Le Top10 de l'OWASP

OWASP Genève Spring 2009
23 Avril 2009

Sébastien GIORIA (sebastien.gioria@owasp.fr)
French Chapter Leader

The OWASP Foundation
http://www.owasp.org

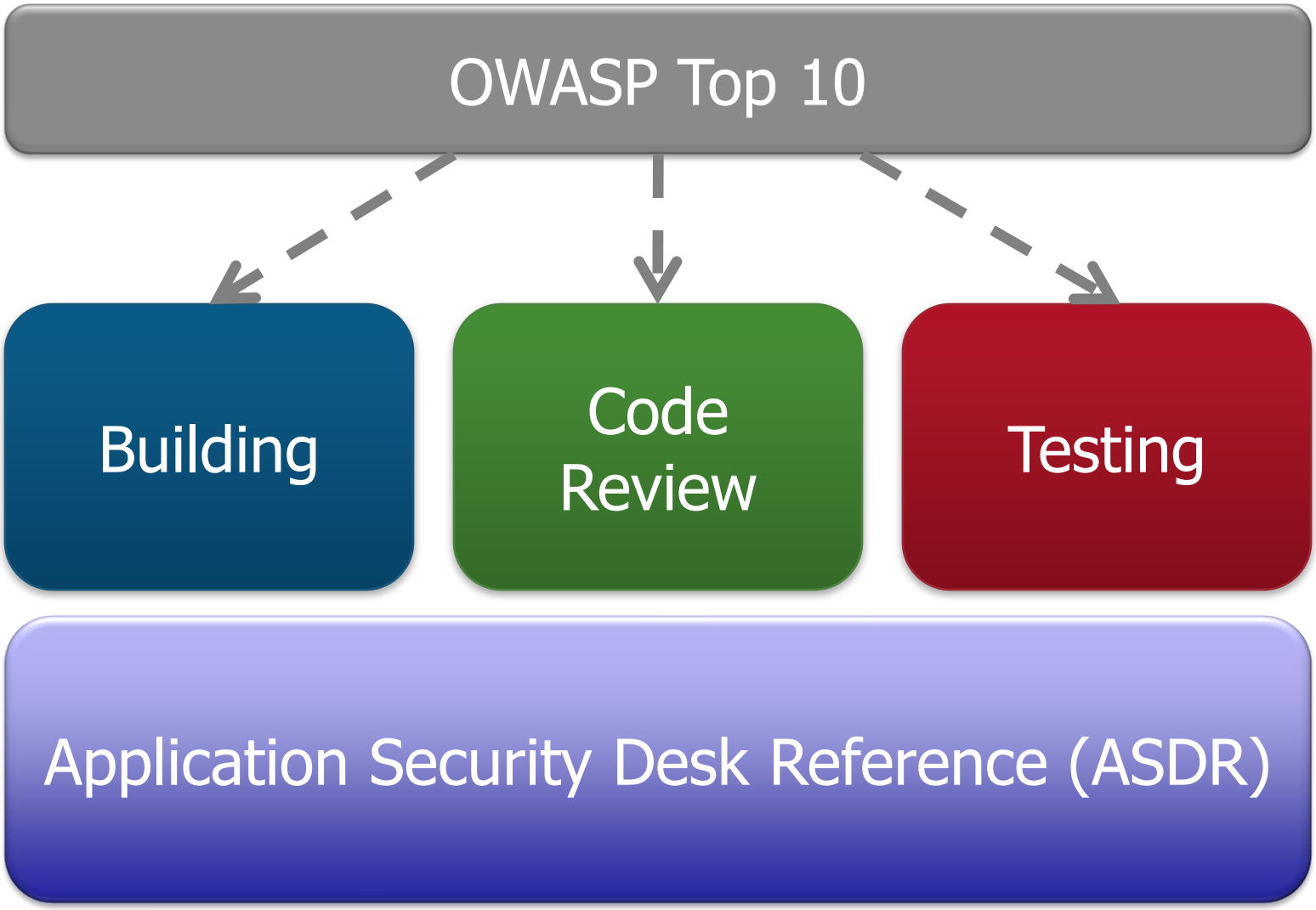
Agenda

- Introduction
- Le Top 10 de l'OWASP
  - Injections côté client (Cross Site Scripting)
  - Injections côté serveur
  - Exécution de fichiers malicieux
  - Références directes non sécurisées aux objets
  - Falsification de requêtes inter-sites (Cross-site Request Forgery)
  - Fuites d'information et traitement des erreurs
  - Attaques sur le traitement des sessions et de l'authentification
  - Utilisation non sécurisée de la cryptographie
  - Communications non sécurisées
  - Manque de restriction d'accès à une URL
- ESAPI & Top 10
- Questions

