


Faibles de sécurité des applications Web

Principe, parades et bonnes pratiques de développement

Par Guillaume HARRY - [FirePrawn](#)

Date de publication : 4 juillet 2012


TOUT PUBLIC


Ce document explique les 10 principales faibles de sécurité des applications Web recensées par l' **OWASP** et montre comment s'en prémunir.

I - Introduction.....	4
II - Applications Web.....	4
II-A - Architecture.....	4
II-A-1 - Le réseau Internet et ses protocoles.....	4
II-A-2 - Évolution des architectures applicatives.....	5
II-A-3 - Web 2.0.....	6
II-B - Composants du client Web.....	6
II-B-1 - Le navigateur.....	6
II-B-2 - Les scripts.....	7
II-C - Composants serveur.....	8
II-C-1 - Serveurs Web et serveurs d'application.....	8
II-C-2 - Serveurs de données.....	8
III - Faillies de Sécurité.....	9
III-A - Menaces et risques applicatifs.....	9
III-A-1 - Types d'attaques.....	9
III-A-2 - Risques de sécurité.....	9
III-A-3 - Correspondances entre les définitions de l'OWASP et du WASC.....	10
III-B - Injection.....	11
III-B-1 - Principe.....	11
III-B-2 - Exemples d'attaque.....	12
III-B-3 - Parade et bonnes pratiques.....	13
III-C - Cross-Site Scripting (XSS).....	14
III-C-1 - Principe.....	14
III-C-2 - Exemples d'attaque.....	14
III-C-3 - Parade et bonnes pratiques.....	16
III-D - Violation de gestion d'authentification et de session.....	16
III-D-1 - Principe.....	16
III-D-2 - Exemples d'attaque.....	17
III-D-3 - Parade et bonnes pratiques.....	18
III-E - Référence directe non sécurisée à un objet.....	18
III-E-1 - Principe.....	18
III-E-2 - Exemples d'attaque.....	19
III-E-3 - Parade et bonnes pratiques.....	19
III-F - Falsification de requête intersites (CSRF).....	19
III-F-1 - Principe.....	19
III-F-2 - Exemples d'attaque.....	20
III-F-3 - Parade et bonnes pratiques.....	20
III-G - Mauvaise configuration de sécurité.....	21
III-G-1 - Principe.....	21
III-G-2 - Exemples d'attaque.....	21
III-G-3 - Parade et bonnes pratiques.....	21
III-H - Stockage de données cryptographiques non sécurisé.....	22
III-H-1 - Principe.....	22
III-H-2 - Exemples d'attaque.....	22
III-H-3 - Parade et bonnes pratiques.....	22
III-I - Défaillance dans la restriction des accès à une URL.....	22
III-I-1 - Principe.....	22
III-I-2 - Exemples d'attaque.....	23
III-I-3 - Parade et bonnes pratiques.....	23
III-J - Protection insuffisante de la couche transport.....	24
III-J-1 - Principe.....	24
III-J-2 - Exemples d'attaque.....	24
III-J-3 - Parade et bonnes pratiques.....	24
III-K - Redirection et renvois non validés.....	24
III-K-1 - Principe.....	24
III-K-2 - Exemples d'attaque.....	25
III-K-3 - Parade et bonnes pratiques.....	25
IV - Bonnes Pratiques.....	26


IV-A - Règles de développement.....	26
IV-A-1 - Toutes les données doivent être vérifiées.....	26
IV-A-2 - Privilégier l'utilisation des requêtes POST.....	26
IV-A-3 - Utiliser les requêtes paramétrées.....	26
IV-A-4 - Ne pas réinventer la roue.....	26
IV-A-5 - Sécuriser l'accès aux données.....	27
IV-B - Configuration des composants serveur.....	27
IV-C - Audit.....	27
V - Conclusion.....	28
V-A - Constat.....	28
V-B - Perspectives.....	29
VI - Bibliographie.....	29
VII - Remerciements.....	30


I - Introduction

Bien que les années 2000 aient vu l'explosion de la bulle Internet et l'arrivée du Web 2.0, les standards ont peu évolué. Pourtant les développeurs ont réussi à offrir de l'interactivité avec l'utilisateur et à mettre à disposition de véritables applications sur le Web. Le  **W3C** définit les applications Web comme des applications basées sur le protocole HTTP, indépendantes des plates-formes et langages d'implémentation, reposant sur des architectures Web. Elles peuvent interagir avec d'autres applications de type Web ou autres.

Le Web est devenu un lieu où on peut échanger des informations mais il est également devenu un marché à part entière pour la vente et l'achat de biens matériels. Les acteurs de ce nouveau marché ont besoin de sécurité sous tous ses aspects, tels que définis par  **l'ANSSI** (Agence Nationale de Sécurité des Systèmes d'Information) « la protection de la confidentialité, de l'intégrité et de la disponibilité de l'information ».

Dans ce contexte, de nombreux organismes ont été constitués afin de lutter et de prévenir les risques liés à la sécurité des informations sur le Web.

En France, l'ANSSI est une agence rattachée au Secrétaire général de la défense et de la sécurité nationale. Elle met à disposition des guides sur la gestion des menaces informatiques et des articles sur les recommandations et bonnes pratiques pour la sécurité des systèmes informatiques. Le  **CLUSIF** (Club de la Sécurité de l'Information Français) est une association d'organisations privées et publiques dont le but est de sensibiliser les entreprises et les collectivités françaises à la sécurité de l'information.

Aux États-Unis, le Web Application Security Consortium ( **WASC**) est une association constituée d'experts internationaux, d'industriels et d'organisations du monde Open Source, qui publie des recueils de bonnes pratiques de sécurité pour le Web ainsi que le rapport « WASC Threat Classification » qui décrit et classe les menaces de sécurité sur les applications Web. L'Open Web Application Security Project (OWASP) est une association de bénévoles dont l'objectif est de promouvoir la sécurité logicielle et de sensibiliser les organisations et les personnes sur les risques liés à la sécurité des applications Web. Tous les trois ans, elle publie le classement des dix failles de sécurité les plus dangereuses dans le document « OWASP Top 10 ». Dans sa dernière version de 2010, la liste a été réévaluée afin de prendre en compte les risques et non plus le danger représenté par ces vulnérabilités. En effet, les failles sont maintenant évaluées en fonction de la facilité à trouver la faille, à attaquer l'application Web par ce biais et du préjudice que l'attaque peut causer.

