUTHN-007 Testing for Weak password policy

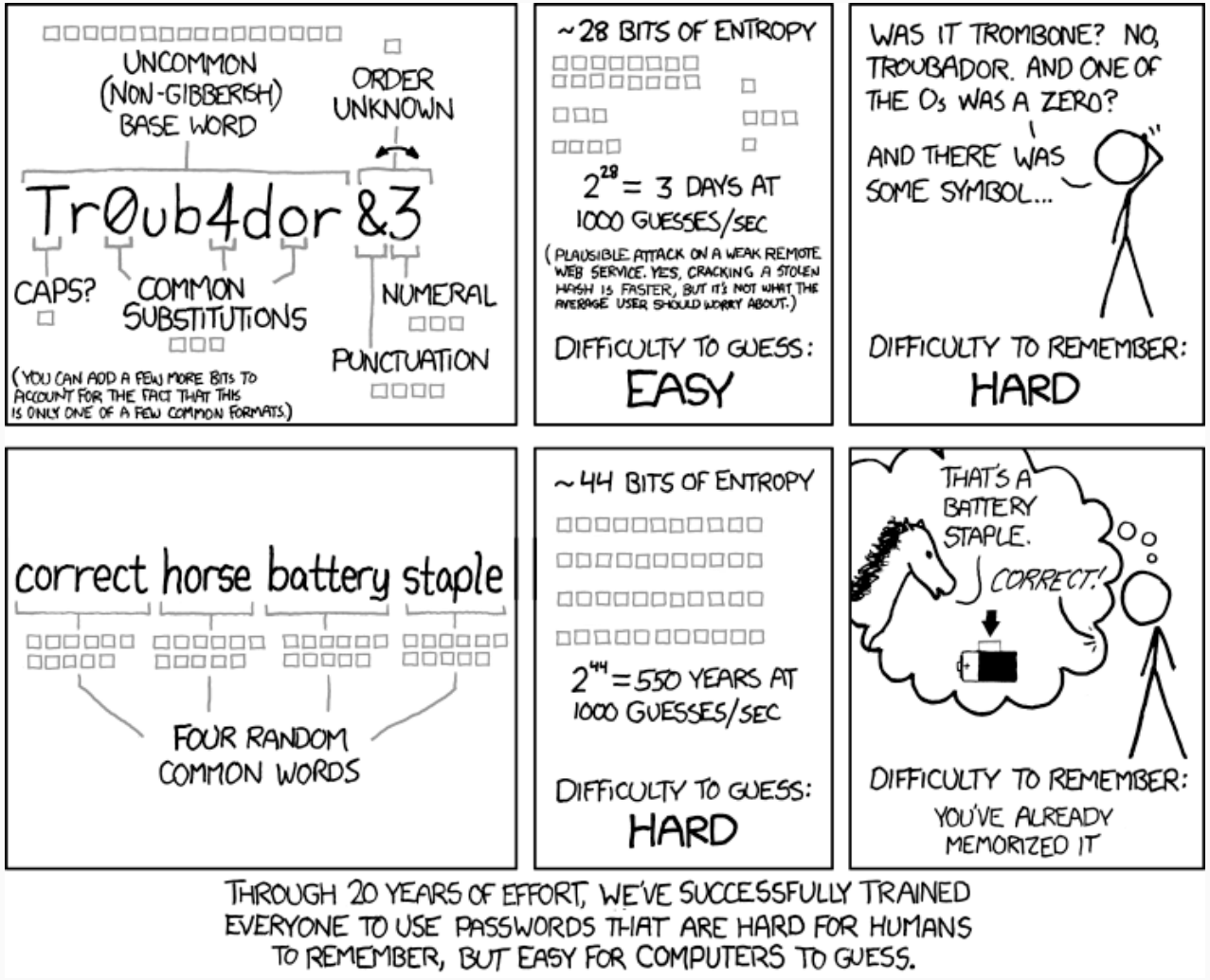
Dilbert sur les mots de passe

Source: http://dilbert.com/strip/1998-04-06



Source: http://dilbert.com/strip/2007-11-16

XKCD: Password Strength



Source: https://xkcd.com/936/

OWASP - A3:2017 - Sensitive Data Exposure

«De nombreuses applications Web et API ne protègent pas correctement les données sensibles, telles que les données financières, soins de santé, et mots de passe. Les attaquants peuvent voler ou modifier de telles données faiblement protégées: fraude à la carte, vol d'identité ou autres crimes. Les données sensibles doivent être protégées autant au repos (stockage) qu'en transit (transmission)»