Sébastien Gioria

- Plus de 10 ans en Sécurité des Systèmes d'Informations.
  - Leader du chapitre Français de l'OWASP 2006
  - Consultant indépendant en sécurité des systèmes d'informations.
  - Président du club Régional de la SSI de Poitou-Charentes (affilié au CLUSIF).
- Expériences professionnelles :
  - Directeur Technique Hébergement et RSSI d'un opérateur télécoms mondial.
  - RSSI Adjoint d'une Banque Nationale Française.
  - RSSI Adjoint au sein d'une Assurance Française.
- Domaines de prédilection :
  - Web 0.9 à Web 2.0, Interfaces riches (Flex, Air, …), WebServices

Les 4 guides, piliers de l'OWASP



Les Guides

- En libre accès sur www.owasp.org
- Issus de l'expérience de milliers d'experts à travers le monde
- OWASP guide
  - Un ouvrage pour la création d'applications Web sécurisées à l'intention des : Développeurs, Architectes, …
  - Inclut les meilleures pratiques dans différents langages (PHP, Java, .Net, …)
  - Plusieurs centaines de pages
- OWASP Testing guide
  - Ouvrage dédié à l'audit sécurité des applications Web à l'intention des pen-testeurs principalement.
- OWASP Code Review Guide
  - Ouvrage destiné à accompagner les auditeurs/testeurs lors des revues de code.

Le Top 10 des vulnérabilités de l'OWASP

- Liste les 10 vulnérabilités les plus rencontrées dans les applications Web
- Mise à jour régulière: *v.2009 en cours*
- Adopté par d'importantes organisations:
  - Federal Trade Commission (US Gov)
  - US Defense Information Systems Agency
  - VISA (Cardholder Information Security Program)
  - Le NIST

A1: Injections client (XSS)

A2: Injections serveur

A3: Execution de fichier malicieux

A4: Référence directe non sécurisée à un objet

A5: Falsification de requête inter-site (CSRF)

A6: Fuite d'information et traitement d'erreur incorrect

A7: Attaques sur la session ou l'authentification

A8: Utilisation non sécurisée de la cryptographie

A9: Communications non sécurisées

A10: Manque de restriction d'accès à une URL

OWASP

The Open Web Application Security Project

http://www.owasp.org

www.owasp.org/index.php?title=Top_10_2007

Les failles d'injection

A1 : Cross Site Scripting

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, potentiellement introduire des vers, etc

A2 : Failles d'injections

L'injection se produit quand des données écrites par l'utilisateur sont envoyées à un interpréteur en tant qu'élément faisant partie d'une commande ou d'une requête. Les données hostiles de l'attaquant dupent l'interpréteur afin de l'amener à exécuter des commandes fortuites ou changer des données

A3 : Execution de fichier malicieux

Un code vulnérable à l'inclusion de fichier à distance permet à des attaquants d'inclure du code et des données hostiles, ayant pour résultat des attaques dévastatrices, telles la compromission totale d'un serveur.

A5 : Falsification de requête inter-site (CSRF)

Une attaque CSRF force le navigateur d'une victime authentifiée à envoyer une demande pré-authentifiée à une application web vulnérable, qui force alors le navigateur de la victime d'exécuter une action hostile à l'avantage de l'attaquant.

La fuite d’information

- **A6 : Fuite d’information et traitement d’erreur incorrect**

  Les applications peuvent involontairement divulguer des informations sur leur configuration, fonctionnements internes, ou violer la vie privée à travers toute une variété de problèmes applicatifs. Les attaquants utilisent cette faiblesse pour subtiliser des données sensibles ou effectuer des attaques plus sérieuses.

- **A9 : Communications non sécurisées**

  Les applications échouent fréquemment à chiffrer le trafic de réseau quand il est nécessaire de protéger des communications sensibles.

- **A10 : Manque de restriction d’accès à une URL.**

  Fréquemment, une application protège seulement la fonctionnalité sensible en empêchant l'affichage des liens ou des URLs aux utilisateurs non autorisés. Les attaquants peuvent utiliser cette faiblesse pour accéder et effectuer des opérations non autorisées en accédant à ces URL directement.

La mauvaise gestion de l’authentification

- **A4 :Référence directe non sécurisée à un objet**

Une référence directe à un objet se produit quand un développeur expose une référence à un objet d'exécution interne, tel qu'un fichier, un dossier, un enregistrement de base de données, ou une clef, comme paramètre d'URL ou de formulaire. Les attaquants peuvent manipuler ces références pour avoir accès à d'autres objets sans autorisation.

- **A7 : Violation de la gestion de l’authentification et des sessions**

Les droits d'accès aux comptes et les jetons de session sont souvent incorrectement protégés. Les attaquants compromettent les mots de passe, les clefs, ou les jetons d'authentification identités pour s'approprier les identités d'autres utilisateurs.