Le présent document « Les principales failles de sécurité des applications Web actuelles : principes, parades et bonnes pratiques de développement » a pour objectif de détailler chacune des failles citées dans le document « OWASP Top 10 ». Afin de mieux les comprendre, l'architecture des applications Web sera abordée dans un premier temps. Dans un second temps chacune des failles sera détaillée en expliquant l'origine du problème et en donnant des exemples-types d'attaque. Les exemples sont écrits pour un environnement Apache/MySQL/PHP. Puis des conseils seront donnés pour se protéger de ce type d'attaque. Enfin un recueil de bonnes pratiques permettra de se prémunir contre la plupart des risques de sécurité dans le développement des applications Web.

II - Applications Web

II-A - Architecture

II-A-1 - Le réseau Internet et ses protocoles

Le Web repose sur le réseau Internet et comme tous les réseaux informatiques, celui-ci repose sur des couches de protocoles de communication. Le paquet est l'unité de base de la transmission de données sur Internet.

Shklar et R. Rosen [1] font la description suivante de la couche de protocoles pour Internet dont le couple TCP/IP est le fondement :

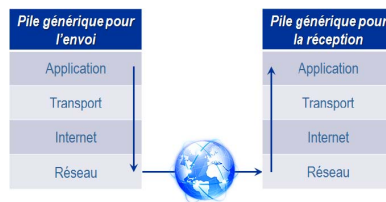


Figure 1 - Transfert des données à travers la pile de protocoles d'Internet

La couche « Réseau » est la couche responsable de la transmission physique des données. Elle peut être implémentée par les lignes téléphoniques, par le GSM ou par du réseau Ethernet par exemple. L'information peut cheminer sur différents supports avant d'atteindre la destination.

La couche « Internet » est la couche qui indique où les données doivent être envoyées, sans garantie que la destination sera bien atteinte. Elle peut utiliser les protocoles IP (Internet Protocol) et ICMP (Internet Control Message Protocol). ICMP permet de vérifier que des messages peuvent être échangés et de gérer les erreurs de transmission. Il est particulièrement utilisé par des outils tels que « ping » et « traceroute ». IP est utilisé pour la plupart des communications sur Internet. Il prend les données issues des couches supérieures, décrites ci-dessous, et les divise en paquets de taille prédéterminée pour faciliter leur transmission sur le réseau [2]. Ainsi, si un paquet est corrompu durant la transmission, il n'est pas nécessaire de réémettre tout le message, mais uniquement le paquet. Chaque paquet est transmis individuellement et peut emprunter un chemin différent des autres. À l'arrivée la couche Internet réassemble l'ensemble des paquets pour reformer le message original.

La couche « Transport » repose sur deux protocoles : TCP et UDP (User Datagram Protocol). TCP s'assure que les paquets sont reçus dans le même ordre qu'ils ont été envoyés et que les paquets perdus sont à nouveau envoyés. TCP est donc un moyen de transmission fiable puisqu'il s'assure que les paquets sont arrivés. Comme indiqué par G.Florin et S.Natkin [3], UDP est un protocole simplifié. Cela permet de transmettre des informations plus rapidement qu'avec TCP puisqu'il y a finalement moins d'informations échangées. UDP est utilisé notamment par NFS (Network File System) et les applications de streaming audio et vidéo telles que la vidéoconférence et la téléphonie sur IP où la perte de paquets est acceptable et la vitesse de communication primordiale.

La couche « Application » est celle qui permet aux utilisateurs finaux de communiquer sur Internet avec des protocoles tels que Telnet, pour agir sur un serveur : FTP (File Transfer Protocol) pour la transmission de fichiers, SMTP (Simple Mail Transfer Protocol) pour l'envoi de courrier électronique et HTTP pour le Web. HTTP est un protocole de messages de type texte, basé sur le paradigme « requête/réponse ». L'utilisateur envoie via son navigateur un message, la requête, au serveur HTTP. Chaque requête est traitée individuellement et de façon unique. Ensuite le serveur renvoie un message, la réponse, au navigateur. HTTP est un protocole déconnecté, c'est-à-dire que le protocole ne permet pas d'établir des communications entre requêtes pour partager des informations, alors qu'une application Web a besoin de conserver les réponses des différentes requêtes d'un utilisateur pour avoir le même comportement qu'avec une application non Web. C'est pourquoi la plupart des navigateurs intègrent le système de « cookie » qui permet de conserver le résultat d'une requête.

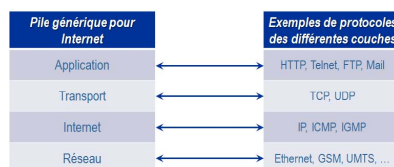


Figure 2 - Pile de protocoles d'Internet

II-A-2 - Évolution des architectures applicatives

Les applications Web ont suivi la même évolution que les applications plus anciennes.

La décennie 1970-1980 était dominée par le système Mainframe. Un serveur centralisait l'ensemble des informations, exécutait les traitements, gérait les droits d'accès. Le client manipulé par l'utilisateur permettait d'envoyer des

demandes de traitement au serveur et d'en afficher les résultats. La machine était passive. Ce mode de fonctionnement est le même que celui du protocole HTTP. Le navigateur ne sert qu'à afficher les réponses HTTP.

La décennie suivante 1980-1990 a vu l'émergence du système client/serveur. Le client récupérait des données depuis le serveur de base de données, exécutait les traitements puis affichait les informations à l'écran et enfin mettait à jour les données sur le serveur si nécessaire. Le serveur ne servait plus qu'à stocker les informations et à exécuter éventuellement des traitements de nuit. Cette architecture posait des problèmes de maintenance des applications sur chacun des postes utilisateur concernés. Les applications Web ont suivi cette évolution avec les « applets » au début des années 90. Il s'agissait d'applications écrites dans un langage de développement, exécutées par le navigateur depuis un site Web.

À partir des années 90, les architectures étaient composées de plusieurs tiers. L'application cliente présentait alors les informations à l'utilisateur et invoquait des services. Les services étaient responsables de l'exécution des processus. Les processus pouvaient être distribués sur plusieurs serveurs. Enfin des serveurs étaient responsables du stockage des données. Dans le milieu des années 90, les applications Web ont également intégré plusieurs composants. Le navigateur Web ne s'occupe plus que de l'affichage. Le serveur HTTP pour répondre aux requêtes génère dynamiquement l'interface graphique dans les pages HTML en faisant appel à des services ou en interrogeant les bases de données.