Mots de passe – Bonnes pratiques

Ne jamais stocker en clair

- Toujours hacher, toujours saler, sel distinct par utilisateur
- Utiliser des fonctions cryptographiques reconnues

Ne jamais transmettre en clair

- Utiliser du HTTPS
- N'envoyer le mot de passe qu'une seule fois $\longrightarrow$ utiliser un identifiant de session par la suite

Vol d'«accréditiels»

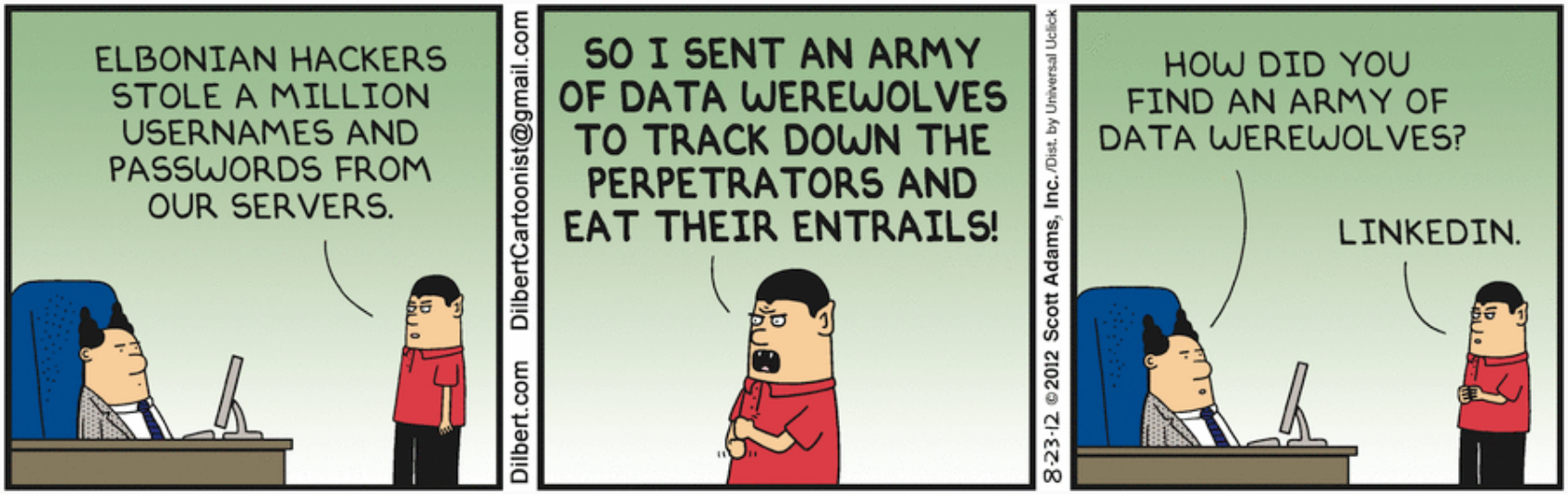
- vol d'identifiants et d'adresses courriel
- vol de mots de passe hachés ou en clair