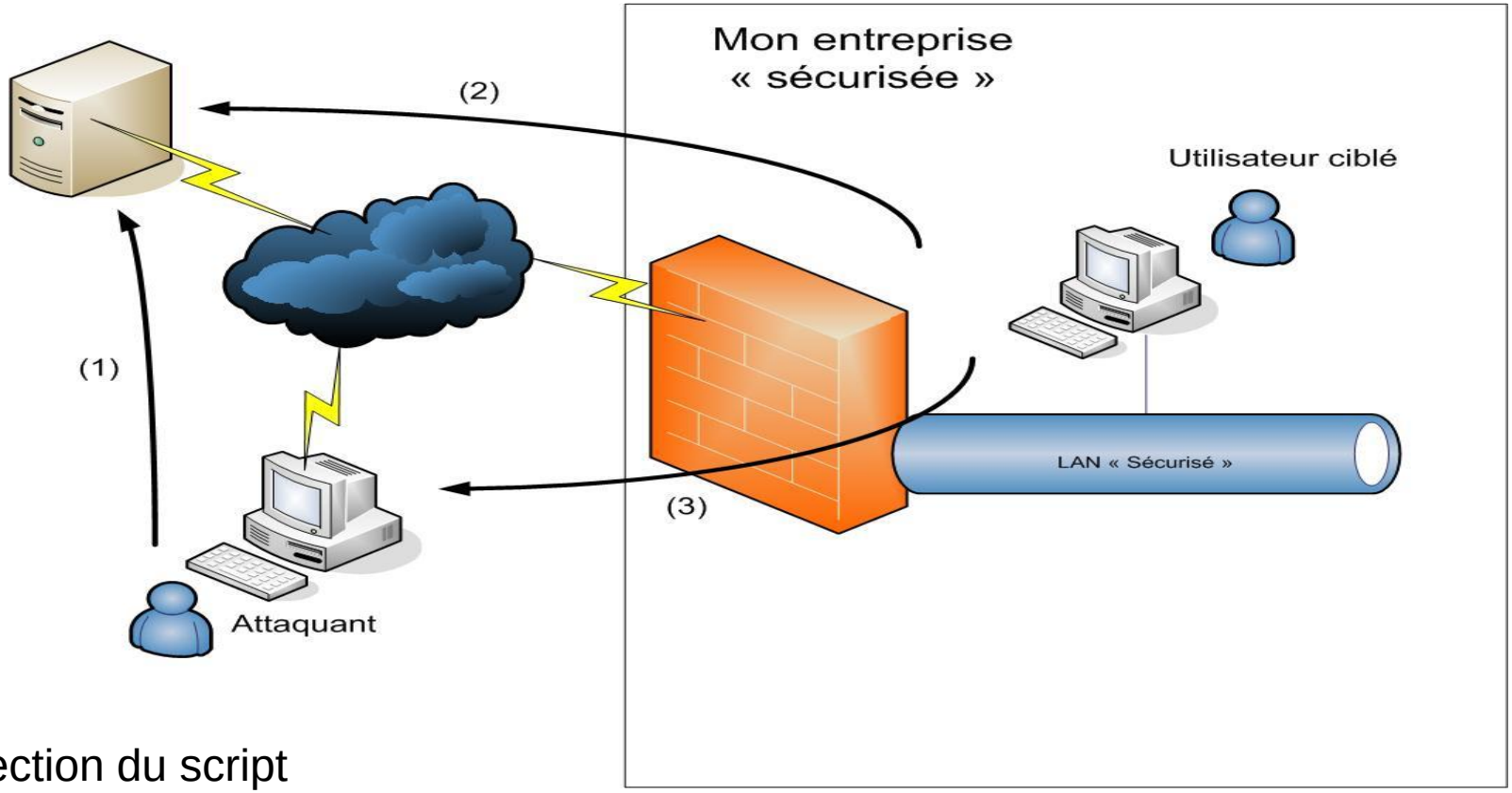
- **A8 :Stockage cryptographique non Sécurisé.**

Les applications web utilisent rarement correctement les fonctions cryptographiques pour protéger les données et les droits d'accès. Les attaquants utilisent des données faiblement protégées pour perpétrer un vol d'identité et d'autres crimes, tels que la fraude à la carte de crédit.

A1 - Principe d'une attaque XSS

- But :
  - Envoyer l'utilisateur vers un site Web malicieux
  - Récupérer des informations contenues dans le navigateur
- Principe :
  - Mail ou lien malicieux
  - Exécution de code dans le navigateur
  - Récupération de données :
    - cookies,
    - objets(IE)
  - Envoi des données vers l'attaquant.
- Dangerosité :
  - Passe outre les contrôles de sécurité (Firewall, IDS, …)
  - Couplée à certaines attaques, permet d'accéder au LAN

A1 - Principe d'une attaque XSS



(1) Injection du script
(2) l'utilisateur se rend sur le serveur vulnérable :
- Suite à un SPAM
- Sur un forum

(3) Récupération des données de façon malicieuse

XSS - Exemple de code vulnérable

- En Java :

```
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class HelloServlet extends HttpServlet
{
public void doGet (HttpServletRequest req, HttpServletResponse res) throws
ServletException, IOException
{

String input = req.getHeader("USERINPUT");

PrintWriter out = res.getWriter();
out.println(input);  // echo User input.
out.close();
}
}
```