Depuis les années 2000 les applications Web et les autres types d'applications clientes peuvent utiliser les mêmes services, ce qui facilite la réutilisation des développements et évite la redondance des données.

II-A-3 - Web 2.0

J. Governor, D. Hinchcliffe et D. Nickull [4] expliquent que le Web 2.0 n'est pas une mise à jour technique mais un changement de comportement des internautes. Comme évoqué précédemment, le Web avait pour but initial de mettre à disposition des informations. L'utilisateur était passif face aux sites Web. Puis le Web est devenu collaboratif, l'utilisateur est alors devenu créateur de contenu sans avoir à connaître les protocoles techniques sous-jacents. L'internaute ne consulte plus l'information, il publie du contenu quel que soit le média (texte, vidéo, musique). Internet a permis de mettre en relation des ordinateurs et est devenu le support du Web qui a permis d'y mettre à disposition des informations. À son tour le Web est devenu le support du Web 2.0 qui a permis de mettre en relation des personnes.

Ce nouveau comportement a pu naître grâce à la possibilité de modifier l'interface graphique sans recharger complètement la page Web. Elle devient en réalité un conteneur dans lequel il est possible de mettre à jour et de différencier le contenu, les fonctions, selon la zone de la page.

II-B - Composants du client Web

II-B-1 - Le navigateur

Dans les architectures citées précédemment, le navigateur est une application cliente [2]. Il permet d'envoyer des requêtes HTTP au serveur Web et d'en interpréter la réponse. Les navigateurs sont aujourd'hui capables de travailler également avec le protocole FTP et d'afficher d'autres formats tels que XML. Il existe plusieurs méthodes HTTP pour envoyer des requêtes au serveur. Les plus répandues sont GET, HEAD et POST [1]. GET permet de demander le téléchargement du contenu d'un document, HEAD permet de n'en récupérer que l'en-tête et POST permet d'envoyer des informations au serveur pour traitement. Lorsque l'utilisateur saisit une adresse ou clique sur un lien hypertexte, le navigateur envoie une requête GET au serveur qui ne comprend qu'un en-tête. Les requêtes POST ont un corps de message qui comporte l'ensemble des informations saisies dans un formulaire, alors qu'avec GET, ces informations sont transmises en ajoutant des paramètres à l'adresse.

HTML est un langage qui permet de décrire le contenu et la structure d'une page Web. Il est composé d'environ 50 instructions de mise en forme. La dernière version rédigée par le W3C en 1999 est 4.01 [5]. La version 5 prévue pour 2012 permettra d'intégrer et de standardiser toutes les innovations développées pour compléter version actuelle qui

ne permet pas de représenter des graphiques, de représenter une animation ou d'interagir avec l'utilisateur. Dans un document HTML, la structure de la page peut être représentée par un arbre appelé DOM (Document Object Model).

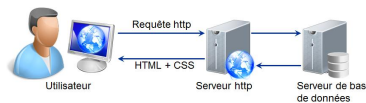


Figure 3 - Mode de fonctionnement des applications Web

XHTML est une extension d'HTML basée sur le langage structuré de description de données XML (eXtensible Markup Language).

CSS (Cascading Style Sheets) est un complément d'HTML dont la version 2 actuelle a été développée en 1998. Une feuille de style (style sheet) est un ensemble de règles à appliquer aux éléments HTML. L'utilisation de CSS permet de séparer les responsabilités dans la présentation des données à l'utilisateur. HTML peut ainsi être responsable du contenu de la page affichée alors que CSS sera responsable de la mise en page. En modifiant seulement le CSS, il est ainsi possible de modifier l'apparence d'une application Web. La future version CSS3 est prévue pour 2012.

RSS (Really Simple Syndication ou Resource description framework Site Summary) est un format simple pour la syndication de contenu. C'est un des outils identifiés comme faisant partie du Web 2.0. Il permet de centraliser des liens vers des sites en affichant pour chacun un titre et une description.

II-B-2 - Les scripts

HTML n'offrant pas de comportement dynamique aux pages Web, les éditeurs ont créé des langages de scripts et des extensions aux navigateurs.

JavaScript est un langage de scripts inventé en 1995 pour le navigateur Netscape. Il permet de manipuler les objets de l'arbre DOM et de gérer la réaction à des événements générés par les objets de la page. Il permet en fait de modifier la page HTML sans envoyer de requête HTTP. Bien qu'il existe une norme ECMAScript gérée par l'organisme de spécifications ECMA, chaque navigateur a développé son propre interpréteur, ce qui pose des problèmes de portabilité des applications Web.

Afin de compléter JavaScript pour permettre de modifier la page HTML affichée en communiquant avec le serveur HTTP, sans recharger toute la page, Microsoft a développé en 2002 un nouvel objet JavaScript : XMLHttpRequest. Aujourd'hui la plupart des navigateurs intègrent cet objet. Cela permet de communiquer de manière asynchrone avec le serveur, ce que ne permet pas actuellement HTML. Ce nouvel outil a été la base du développement du Web 2.0 et sera intégré à la future norme HTML 5.

AJAX (Asynchronous Javascript And XML) est un terme inventé en 2005 par Jesse James Garrett. Il ne s'agit pas d'une nouvelle technologie mais d'une façon d'utiliser conjointement les technologies XHTML, JavaScript et CSS. Les applications Web développées selon le paradigme Ajax utilisent massivement les requêtes GET pour mettre à jour l'interface graphique.

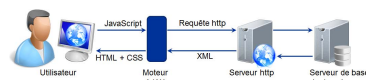


Figure 4 - Mode de fonctionnement des applications Web 2.0

Des éditeurs tels que Macromedia et Microsoft ont également développé des extensions (plugin) aux navigateurs comme Flash et Silverlight pour offrir des interfaces plus riches. Comme le présent document ne traitera pas les failles de sécurité de ces composants, ils ne seront pas plus amplement détaillés.

II-C - Composants serveur

II-C-1 - Serveurs Web et serveurs d'application

Comme évoqué précédemment, le navigateur et le serveur communiquent en utilisant le protocole HTTP. Les serveurs ne sont pas obligés d'implémenter toutes les méthodes HTTP, seulement GET et HEAD. Bien qu'optionnelle, peu de serveurs actuels n'implémentent pas la méthode POST.

Sur Internet le navigateur et le serveur HTTP communiquent rarement directement. Le plus souvent un serveur intermédiaire est présent : le serveur proxy. Les requêtes à destination du serveur sont interceptées par le serveur proxy qui peut leur faire subir un traitement, avant de les retransmettre au serveur. Ce principe est également appliqué aux réponses. Le serveur proxy peut servir de cache pour moins solliciter le serveur HTTP. Il est possible de faire agir plusieurs serveurs proxy en cascade.