Top ~~5~~ 8 corpo

- MySpace, en 2008, 360M SHA1 des mots de passe (non salés)
- Wattpad, en 2020, 268M bcrypt des mots de passe
- NetEase (chinois), en 2015, 235M mots de passes en clair
- Zynga, en 2019, 173M SHA1 des mots de passe
- AdultFriendFinder, en 2016 SHA1 des mots de passe
- Dubsmash, en 2018, 162M PBKDF2 des mots de passe
- LinkedIn, en 2016, 160M SHA1 des mots de passe (non salés)
- Adobe, en 2013, 153M mots de passe mal chiffrés

https://haveibeenpwned.com/

Vol de mots de passe



Source: http://dilbert.com/strip/2000-06-05

Mots de passe - Bonnes pratique des utilisateurs

- Utiliser des mots de passes forts

→ Longs: phrases de passe

- Ne pas réutiliser les mots de passe

→ Ne pas faire confiance

- Utiliser un gestionnaire de mot de passes

→ fiable, portable, réparti

Démo: Croustillant (niveau 2)

http://croustillant.kaa/

Objectif: devenir admin

croustillant/index.php

```php
4   $user = null;
5   if(isset($_GET["name"])) {
6     $name = $_GET["name"];
7     $pass = $_GET["pass"];
8     $user = db_get_user($name);
```

... si l'authentification est valide ...

```php
23    $cred = $user->cred();
24    $credstr = base64_encode(serialize($cred));
25    setcookie('cred', $credstr);
26  } elseif (isset($_COOKIE['cred'])) {
27    $credstr = $_COOKIE['cred'];
28    $cred = unserialize(base64_decode($credstr, true));
29    $user = db_get_user($cred->name);
30    if ($user == null || $user->token != $cred->token) {
31      $user = null;
32      echo "<h1>Session perdue</h1><p>Reconnectez-vous</p>";
33    }
```

Bricolage de sessions

Les témoins de session coté client contiennent de l'information

- Intéressante
- Voire pas (ou mal) protégée

Contre-mesures

Utiliser des identifiants de session

- Non sémantiques (opaques)
- Longs et aléatoires

Autres ressources

- OWASP: Session Management Cheat Sheet
- WSTG-SESS-*: Testing for Session Management

PHP jongle avec les types et les valeurs

```php
echo "zero" == 0;
echo null == 0;
echo null == "zero";
echo 0xA == 10;
echo "0xA" == 10;
echo 1e9 == 1000000000;
echo 1e9e5 == 1000000000;
echo "1d9" == 1;
echo "1e9e5" == 1000000000;
echo "1e9" == "1000000000";
echo " 2" == "2";
echo "2 " == "2";
$a = "1d9"; $a++; $a++; echo $a;
echo null < 0;
echo null < -1;
```

Contre-mesures

- Utiliser `===` ; se méfier des autres opérateurs

PHP: The Good Parts

Source: Tom Hudson

Démo: Croustillant (niveau 3)

On a trouvé ça dans le code de `croustillant/util.php`.

```php
class UserCred {
  public $name; public $token;
  function __construct($name, $token) {
    $this->name = $name; $this->token = $token;
  }
  function __destruct() {
    error_log("Debug: fin de " . $this->name);
  }
}
class SafeCmd {
  private $cmd;
  function __construct($unsafe) {
    $this->cmd = escapeshellcmd($unsafe);
  }
  function __toString() {
    return shell_exec($this->cmd);
  }
}
```

Objectif: avoir un shell sur le serveur

Déserialisation insécuritaire

- CWE-502 Deserialization of Untrusted Data
- OWASP A8:2017 - Insecure Deserialization

« La désérialisation non sécurisée conduit souvent à l'exécution de code à distance. Même si des erreurs de désérialisation se produisent dans l'exécution du code à distance, elles peuvent être utilisées pour effectuer des attaques, y compris des attaques par rejeu, des attaques par injection et des attaques par escalade de privilèges. »

Contre-mesures

Ne jamais accepter d'objets sérialisés de l'utilisateur

Injection d'objets

Surface d'attaque

L'attaquant

- ne contrôle pas: le code
- contrôle: les objets instanciés
- contrôle: leurs relations

Principe

- Identifier une méthode automatiquement invoquée (source)
- Identifier une méthode profitable (cible)
- Construire une chaîne de gadgets qui relie les deux
- Gadgets: morceaux de code de l'application L'attaquant les lie ensemble pour faire l'exploit