- En JSP:

```
<%=request.getParameter("query");%>
```


A1 - Les protections

- Effectuer une validation en profondeur de toute donnée entrante:
  - En-têtes,
  - Cookies,
  - Chaînes de requêtes,
  - Champs de formulaires,
  - Champs cachés,
  - Etc.
- Utiliser des librairies éprouvées:
  - Filtres OWASP (Java/PHP) : http://www.owasp.org/index.php/Category:OWASP_Filters_Project
  - OWASP Stinger project (Filtres JEE) : http://www.owasp.org/index.php/Category:OWASP_Stinger_Project
  - OWASP ESAPI project
  - Microsoft Anti-XSS API (.Net) http://www.microsoft.com/downloads/details.aspx?FamilyID=efb9c819-53ff-4f82-bfaf-e11625130c25&DisplayLang=en

A1 – Protection Java EE

- Validation coté serveur :
  - Utilisation des mécanismes de Struts
  - Mise en place d'expressions régulières
- Validation en sortie :
  - Utiliser les mécanismes de struts *<bean:write … >*
  - Ne surtout pas utiliser les principes ***<%= … %>*** sans être certain de la qualité des données.
- ESAPI :

```
if ( !Validator.getInstance().isValidHTTPRequest(request {
      response.getWriter().write( "<P>Invalid HTTP Request – Invalid
   Characters</P>" );
}
```


A2 - Injections côté serveur

- Ex: injections SQL, LDAP, commandes, etc.
- But :
  - Corrompre les données d'une base, d'un annuaire.
  - Récupérer des informations sensibles dans des bases ou annuaires
  - Exécuter des commandes sur un système distant.
- Principe :
  - Par la modification de la donnée attendue, la requête d'accès à une base SQL est modifiée.
- Dangerosité :
  - Est-il utile de l'expliciter ?

Injection de données - Exemple de code vulnérable

- En PHP :

```
$sql = "SELECT * FROM table WHERE id = '" . $_REQUEST['id’] . "’";
```

- En Java :

```
String query = "SELECT user_id FROM user_data WHERE user_name = '"
+ req.getParameter("userID") + "' and user_password = '" +
req.getParameter("pwd") +"'";
```

- En Filtre LDAP:

Imaginons :
- user=*)(uid=*))(|(uid=*
- pass=password


```
searchlogin=
"(&(uid="+user+")(userPassword={MD5}"+base64(pack("H*",
md5(pass)))+"))";
```



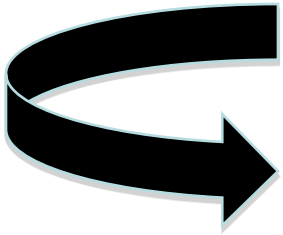
```
searchlogin="(&(uid=*)(uid=*))(|(uid=*)(userPassword={MD5}X03MO
1qnZdYdgyfeuILPmQ==))";
```


Injection de données - En XML:

DTD :

```
<!DOCTYPE users [
        <!ELEMENT users (user+) >
        <!ELEMENT user (username,password,userid,mail+)
>
        <!ELEMENT username (#PCDATA) >
        <!ELEMENT password (#PCDATA) >
        <!ELEMENT userid (#PCDATA) >
        <!ELEMENT mail (#PCDATA) >
]>
```


http://www.example.com/addUser.jsp?username=tony&password=Un6R34kb!e</password><!--&email=--><userid>0</userid><mail>s4tan@hell.com



Résultat :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
        <user>
        <username>tony</username>
        <password>Un6R34kb!e</password><!--</password>
        <userid>500</userid>
        <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
    </user>
</users>
```

Injection SQL – procédures stockées

Considérons :

```
Create procedure get_report @columnamelist varchar(7900) As
Declare @sqlstring varchar(8000)
Set @sqlstring  = ‘Select ‘ + @columnamelist + ‘ from ReportTable‘
exec(@sqlstring)
Go
```

+

Soit :

```
Columnamelist => from users; update users set password = 'password'; select *
```

=

Résultat :

@sqlstring = ‘Select from users; update users set password = 'password'; select * from ReportTable’

A2 - Injection côté serveur - Protection

- Valider les données
- Renforcer les mécanismes du moindre privilège.
- Java EE: utilisation de Spring et Hibernate
- .NET : utilisation de SqlCommand avec SqlParameter ou Hibernate
- PHP : utilisation de PDO
- Utiliser l'OWASP ESAPI

A2 – Injection - Protections JEE

- JAVA EE :
  - PreparedStatements,
  - Spring ou Hibernate
- ESAPI :

```
String input = request.getParameter("param");
if (input != null) {
    if (!Validator.getInstance().isValidString("^[a-zA-Z ]*$", input)) {
                    response.getWriter().write("Invalid: " +
    Encoder.getInstance().encodeForHTML(input) + "<br>");
    } else {
                    response.getWriter().write("Valid: " +
    Encoder.getInstance().encodeForHTML(input) + "<br>");
     }
}
```

A3 - Exécution de fichier malicieux

- But :
  - Installation de code sur le poste distant.
  - Installation de rootkits
- Principe :
  - Par la modification d'une donnée, un fichier de commande est envoyé sur le serveur et exécuté
- Dangerosité :
  - Est-il utile de l'expliciter ?

A3 – Execution de fichier

Soit la portion de code source :