La seule fonction du serveur Web étant d'envoyer le contenu des fichiers au client, des extensions peuvent y être ajoutées, permettant de faire appel à des services pour générer dynamiquement les informations à transmettre. Ils traitent les requêtes HTTP que le serveur HTTP leur a fait suivre, interprètent et exécutent le code de l'application, puis génèrent une réponse qu'ils renvoient au serveur HTTP qui l'enverra au navigateur de l'utilisateur. Si ces services fonctionnent indépendamment du serveur HTTP, ils sont appelés serveurs d'applications.

L'extension CGI (Common Gateway Interface) permet l'exécution d'un programme extérieur appelé « gateway » dont la sortie standard d'affichage sera renvoyée au client par le serveur HTTP. Le langage utilisé pour le développement du programme n'a pas d'importance. Cela peut être du C++, du Perl ou même Java. Il faut seulement que le programme puisse être exécuté sur la machine hébergeant le serveur HTTP.

De même que CGI, l'extension Java Servlet développée par Sun permet d'intercepter les requêtes, de générer les réponses pour exécuter des applications Java. La différence est que la machine virtuelle Java peut être lancée sur un serveur différent du serveur HTTP. Cette extension permet de conserver des informations entre les requêtes.

JSP (Java Server Pages) est un langage qui permet d'insérer des blocs de script basés sur Java dans un contenu HTML. Les pages JSP sont interprétées et transformées en Servlet par un serveur d'application pour être exécutées.

ASP (Active Server Pages) est le concurrent de JSP développé par Microsoft. Étant basé sur le langage VBScript, ASP est devenu très populaire dans le milieu des développeurs Visual-Basic dont il est très proche. Tout comme JSP, ASP permet d'insérer des blocs de script dans du contenu HTML. Pour combler les lacunes d'ASP, Microsoft a développé la plate-forme « .NET » qui permet d'utiliser les autres langages de développement du monde Microsoft, tel que C#, pour la génération dynamique de contenu HTML. L'exécution de code ASP ou .Net est limitée aux serveurs Windows.

PHP (sigle récursif pour « PHP: Hypertext Preprocessor ») est un langage proche de Perl et des scripts Shell, ce qui a fait son succès auprès de la communauté du monde Unix. C'est ce langage qui sera utilisé pour les exemples du présent document.

II-C-2 - Serveurs de données

Les données étant principalement gérées par des serveurs dédiés, les langages cités précédemment offrent des moyens d'interagir avec eux. Les systèmes de gestion de bases de données permettent d'interroger les données et de les mettre à jour. Le langage le plus répandu est SQL (Structured Query Language) pour les bases de données relationnelles. La base de données MySQL sera le moteur relationnel utilisé pour les exemples du présent document. Le Web 2.0 a apporté de nouveaux besoins auxquels le modèle relationnel ne peut pas répondre. Les modèles NoSQL (Not Only SQL) ont alors pu émerger.

Les informations sur les personnes et ressources ainsi que les informations nécessaires à l'authentification peuvent être stockées et gérées par des annuaires LDAP (Lightweight Directory Access Protocol). Il s'agit d'un service qui peut être interrogé par le protocole LDAP de la couche « Application ».

Les services Web sont des applications Web dont le but est de fournir des données selon une structure prédéfinie et des services à une autre application en utilisant les protocoles standards d'Internet.

III - Failles de Sécurité

III-A - Menaces et risques applicatifs

III-A-1 - Types d'attaques

Le WASC établit dans son rapport « WASC Threat Classification » une liste exhaustive des menaces qui pèsent sur la sécurité des applications Web. Elles sont regroupées en six catégories définies dans la version 2004 de ce rapport.

La catégorie « authentification » regroupe les attaques de sites Web dont la cible est le système de validation de l'identité d'un utilisateur, d'un service ou d'une application.

La catégorie « autorisation » couvre l'ensemble des attaques de sites Web dont la cible est le système de vérification des droits d'un utilisateur, d'un service ou d'une application pour effectuer une action dans l'application.

La catégorie « attaques côté client » rassemble les attaques visant l'utilisateur pendant qu'il utilise l'application.

La catégorie « exécution de commandes » englobe toutes les attaques qui permettent d'exécuter des commandes sur un des composants de l'architecture du site Web.

La catégorie « révélation d'informations » définit l'ensemble des attaques permettant de découvrir des informations ou des fonctionnalités cachées.

La catégorie « attaques logiques » caractérise les attaques qui utilisent les processus applicatifs (système de changement de mot de passe, système de création de compte...) à des fins hostiles.

III-A-2 - Risques de sécurité

Contrairement au WASC qui décrit toutes les attaques possibles sur une application Web, l'OWASP ne traite que les dix plus grands risques de sécurité. Le rapport « OWASP Top 10 » permet ainsi à l'équipe projet de se focaliser sur la protection de l'application Web face aux menaces les plus importantes, ce qui est moins coûteux et plus facilement réalisable que d'essayer de se protéger de tous les dangers. L'OWASP établit le classement 2010 ci-dessous, dont chacune des failles est développée dans les chapitres suivants.

Une faille d'injection 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 inhabituelles ou à accéder à des données non autorisées.

Les failles de Cross-Site Scripting (XSS) se produisent chaque fois qu'une application prend des données non fiables et les envoie à un navigateur Web sans validation. 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.

Les failles de violation de gestion d'authentification et de session se produisent quand les fonctions correspondantes ne sont pas mises en œuvre correctement, permettant aux attaquants de compromettre les mots de passe, clés, jetons de session, ou d'exploiter d'autres failles d'implémentation pour s'approprier les identités d'autres utilisateurs.

Une faille de référence directe à un objet se produit quand un développeur expose une référence à une variable interne, tels un nom de fichier, de 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.

Une attaque par falsification de requête intersites (CSRF) force le navigateur d'une victime authentifiée à envoyer une requête HTTP, 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, l'application vulnérable considérant alors qu'elles émanent légitimement de la victime.

Une faille due à une mauvaise configuration de sécurité se produit quand les serveurs d'application, serveurs Web, serveur de base de données, et la plate-forme n'ont pas de configuration sécurisée correctement établie et déployée. Tous ces paramètres doivent être définis, mis en œuvre, et maintenus. Ceci implique de maintenir tous les logiciels à jour, notamment toutes les bibliothèques de code employées par l'application.