Chats

Démo: http://chat.kaa/

Objectif: obtenir `index.php` et `/etc/passwd`

chat/fact.php

```
1  <h1>Fait du jour</h1>
2  <?php
3  if (isset($_GET['src']))
4    $src = $_GET['src'];
5  else
6    $src = "chatfaits.txt";
7  $lines=file($src);
8  $i=array_rand($lines);
9  $line=htmlentities($lines[$i]);
10 echo "<blockquote>$i. $line</blockquote>\n";
11 ?>
12 <p>Un <a href="?page=fact">autre chatfait</a> ou un
13 <a href="?page=fact&src=catfacts.txt">cat fact</a>?</p>
```

Traversée des répertoires (*path traversal*)

Quoi?

- L'attaquant trompe l'application
- pour lui faire mécaniquement lire un fichier
- au profit de l'attaquant

Serveur frontal vs. langage serveur

- Chacun interprète et protège les fichiers différemment
- Attention aux API des fichiers: Elles sont souvent plus expressive qu'il n'y parait http://chat.kaa/chat?page=fact&src=php://filter/convert.base64-encode/resource=index.php

Traversée des répertoires: contres-mesures

Rappel d'exploitation système

- Pas de donnée utilisateur dans les API des fichiers
- Faire une base de donnée des ressources (identifiants opaques)
- Au pire, faire une liste blanche

Ressources pour le développement web

- https://www.owasp.org/index.php/File_System
- WSTG-CONFIG-009: Test File Permission
- OTG-AUTHZ-001: Testing Directory traversal/file include

Chats (niveau 2)

Démo: http://chat.kaa/

Objectif: avoir un shell sur le serveur

chat/profil.php

```php
if(isset($_FILES["file"])) {
  $file = $_FILES["file"];
  $save = "img/" . basename($file["name"]);
  $error = null;
  $check = getimagesize($file["tmp_name"]);
  if($check == false or $check['mime'] !== "image/gif") {
    $error = "Le fichier n'est pas une image GIF.";
  } elseif ($file["size"] > 100000) {
    $error = "Le fichier est trop gros.";
  } elseif (!move_uploaded_file($file["tmp_name"], $save)) {
    $error = "Erreur lors du téléversement.";
  }
  header('Location: ' . $_SERVER['REQUEST_URI']);
  if($error !== null) {
    echo $_SESSION['error'] = $error;
  } else {
    $_SESSION['avatar'] = $save;
  }
```

Inclusion de fichiers locaux (*local file inclusion*)

Quoi?

- L'attaquant trompe l'application
- pour lui faire mécaniquement exécuter un fichier
- au profit de l'attaquant

Mitigation/Contre-mesure

- La même chose que pour la traversée de répertoires
- Contrôler le téléversement des fichiers
- Rendre inexécutable les répertoires accessibles en écriture par l'utilisateur
- WSTG-INPVAL-012: Testing for Code Injection
- WSTG-INPVAL-012.1: Testing for Local File Inclusion
- WSTG-BUSLOGIC-008: Test Upload of Unexpected File Types
- WSTG-BUSLOGIC-009: Test Upload of Malicious Files

Flacon (niveau 1)

Démo: http://flacon.kaa/

- Objectif 1: gagner au jeu du 20

Cadriciel Web (*Web Framework*)

Suite logicielle conçue pour faciliter le développement d'applications web

- Construction et déploiement
- Gestion de services communs: accès BD, gestion des sessions, routes

Exemples

- ASP.NET, Java EE, Ruby on Rails, Django, Flask, etc.