```
String dir = s.getContext().getRealPath("/ebanking")
String file = request.getParameter("file");
File f = new File((dir + "\\" + file).replaceAll("\\\\", "/"));
```

Imaginons :

www.victime.com/ebanking?file=../../etc/passwd

A3 - Protections

- Effectuer une validation des noms de fichiers
- Activer le security manager de Java EE
- ESAPI : classes HttpUtilities

A4 – Référence directe non sécurisée a un objet

- But :
  - Elévation de privilèges
  - Passer outre les contrôles d'authentification
- Principe :
  - Une fois authentifié, la modification d'un paramètre peut parfois donner accès à des informations non accessibles normalement
- Dangerosité :
  - Forte

A4 – Référence directe

•Code Vulnérable en Java

```
int cartID = Integer.parseInt( request.getParameter( "cartID"
    ) );
```

```
String query = "SELECT * FROM table WHERE cartID=" +
    cartID;
```

•Code Vulnérable en PHP

```
require_once ($_REQUEST['language’]."lang.php");
```

A4 - Protection

- Utiliser un index plutôt que d'exposer directement la valeur du champ SQL
- Vérifier que les données ne contiennent pas de caractères tels que:
  - ../
  - %00
- Appliquer le contrôle d'accès au plus près des données:

```
try {
        int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
} catch (NumberFormatException e) {
        out.println ('Erreur');
}
User user = (User)request.getSession().getAttribute( "user" ); String query =
"SELECT * FROM table WHERE cartID=" + cartID + " AND userID=" + user.getID();
```

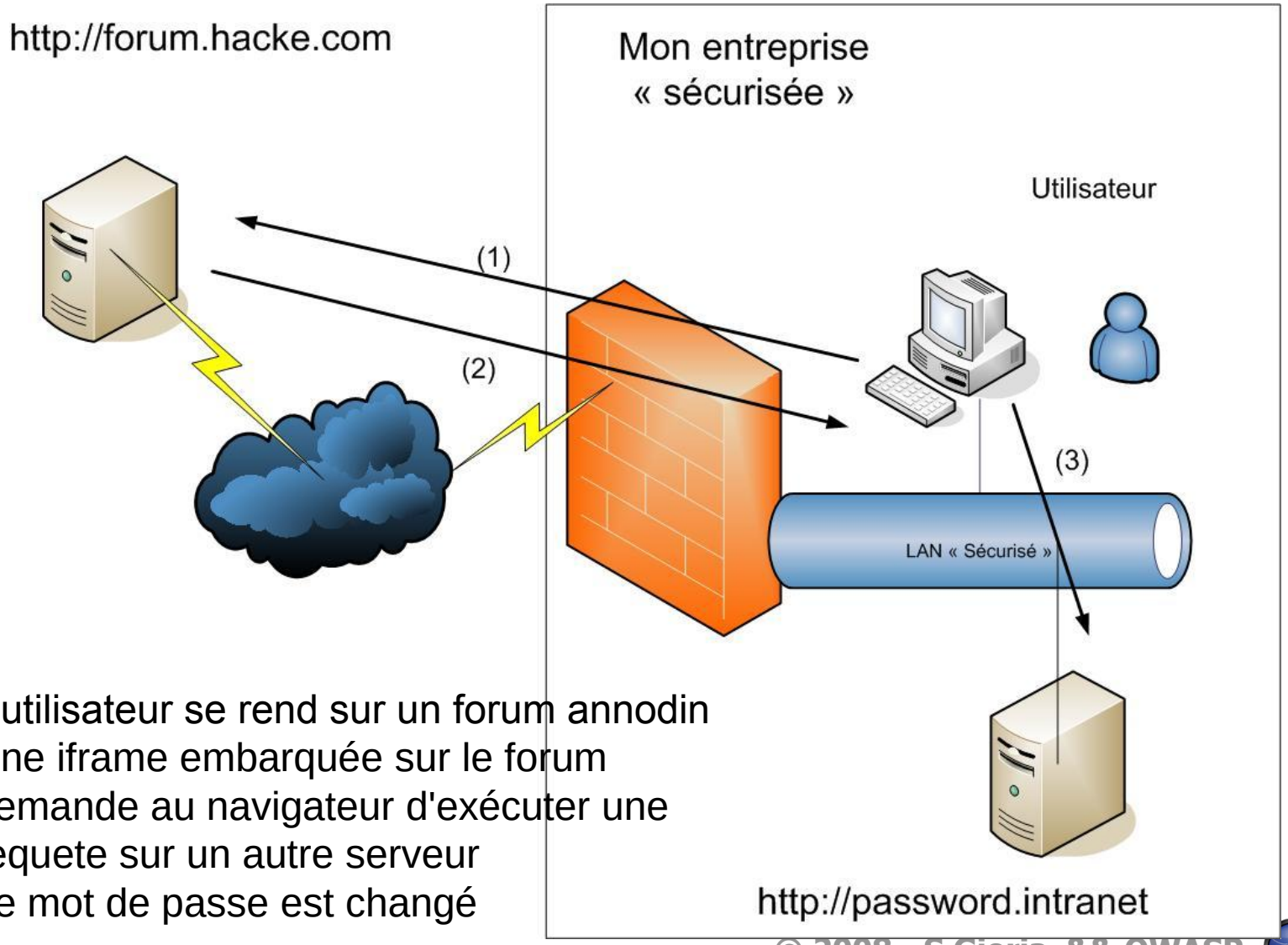

A5 - Cross Site Request Forgery - CSRF

- But :
  - Exécuter une action non désirée par un client sur un site.
  - Récupérer des informations internes
- Principe :
  - Exécution de requêtes sur un site malveillant de façon cachée (via une iframe/image par ex).