Une faille de stockage de données non sécurisées se produit quand une application Web ne protège pas correctement les données sensibles, telles que les numéros de cartes de crédit, de sécurité sociale, les informations d'authentification, avec un algorithme de chiffrement ou de hash approprié. Les pirates peuvent voler ou modifier ces données faiblement protégées pour perpétrer un vol d'identité et d'autres crimes, tels que la fraude à la carte de crédit.

La défaillance dans la restriction des accès à une URL se produit quand une application Web ne protège pas l'accès aux URL. Les applications doivent effectuer des contrôles d'accès similaires chaque fois que ces pages sont accédées, sinon les attaquants seront en mesure de forger des URL pour accéder à ces pages cachées.

La faille de protection de la couche transport se produit quand les applications ne peuvent pas chiffrer et protéger la confidentialité et l'intégrité du trafic réseau sensible. De plus, quand elles le font, elles supportent parfois des algorithmes faibles, utilisent des certificats expirés ou invalides, ou ne les emploient pas correctement.

Une faille de redirection et renvoi non validés se produit quand une application Web réoriente les utilisateurs vers d'autres pages et sites Web, et utilise des données non fiables pour déterminer les pages de destination. Sans validation appropriée, les attaquants peuvent rediriger les victimes vers des sites de phishing ou de logiciel malveillant, ou utiliser les renvois pour accéder à des pages non autorisées.

III-A-3 - Correspondances entre les définitions de l'OWASP et du WASC

Le tableau ci-dessous fait le lien entre les catégories et les attaques définies par le WASC dans le rapport de 2010 et les menaces identifiées par l'OWASP dans le classement de 2010.

Risques identifiés par l'OWASP en 2010	Attaques identifiées par le WASC en 2010	Catégories
Injection	SQL Injection (WASC-19) XML Injection (WASC-23) Null Byte Injection (WASC-28) LDAP Injection (WASC-29) Mail Command Injection (WASC-30) OS Commanding (WASC-31) XPath Injection (WASC-39) XQuery Injection (WASC-46)	Exécution de commandes
Cross-Site Scripting	Cross-Site Scripting (WASC-08)	Attaques côté client
Violation de Gestion d'Authentification et de Session	Insufficient Authentication (WASC-01) Brute Force (WASC-11)	Autorisation, authentification

	Credential/Session Prediction (WASC-18) Session Fixation (WASC-37) Insufficient Session Expiration (WASC-47)	
Référence directe à un objet non sécurisée	Insufficient Authentication (WASC-01) Insufficient Authorization (WASC-02) Path Traversal (WASC-33) Predictable Location (WASC-34)	Autorisation, authentification, révélation d'informations
Falsification de requêtes intersites	Cross-Site Request Forgery (WASC-09)	Attaques logiques
Gestion de configuration non sécurisée	Server Misconfiguration (WASC-14) Application Misconfiguration (WASC-15)	Révélation d'informations
Stockage de données non sécurisé	Insufficient Data Protection (WASC-50)	Révélation d'informations
Défaillance de restriction d'accès à une URL	Insufficient Authorization (WASC-02) Denial of Service (WASC-10) Brute Force (WASC-11) Insufficient Anti-automation (WASC-21) Predictable Location (WASC-34)	Autorisation, authentification, révélation d'informations
Communications non sécurisées	Insufficient Transport Layer Protection (WASC-04)	Révélation d'informations
Redirection et renvoi non validés	URL Redirector Abuse (WASC-38)	Attaques logiques

III-B - Injection

III-B-1 - Principe

L'attaque par injection est évaluée par l'OWASP comme étant la plus risquée, car la faille est assez répandue, il est facile de l'exploiter et l'impact peut être très important. Cela va de la simple récupération de données à la prise totale de contrôle du serveur. La victime de l'attaque est un des composants techniques de l'application Web.

Contensin [6] explique que pour réaliser une attaque de ce type, il faut injecter dans les zones de saisie classiques présentées à l'utilisateur du code malicieux. Ces données seront interprétées comme des instructions par un des composants de l'application Web. Les champs de formulaires peuvent être protégés par JavaScript pour vérifier que les valeurs saisies correspondent à ce qui est attendu. Cependant, J. Scambray, V. Liu et C. Sima [7] démontrent qu'il est possible d'outrepasser ces vérifications en faisant appel à un serveur proxy personnel, par exemple, qui permettra d'intercepter les requêtes pour les modifier et envoyer le code malicieux. La difficulté de l'attaque réside finalement dans la détection des technologies utilisées pour formuler le code d'attaque adéquat. Cependant, la plupart des applications Web de gestion de contenu présentes sur le Web sont basées sur des projets Open Source. H. Dwivedi, A. Stamos, Z. Lackey et R. Cannings [8] montrent qu'il est alors facile d'identifier les technologies employées en parcourant le code source mis à disposition. De plus, il existe des outils d'injection automatique disponibles sur le Web, rendant le risque plus élevé. L'exploitation de la faille devient automatisable.

III-B-2 - Exemples d'attaque

L'attaque par injection SQL consiste à injecter du code SQL qui sera interprété par le moteur de base de données. Le code malicieux le plus répandu est d'ajouter une instruction pour faire en sorte que la requête sous-jacente soit toujours positive. Cela permet par exemple d'usurper une identité pour se connecter à une application Web, de rendre l'application inutilisable ou de supprimer toutes les données de la table visée, voire de la base de données complète. L'exemple suivant va interroger une table qui contient la liste des cartes bancaires enregistrées dans la base de données de l'application Web d'un site marchand. Le script de création de cette table est le suivant :

Figure 6 - Script SQL de création de la table « comptes »

```
CREATE TABLE IF NOT EXISTS `comptes` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'identifiant',
  `nom` varchar(30) NOT NULL COMMENT 'nom d'utilisateur',
  `motdepasse` varchar(41) NOT NULL,
  `typecarte` varchar(30) NOT NULL COMMENT 'type de carte',
  `numerocarte` varchar(30) NOT NULL COMMENT 'numéro de carte',
  PRIMARY KEY (`id`),
  UNIQUE KEY `nom` (`nom`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=5 ;
```

Pour afficher le numéro de carte bancaire, l'utilisateur doit s'authentifier. La requête SQL suivante permet de vérifier que le couple utilisateur « user4 »/mot de passe du compte « eng111 » est correct et si tel est le cas renvoie le numéro de carte bancaire :