Cadriciels

Avantage (théoriques)

- Code mature et testé
- Respecte les bonnes pratiques
- Réduction des coûts de développement et de maintenance

Inconvénient: cible facile

- OWASP 2017:A5 Security Misconfiguration
- OWASP 2017:A9 Using Components with Known Vulnerabilities
- Metasploit: base de données d'exploits pour logiciels, services et *frameworks* cible versions vulnérables et mauvaises configurations

C'est pas parce que c'est pas PHP que c'est forcément plus sécuritaire

Flask

- Cadriciel web en Python
- Commencé en comme un poisson d'avril en 2010

Fonctionnalités

- Débogueur embarqué
- Tests unitaires
- Support HTTP (Werkzeug)
- Gestion REST
- Moteur de patrons (Jinja2)
- Cookies de session sécurisés coté clients

flacon/app/jeu.py

```
4  @app.route('/jeu')
5  def jeu():
6    msg = ""
7    win = False
8    if 'goal' not in session:
9      jeu_reset()
10   elif 'guess' in request.args:
11     session['try'] += 1
12     guess = int(request.args.get('guess'))
13     if guess == session['goal']:
14       msg = "Gagné! On rejoue?"
15       win = True
16       jeu_reset()
17     elif session['try'] > 3:
18       msg = "Perdu! On rejoue?"
19       jeu_reset()
20     else:
21       msg = "C'est pas " + str(guess)
```

Attaque par rejeu (*replay attack*)

Principe

- L'attaquant rejoue, intervertie, ou retarde
- une séquence de communication avec un serveur
- même si chaque communication est chiffrée ou opaque

Exemple

- Vol de séquence d'authentification
  - L'attaquant reproduit une communication sans la comprendre

Contre-mesures (ici)

- Ne pas faire confiance à l'utilisateur
- Stocker les données de session coté serveur
  - Ne donner qu'un identifiant de session à l'utilisateur
  - $\longrightarrow$ long, aléatoire, non sémantique, répudiable

Flacon (niveau 2)

Démo: http://flacon.kaa/

- Objectif 2: faire fuiter la clé secrète

flacon/app/main.py

```
4  def simplepage(page):
5    return '{%extends "base.html"%}{%block content%}'\
6           + page + '{%endblock%}'
7  @app.route('/')
8  def hello():
9    page = "<h1>Veritas</h1><p>« %s »</p>\n"\
10           % Markup.escape(get_fortune())
11   return render_template_string(simplepage(page))
12 @app.errorhandler(404)
13 def page_not_found(e):
14   page = "<h1>La page n'existe pas</h1><tt>%s</tt>"\
15          % request.url
16   return render_template_string(simplepage(page)), 404
17 @app.route('/admin')
18 def admin():
19   if 'is_admin' not in session:
20     session['is_admin'] = False
21   return render_template("admin.html")
```

Injection de patrons coté serveur (*Server-side template injection*)

Principe

- La plupart des moteurs de patrons (*template engine*) permettent d'exécuter du code
- L'attaquant injecte des commandes dans un patron qui sera exécuté

Contres-mesures

- Utiliser les API dédiées pour nourrir le patron
- Ne jamais construire dynamiquement un patron

C'est l'objectif d'un patron!

Flacon (niveau 3)

Démo: http://flacon.kaa/

- Objectif 3: devenir admin

flacon/app/main.py (rappel)

```
4  def simplepage(page):
5    return '{%extends "base.html"%}{%block content%}'\
6           + page + '{%endblock%}'
7  @app.route('/')
8  def hello():
9    page = "<h1>Veritas</h1><p>« %s »</p>\n"\
10           % Markup.escape(get_fortune())
11   return render_template_string(simplepage(page))
12 @app.errorhandler(404)
13 def page_not_found(e):
14   page = "<h1>La page n'existe pas</h1><tt>%s</tt>"\
15          % request.url
16   return render_template_string(simplepage(page)), 404
17 @app.route('/admin')
18 def admin():
19   if 'is_admin' not in session:
20     session['is_admin'] = False
21   return render_template("admin.html")
```

Flacon 1, en vrai

Les cookies de flask c'est du base64 avec un code d'authentification

Code d'authentification de messages (*mac*)

- Code accompagnant un message pour en assurer l'intégrité
- En général c'est un haché du message et d'une clé secrète
- Pas besoin d'être réversible

Cryptosystème

- Principe de Kerckhoffs: la sécurité d'un bon cryptosystème ne repose que sur la sécurité de la clé

$\longrightarrow$ il faut protéger la clé

Flacon (niveau 4)

Démo: http://flacon.kaa/

- Objectif 4: avoir un shell

Sortie de prison (*jail escape*)

Pour limiter les risques

Le code exécutable est volontairement limité dans ses fonctionnalités

Mais l'attaquant est rusé

Il profite quand même de l'expressivité du langage (ou des bugs du moteur d'exécution) pour arriver à ses fins.