```
<img src="http://www.victime.com/transfer.do?frmAcct=document.form.frmAcct
&toAcct=4345754&toSWIFTid=434343&amt=3434.43">
```

- Dangerosité :
  - Passe outre les firewalls/IDS
  - Permet d'accéder au LAN Interne
  - Pas besoin de Javascript

Attaque CSRF



(1) L'utilisateur se rend sur un forum annodin
(2) Une iframe embarquée sur le forum demande au navigateur d'exécuter une requete sur un autre serveur
(3) Le mot de passe est changé

A5 – Les protections

- Vérifier que le code est immune aux vulnérabilités XSS…
- Ne pas utiliser GET pour les traitements sensibles
- Ajouter des jetons aléatoires et uniques qui ne peuvent être envoyés automatiquement par le navigateur
  - Pour ASP.NET, utilisez ViewStateUserKey

```
<form action="/transfer.do" method="post">
 <input type="hidden" name="8438927730" value="43847384383 »>
</form>
```

- Utiliser des librairies éprouvées:
  - OWASP CSRF Guard, http://www.owasp.org/index.php/CSRF_Guard
  - OWASP PHP CSRF Guard, http://www.owasp.org/index.php/PHP_CSRF_Guard

A5 – Protection - ESAPI

```
try {
        HTTPUtilities.getInstance().checkCSRFToken( request );
} catch( IntrusionException e ) {
        response.getWriter().write( "<P>Invalid HTTP Request – Pas de jeton CSRF </P>"
);
}
 String valid =
HTTPUtilities.getInstance().addCSRFToken("/ESAPITest/test?param=test");
response.getWriter().write("  <a href=\""+ valid +"\">valid</a><br>");
```

A6 - Fuite d'informations

- But :
  - Récupérer de l'information sur l'application ou le système sous-jacent à l'application.
- Principe :
  - Emettre des requêtes déclenchant des erreurs ou exceptions
- Dangerosité :
  - Faible, mais néanmoins très utile pour un attaquant

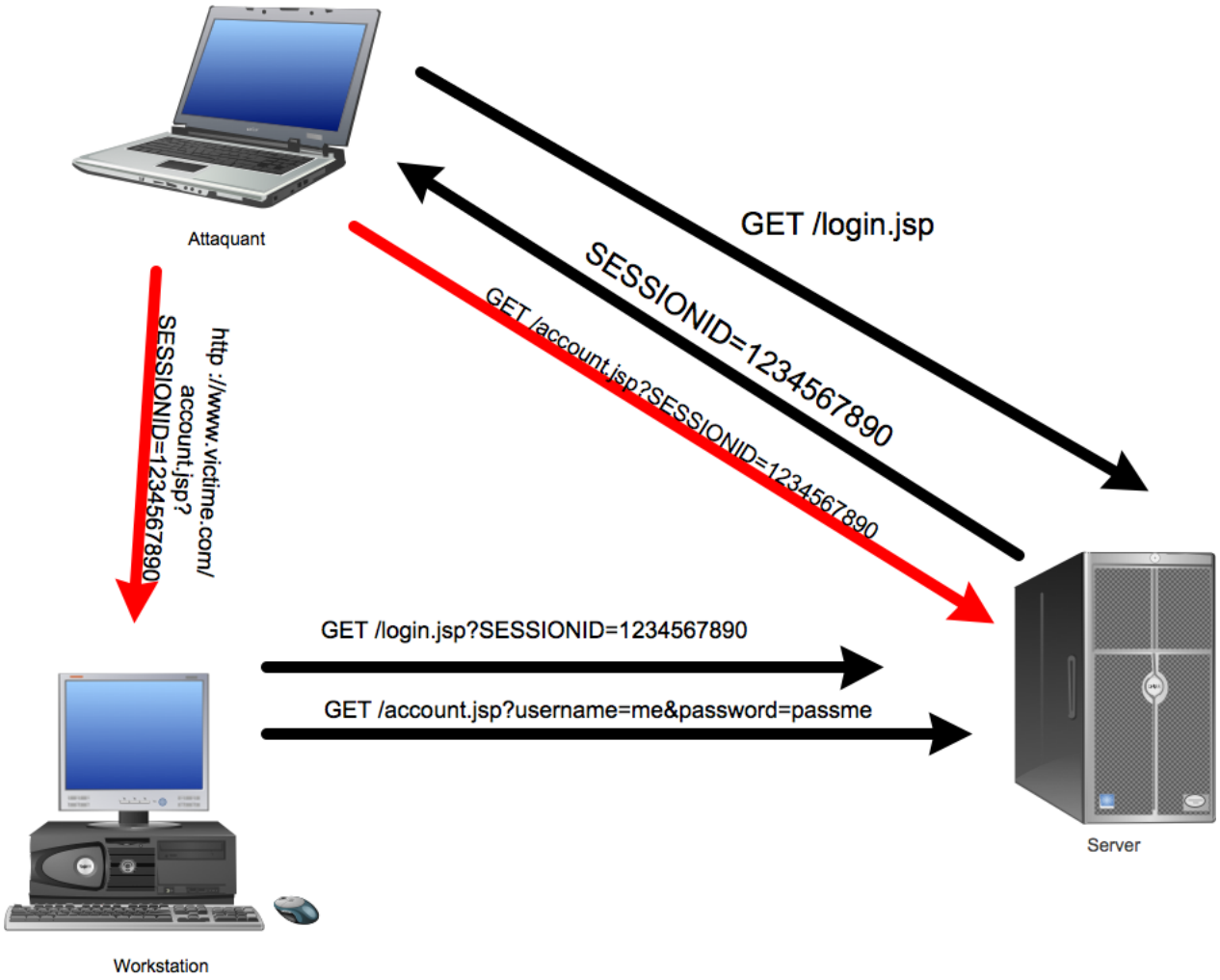
A6 - Les protections

- Désactiver ou limiter la gestion des erreurs/exceptions.
- Modifier le traitement d'erreur pour qu'il retourne une code HTTP 200 Ok.

A7 – Attaques sur la session ou l'authentification

- But :
  - Passer outre les contrôles d'authentification ou de gestion de la session
- Principe :
  - Divers. Ex: Injection d'un cookie bien formé dans les requêtes
- Dangerosité :
  - Forte

A7 – Attaques sur la session ou l'authentification



A7 – Protections

- Préférer les gestionnaires de sessions standards aux librairies ou au Framework
- Ne pas accepter des identifiants de sessions non générés par le serveur
- Éviter d'utiliser ou limiter l'utilisation de cookies personnalisés pour l'authentification
- Imposer un délai maximum d'inactivité de la session

A7 – Protections Java EE

ESAPI :