Figure 7 - Requête SQL pour l'authentification et affichage du numéro de carte

```
SELECT `numerocarte`
FROM `comptes`
WHERE `nom` = 'user4'
AND `motdepasse` = PASSWORD( 'eng111' )
```

Le script PHP pour exploiter cette requête de façon dynamique avec les informations fournies par l'utilisateur est le suivant :

Figure 8 - Script PHP pour l'authentification et l'affichage du numéro de carte

```
<?php //connexion a la base de donnees
mysql_connect('localhost', 'root', '');
mysql_select_db('eng111');
//recuperation des parametres
$nom = $_GET['nom'];
$motdepasse = $_GET['motdepasse'];
//generation de la requete
$requeteSQL = "SELECT numerocarte FROM comptes WHERE nom = '$nom' AND motdepasse = PASSWORD( '$motdepasse' )";
//execution de la requete
$reponse = mysql_query($requeteSQL);
$resultat = mysql_fetch_assoc($reponse);
//affichage du resultat
echo $resultat['numerocarte'];
?>
```

En remplissant le formulaire avec la valeur « ' OR 1=1 -- ' » pour le champ « nom » et n'importe quelle valeur pour le mot de passe, la requête qui sera envoyée à la base de données devient :

Figure 9 - Requête SQL incluant du code frauduleux d'injection

```
SELECT numerocarte FROM comptes WHERE nom = '' OR 1=1 -- '' AND motdepasse = PASSWORD( 'x' )
```

Ainsi la condition 1=1 est toujours vérifiée et le reste de la commande est mis en commentaire grâce à la chaîne de caractères « -- ». Cela permet donc de récupérer aléatoirement un numéro de carte.

L'attaque par injection de XPath suit le même principe que pour SQL [9]. En effet, XPath est un langage de requête pour gérer les données stockées au format XML, comme le fait SQL pour les bases de données relationnelles. XPath

et Xquery, dont XPath est un sous-ensemble, souffrent donc des mêmes vulnérabilités face à l'injection de code malicieux.

L'attaque par injection LDAP permet d'accéder à des informations privées qui sont enregistrées dans l'annuaire d'entreprise. En modifiant le comportement du filtrage dans la requête LDAP qui sera générée, il est possible de récupérer la liste exhaustive des adresses de courrier électronique d'une entreprise pour les saturer de spam par exemple.

L'attaque par injection de commandes est surtout principalement possible sur les scripts CGI écrits en Perl, PHP et Shell. Il est possible de prendre le contrôle du serveur. Il faut pour cela faire en sorte que la commande initiale soit exécutée sans problème et ajouter des commandes du système d'exploitation du serveur qui seront exécutées par le serveur.

L'attaque par traversée de répertoire permet d'accéder à des fichiers présents sur le serveur. Les fichiers cibles privilégiés étant ceux contenant des informations de sécurité comme le fichier des mots de passe ou les fichiers contenant les clés privées de chiffrement pour SSL par exemple. Cette attaque est rendue possible si l'application Web inclut du contenu de fichier en passant l'adresse de ce fichier en paramètres de la requête.

Les attaques XXE (XML eXternal Entity) sont un dérivé des attaques par traversée de répertoire. Les conséquences vis-à-vis des fichiers présents sur les serveurs sont donc les mêmes. Ce type d'attaque est basé sur la fonctionnalité de XML « entités externes ». Les entités sont des substituts pour des séquences d'information. Elles sont équivalentes aux variables dans les langages de programmation. Les entités externes permettent de déclarer des documents dont le contenu sera affiché lors de l'utilisation de l'entité. Si l'entité pointe sur un fichier existant sur le serveur, son contenu pourra être divulgué à l'attaquant. Cette fonctionnalité peut être exploitée en plaçant un fichier XML au format RSS sur un site et de l'intégrer à un agrégateur en ligne. Si ce dernier est vulnérable, il sera alors possible de voir le contenu des fichiers demandés par l'attaquant.

III-B-3 - Parade et bonnes pratiques

Les différentes attaques citées précédemment reposent principalement sur l'utilisation de caractères spécifiques qui permettent de mettre en commentaire des portions de code et d'insérer du code frauduleux. Il est cependant rare que l'application ait besoin d'accepter les caractères suivants :

Figure 10 - Caractères spéciaux communément utilisés dans les attaques d'injection

```
& ~ " # ' { } ( [ ] ( ) - | ` _ \ ^ @ \ * / . < > , ; : ! $
```

Cependant les applications Web de gestion de contenu comme les forums doivent les accepter, notamment les forums utilisés par les développeurs pour partager du code. Dans ce cas, il faut transformer au moins les caractères ci-dessus en code HTML avant de les stocker dans la base de données. L'affichage de l'information ne sera pas différent pour l'utilisateur, mais les données seront plus sûres.

Bien qu'un site puisse subir différents types d'attaques par injection, il suffit de vérifier que les caractères utilisés sont ceux attendus. Ce contrôle doit être effectué au niveau du client grâce à JavaScript et au niveau du serveur lorsque les paramètres sont récupérés pour fermer la faille de sécurité. Par exemple, le langage PHP offre une fonctionnalité qui permet de transformer ces caractères.

Figure 11 - Script PHP pour remplacer les caractères par le code HTML

```
<?php
$nouvelleValeur=htmlspecialchars($valeurParametre, ENT_QUOTES);
?>
```

De plus il faut vérifier que les valeurs sont bien du type et du format attendus (longueur, intervalle de valeur...).

L'ANSSI porte une attention particulière aux outils automatiques d'exploitation des failles SQL dans son bulletin de sécurité CERTA-2011-ACT-045. Pour déterminer si un site est victime de ce type d'agression, il faut vérifier dans

les journaux d'activité du serveur HTTP qu'il n'y a pas d'événement inhabituel, tel qu'un nombre de requêtes HTTP anormalement élevé ou des requêtes ayant pour paramètres des valeurs inappropriées.

III-C - Cross-Site Scripting (XSS)

III-C-1 - Principe

L'OWASP considère la vulnérabilité à XSS comme une faille critique car elle est très répandue et facile à détecter. Les attaques s'appuient principalement sur les formulaires des applications Web. Les victimes sont les utilisateurs des applications Web vulnérables. L'ANSSI signale dans la note d'information CERTA-2002-INF-001-001 que les scripts frauduleux peuvent endommager la base de registre de la victime, afficher des formulaires dont les saisies seront envoyées à l'attaquant, récupérer les cookies présents sur la machine de la victime, exécuter des commandes systèmes et construire des liens déguisés vers des sites malveillants.