Contre-mesures

- Ne jamais laisser le client exécuter du code dans l'environnement courant, même filtré
- Au pire utiliser un DSL dédié dans un environnement limité

L’Attaque des Clients

Objectifs

L'attaquant cible l'utilisateur (ou son navigateur)

- Voler de l'information confidentielle Donnée personnelle, identifiants
- Tromper l'utilisateur Redirection vers site frauduleux, hameçonnage (*phishing*)
- Faire faire des action non consenties Clics de pub, action sur réseau sociaux
- Faire télécharger du logiciel malveillant (*malware*)
- Voler du temps CPU (minage)

État des lieux

Acteurs nombreux (aux objectifs divergents)

- Nombreux navigateurs
- Nombreux frameworks et serveurs HTTP
- Applicatif coté serveur varié
- Applicatif coté client varié

Règles complexes (et imparfaites)

- Extensions HTTP, JavaScript, HTML, etc.
- Normes et pratiques plus ou moins respectées

C’est le bazar

- Support variable dépendant des versions
- Rétro compatibilité, bogues et heuristiques
- Course à l’armement (fonctionnalités et sécurité)


進撃の利用者
L'attaque des Clients

Le Mur (niveau Maria)

Démo: http://mur.kaa/

Objectif: tromper un utilisateur et le faire cliquer sur un lien vers

`http://pwn.kaa/`

mur/index.php

```
26  ™™
27  <h2>Face publique du mur</h2>
28  <?php
29  $content = file_get_contents("pub/wall.txt");
30  if(isset($_GET['editpub'])) {
31    echo "<form method=post action=.>\n";
32    echo "<textarea rows=4 cols=80 name=txtpub>\n";
33    echo $content;
34    echo "</textarea><br>\n";
35    echo "<input type=submit value=Sauver>\n</form>\n";
36  } else {
37    echo "<p style='border:2px solid black;padding:5px'>";
38    echo "$content</p>\n";
39    echo "<a href=?editpub=true>Éditer?</a>\n";
40  }
41  ?>
42  <!-- privé -->
43  <h2>Face privée du mur</h2>
```

Cross-Site Scripting (XSS)

L'attaquant injecte du contenu dans une page web

- Permet de faire passer du contenu comme légitime.
- Profite des fonctionnalités modernes du HTML5.

C'est pas forcément du script, ni entre plusieurs sites. cf Mark Slemko

- Grande classe de vulnérabilités
- Pas toujours bien définie

XSS Contre-mesures

Filtrer et valider les données utilisateur affichées

- Utiliser des bibliothèque spécialisées (et moteurs de template)
- Attention au contexte
- Certains contextes sont trop dangereux: `<script>`, `<!-- -->`, `<style>`, etc.

Ressources

- WSTG-INPVAL-0: Testing for Input Validation
- OWASP XSS Prevention Cheat Sheet
- OWASP XSS Filter Evasion Cheat Sheet
- https://html5sec.org/

Analogies de sécurité

Confused deputy problem (adjoint désorienté)

- Le navigateur est l'adjoint de l'utilisateur
- L'attaquant vise à le désorienter
- Origine historique: Norm Hardy, 1988

Le billard

- L'attaquant rebondit sur le serveur
- puis sur le navigateur
- pour atteindre l'utilisateur

Le Mur (niveau Rose)