```
HTTPUtilities esapiHTTPUtilities = ESAPI.httpUtilities();
esapiHTTPUtilities.setCurrentHTTP(request, response);
try {
	esapiHTTPUtilities.changeSessionIdentifier();
} catch(Exception e) {
	e.printStackTrace();
}
```

JAVA EE

```
this.request.getSession().invalidate();
this.request.put("AUTHENTICATED", new Boolean(true));
this.request.getSession(true);
```

A8 – Utilisation non sécurisée de la cryptographie

- But :
  - Obtenir des données sensibles
- Principe :
  - Divers, tels que via une injection ou une XSS
- Dangerosité :
  - Forte

A8 – Utilisation non sécurisée de la cryptographie - Protections

- Utiliser des algorithmes connus et éprouvés:
  - 256 bits minimum pour les transactions bancaires
  - 128 bits minimum pour les transactions autres
  - AES, SHA-256, ….
- Un mot de passe, le numéro d'une carte bancaire, un élément sensible ou autre élément d'authentification ne doit JAMAIS être stocké en clair.
- Accorder autant d'importance à la sécurité des clés qu'aux données elles-mêmes

A9 – Communications non sécurisées

- But :
  - Obtenir des informations
- Technique :
  - Via une écoute réseau, il devient possible d'obtenir des identifiants ou autre si le trafic n'est pas chiffré
- Dangerosité :
  - Faible à Forte

A9 – Communications non sécurisées - Protections

- Utiliser le protocole TLS v1.0/SSL v3.0 pour les communications.
- Utiliser un mécanisme supplémentaire d'authentification sur les transactions sensibles.
- Utiliser des mécanismes d'authentification forte:
  - Pour les accès de niveau administrateur
  - Pour les transactions les plus sensibles

A10 – Manque de restriction d'accès à une URL

- But :