Y.-W. Huang, C.-H. Tsai, T.-P. Lin, S.-K. H., D.T. Lee et S.-Y. Kuo [10] indiquent que l'attaque XSS est également une attaque par injection car l'objectif de l'attaquant est de soumettre un code frauduleux à l'application. A. Kiezun, P. J. Guo, K. Jayaraman, M. D. Ernst [11] montrent qu'il existe en fait deux types d'attaque XSS.

L'attaque XSS par réflexion (reflected XSS) s'appuie sur le fait que l'application Web affiche ce que l'utilisateur vient de saisir dans un formulaire dans une page de résultat. Le navigateur de la victime exécute alors le code frauduleux généré dans la page de résultat. Tous les champs de formulaire sont donc une faille de sécurité potentielle que l'attaquant peut exploiter par XSS. L'attaquant crée un lien déguisé vers l'application Web dont un des paramètres contient du code JavaScript frauduleux. En utilisant ce lien, la victime fait exécuter par son navigateur le code JavaScript. Le Web 2.0 et ses systèmes de gestion de contenu ont popularisé cette attaque en permettant de publier des liens aisément et visibles sur tout le Web.

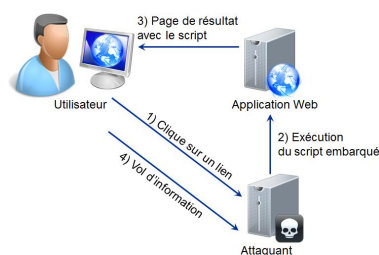


Figure 12 - Principe d'une attaque XSS par réflexion

L'attaque XSS stockée (stored XSS) s'appuie sur le fait que l'attaquant réussisse à stocker dans la base de données du code frauduleux qui sera exécuté par la victime lorsqu'elle tentera d'afficher la donnée malveillante. Cette attaque est plus dangereuse que la première car le code fait partie intégrante des données de l'application Web et peut atteindre plusieurs victimes.

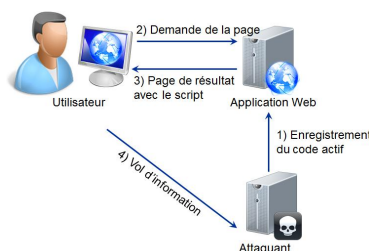


Figure 13 - Principe d'une attaque XSS stockée

III-C-2 - Exemples d'attaque

L'attaque XSS par réflexion peut être implémentée par différents moyens.

Le plus facile est d'utiliser un moteur de recherche vulnérable. Par exemple les outils de forum intègrent des formulaires pour recherche des messages par leur contenu. La page de résultat reprend généralement les mots-clés saisis. Il suffit alors de mettre comme paramètre de recherche un code JavaScript qui sera ensuite interprété par le navigateur de la victime. Pour réaliser cette attaque, il suffit de laisser un lien qui aura pour paramètre le code malveillant.

Figure 14 - Exemple de lien malveillant exploitant une faille XSS

```
http://www.forum-vulnérable.com/recherche.php?parametre=<script>alert(&#8216;attaque XSS')</script>
```

En cliquant sur ce lien, la victime lancera la recherche. Puis le moteur de recherche affichera le paramètre « <script>alert('attaque XSS')</script> » qui sera exécuté par le navigateur.

Des applications Web sont responsables de l'affichage des courriers électroniques : les webmails. Pour consulter son courrier, l'utilisateur va préalablement s'authentifier et ses informations d'identification seront stockées dans des cookies. Un courrier malveillant peut intégrer du code JavaScript qui sera interprété par le navigateur. Ce code sera capable de récupérer les cookies et envoyer les informations à l'attaquant.

L'attaque XSS stockée injecte du code malveillant dans la base de données. L'application Web suivante de type forum va permettre d'illustrer cette attaque. La table support de la démonstration est la suivante :

Figure 15 - Script SQL de création de la table « messages »

```
CREATE TABLE IF NOT EXISTS `messages` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'identifiant',
  `numerosujet` int(11) NOT NULL COMMENT 'numero du sujet',
  `redacteur` varchar(30) NOT NULL COMMENT 'nom du redacteur du message',
  `message` varchar(4000) NOT NULL COMMENT 'contenu du message',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;
```

Le script PHP suivant est responsable de l'enregistrement d'un message. Comme une des informations saisies (le nom du rédacteur) est réaffichée, cela implique que ce code est vulnérable à une attaque XSS par réflexion.

Figure 16 - Script PHP pour l'insertion dans la table « messages »

```
<?php
//recuperation des parametres
$message=$_GET['message'];
$nom=$_GET['nom'];
$numsubject=$_GET['numsubject'];
//generation de la requete
$requeteSQL = "INSERT INTO messages VALUES (NULL, '$numsubject', '$nom', '$message')";
//execution de la requete
$reponse = mysql_query($requeteSQL);
//affichage du resultat
echo "<tr><td>&nbsp;</td><td>Merci $nom de votre participation. Vous venez de saisir : $message</td></tr>";
?>
```

En saisissant comme message un code JavaScript malveillant, il sera enregistré dans la base de données.

Le script PHP suivant est responsable de l'affichage de l'ensemble des messages d'un sujet.

Figure 17 - Script PHP pour la recherche dans la table « messages »

```
<?php
//recuperation des parametres
$numsubject=$_GET['searchsujet'];
//generation de la requete
$requeteSQL = "SELECT * FROM messages WHERE numerosujet=$numsubject order by id";
//execution de la requete
$reponse = mysql_query($requeteSQL);
//affichage du resultat
echo "<tr><td> Sujet $numsubject</td><td>";
while($resultat = mysql_fetch_assoc($reponse)) {
  echo $resultat['redacteur'] . " : " . $resultat['message'] . "<br>";
}
```


Figure 17 - Script PHP pour la recherche dans la table « messages »

```
}  
echo "</td></tr>";  
?>
```

Lorsque des utilisateurs afficheront le fil des messages, le message frauduleux sera automatiquement envoyé aux navigateurs et interprété créant une attaque XSS.

III-C-3 - Parade et bonnes pratiques

Les recommandations faites précédemment pour se prémunir des risques d'injection sont valables pour XSS. Cependant, transformer les six caractères douteux suivants suffit.