Démo: http://mur.kaa/

Objectif Rose: récupérer la face privée du mur d'un utilisateur

JavaScript

- Technologie standard du web, coté client
- 1995: Langage glu, proto codé en 10 jours
- JavaScript != Java JavaScript $\approx$ Lisp et Self « Java is to JavaScript as ham is to hamster » — Jeremy Keith (2009)
- 1997: Standard ECMAScript
- 2005: Asynchronous JavaScript And XML (AJAX)

Content Security Policy (CSP)

Contrôle l'utilisation de ressources d'une page web

- Entête de réponse HTTP `Content-Security-Policy`
- Déclare les origines approuvée des ressources: JavaScript, CSS, etc. (liste blanche)
- Sont suivis par la majorité des navigateurs
- Est utilisé par de nombreux cadriciels web
- Standard W3C

Interdiction par défaut

- JavaScript embarqué (`<script>`, `onclick=`, etc.)
- `eval()` JavaScript
- CSS embarqué (`<style>`, `style=`, etc)
- Plein de trucs HTML5

Protéger le cookie

Les cookies contiennent des information sensibles

HTTP only

- Le cookie n'est que pour le HTTP
- Il n'est pas accessible en JavaScript

$\rightarrow$ On ne fait pas confiance au JavaScript coté client

Secure

- N'envoyer le cookie qu'en https
- On ne fait pas confiance au lien TCP (scan de paquets, homme du milieu)

Le Mur (niveau Sina)

Démo: http://mur.kaa/

- Objectif Sina: Forcer M. Krabs à augmenter le salaire de bob via le formulaire de http://salaire.kaa/

salaire/salaire.php

```php
<?php
session_start();
require "utils.php";
if(isset($_POST['name'])) {
  $user = db_get_user($_POST['name'], $_POST['pass']);
  $_SESSION['user'] = $user;
  header("Location: .");
  exit;
} elseif(isset($_SESSION['user'])) {
  $user = $_SESSION['user'];
} else {
  $user = null;
}
if($user === "admin" and isset($_POST['user'])) {
  db_set_salaire($_POST['user'], $_POST['money']);
  header("Location: .");
  exit;
}
```

Cross-Site Request Forgery (CSRF)

- L'attaquant force l'utilisateur
- à effectuer une action sur un site
- où il est déjà authentifié.

Problème

Le site web ciblé ne distingue pas les requête HTTP originale des requêtes forgées

- même navigateur
- même cookie
- même IP
- etc.

CSRF: Contre-mesures (imparfaites!)

- CSRF Prevention Cheat Sheet

Contrôle d'origine

Vérifier le champ d'entête HTTP `Origin` (ou `Referer`)

Jeton CSRF (*CSRF token*)

Associer formulaire et réponse

- Ajouter un champ secret aux formulaires (à chaque requête)
- Unique et aléatoire (*nonce*)

Inclure l'utilisateur

Pour les opérations critiques (sinon c'est pénible)

- Forcer la réauthentification
- Demander un CAPTCHA

Salaire (niveau 2)

Démo: http://salaire.kaa/

- Note: l'utilisateur utilise un ordinateur public (parce que c'est gratuit)
- Objectif: l'attaquant doit prendre son identité

salaire/index.php (rappel)

```php
<?php
session_start();
require "utils.php";
if(isset($_POST['name'])) {
  $user = db_get_user($_POST['name'], $_POST['pass']);
  $_SESSION['user'] = $user;
  header("Location: .");
  exit;
} elseif(isset($_SESSION['user'])) {
  $user = $_SESSION['user'];
} else {
  $user = null;
}
if($user === "admin" and isset($_POST['user'])) {
  db_set_salaire($_POST['user'], $_POST['money']);
  header("Location: .");
  exit;
}
```

Fixation de session (*session fixation*)

L'attaquant prévoit l'identifiant de session utilisé

- WSTG-SESS-003: Testing for Session Fixation
- CWE-384 - Session Fixation

Contre-mesures

- Réinitialiser l'identifiant de session après authentification
- Utiliser des identifiants de session longs et aléatoires