  - Obtenir des accès supplémentaires
- Technique :
  - Par l'accès à des URLs non protégées , il est possible d'executer des taches différentes
- Dangerosité :
  - Moyenne à forte suivant le type d'URL

A10 – Manque de restriction d'accès à une URL - Protections

- Obliger toute transaction à être authentifiée et habilitée.
- Dans le cas des fichiers d'un système, préférer le renvoi d'un flux binaire plutôt qu'un chemin physique vers le fichier.
- Limiter par des listes de contrôle d'accès les accès aux interfaces de gestion et d'administration.

ESAPI

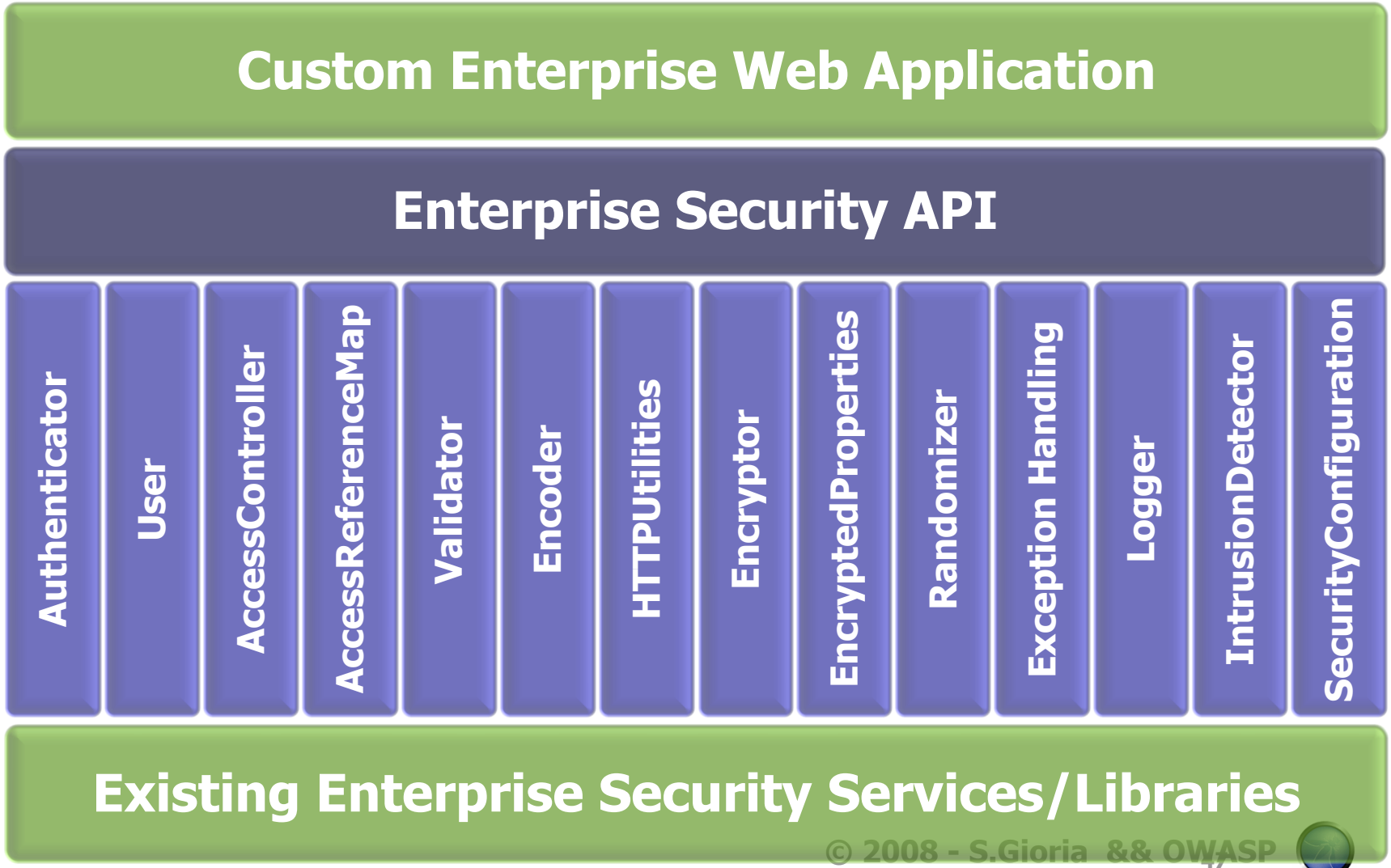
Sébastien GIORIA (sebastien.gioria@owasp.fr)
French Chapter Leader

The OWASP Foundation
http://www.owasp.org

Philosophie de l'ESAPI

- La plupart des développeurs ne devraient pas se poser de questions sur les contrôles de sécurité :
  - Quand les utiliser
  - Comment les utiliser
  - Pourquoi les utiliser
- La plupart des entreprises ont les mêmes besoins

Architecture



Couverture

OWASP Top Ten	OWASP ESAPI
A1. Cross Site Scripting (XSS)	Validator, Encoder
A2. Injection Flaws	Encoder
A3. Malicious File Execution	HTTPUtilities (upload)
A4. Insecure Direct Object Reference	AccessReferenceMap
A5. Cross Site Request Forgery (CSRF)	User (csrftoken)
A6. Leakage and Improper Error Handling	EnterpriseSecurityException, HTTPUtils
A7. Broken Authentication and Sessions	Authenticator, User, HTTPUtils
A8. Insecure Cryptographic Storage	Encryptor
A9. Insecure Communications	HTTPUtilities (secure cookie, channel)
A10. Failure to Restrict URL Access	AccessController

Merci

Questions

"Si vous pensez que l'éducation coûte cher, esssayez donc l'ignorance"

Abraham Lincoln