Figure 18 - Caractères spéciaux à remplacer par leur code

```
& &#61664; &amp;  
< &#61664; &lt;  
> &#61664; &gt;  
" &#61664; &quot;  
' &#61664; &#x27; (&apos; n'est pas recommandé)  
/ &#61664; &#x2F;
```

Côté client avec JavaScript il faut vérifier les données saisies par les utilisateurs. Côté serveur, il faut vérifier les données récupérées en paramètre. Il faut rejeter toutes les données qui ne sont pas conformes à ce qui est attendu.

Pour éviter le vol de cookies par du code JavaScript, il est possible de positionner l'attribut de cookie HTTPOnly [8]. S'il est présent, le navigateur interdit au moteur JavaScript de lire ou écrire dans les cookies. Cet attribut est très peu utilisé par les applications Web, car tous les navigateurs ne le gèrent pas. Cependant il est préférable de l'utiliser car les navigateurs les plus populaires l'implémentent, ce qui diminue les risques liés aux cookies.

Figure 19 - Script PHP pour positionner l'attribut de cookie HTTPOnly

```
<?php  
session.cookie_httponly = True  
?>
```

Les navigateurs intègrent des protections contre XSS en interdisant l'exécution de code JavaScript qui modifie une page Web depuis une page Web ne portant pas le même nom de domaine.

III-D - Violation de gestion d'authentification et de session

III-D-1 - Principe

Cette faille de sécurité regroupe toutes les vulnérabilités pouvant mener à une usurpation d'identité. Ces points de faiblesse dans les applications Web peuvent ouvrir à des attaquants des accès à des fonctionnalités des applications Web auxquelles ils n'ont pas le droit normalement. Cela peut donc leur permettre de voler des informations ou d'endommager le bon fonctionnement de l'application. La protection des accès à l'application repose généralement sur un système d'authentification. La plupart du temps, le système d'authentification est redéveloppé pour chaque application, ce qui implique que ces systèmes ne bénéficient pas de l'expérience acquise sur le développement d'autres applications.

Pour comprendre comment les attaques peuvent être menées, il faut comprendre le mécanisme d'authentification le plus commun des applications Web :

- 1 L'utilisateur non authentifié demande l'accès à une page Web ;
- 2 Le serveur renvoie une page d'authentification ;
- 3 L'utilisateur remplit le formulaire en fournissant un identifiant et un mot de passe et revoie ces informations au serveur Web ;

- 4 Le serveur Web fait appel à un service pour vérifier la validité du couple identifiant/mot de passe ;
- 5 Si la validité est avérée, le serveur Web fournit un identifiant de session à l'utilisateur. Comme expliqué précédemment HTTP est un protocole déconnecté, donc entre deux requêtes HTTP la connexion entre le navigateur et le serveur HTTP est coupée. Donc le serveur HTTP ne peut pas reconnaître un utilisateur qui s'est déjà authentifié et qui a ouvert une session de travail dans l'application Web. Pour remédier à cela, la plupart des systèmes d'authentification reposent sur un identifiant de session. Celui-ci est envoyé à chaque page entre le client et le serveur par le biais d'un cookie, d'un paramètre d'adresse ou d'un champ de formulaire invisible pour l'utilisateur ;
- 6 L'utilisateur peut utiliser l'application Web tant que la session est ouverte.

Les attaques pour usurper une identité peuvent être regroupées en deux catégories :

- les attaques contre le système d'authentification qui cherchent à obtenir un droit d'accès ;
- les usurpations de session qui permettent de s'affranchir de l'étape d'authentification.

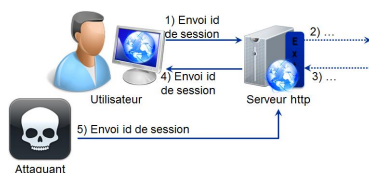


Figure 20 - Principe de détournement de session

III-D-2 - Exemples d'attaque

Parmi les attaques contre les systèmes d'authentification, la plus répandue est l'utilisation de la force brute. Pour cela l'attaquant va bombarder la page d'authentification avec des valeurs d'identifiant et de mots de passe jusqu'à ce qu'il se fasse accepter [7]. L'attaque est facilitée si le message d'erreur de l'échec de l'authentification donne l'origine de l'erreur. Ainsi « l'utilisateur n'existe pas » permet à l'attaquant de ne pas tenter d'entrer des mots de passe pour cet utilisateur absent de la base de compte. « Mot de passe incorrect » permet à l'attaquant de se concentrer sur cet utilisateur, ce qui lui fait gagner beaucoup de temps. De même si l'application Web offre un service pour créer un compte par lui-même et qu'au moment de la saisie de l'identifiant ce système indique si le compte existe déjà ou non, l'attaquant dispose d'un moyen pour trouver des comptes attaquables. L'impact de ce type d'attaque n'est pas seulement limité à une usurpation d'identité pour l'application Web. Un internaute utilise souvent les mêmes valeurs d'identifiant et de mot de passe pour de nombreuses applications présentes sur le Web. L'attaquant peut donc tenter d'utiliser ces valeurs sur différentes applications Web.

Il est possible pour l'attaquant de chercher des couples identifiant/mot de passe sans faire appel à la force brute. L'attaquant va simplement tenter d'utiliser des comptes généralement présents dans les applications Web, comme ceux d'administration. Ceci est surtout possible lorsque les applications sont basées sur des outils Open Source qui ont des comptes créés automatiquement avec des mots de passe par défaut connus du domaine public. L'attaquant n'a plus qu'à consulter le code source pour trouver une liste restreinte d'identifiants/mots de passe valides.

Les pages Web qui permettent de réinitialiser les mots de passe sont une faille importante pour l'usurpation d'identité. En effet, pour s'assurer de l'identité du demandeur, la plupart d'entre elles demandent une information que seule la personne est censée connaître. Or avec les réseaux sociaux, les internautes partagent leur vie privée. Ces informations personnelles visibles de tous peuvent être les mêmes que celles demandées dans les pages de réinitialisation de mot de passe. Dans ce cas l'attaquant peut définir un nouveau mot de passe qu'il pourra utiliser pour se connecter à l'application.

Pour voler un identifiant de session, l'attaque par la force brute est également possible. Dans ce cas l'attaquant va générer des valeurs et tenter de les utiliser comme identifiant de session. S'il réussit à trouver une valeur valide, il pourra utiliser l'application Web sans s'être authentifié. Par ailleurs, une attaque XSS peut permettre de récupérer un identifiant de session présent dans le cookie de l'internaute. L'identifiant de session peut également être découvert par un attaquant en écoutant la transmission de données sur le réseau, en consultant les fichiers de journalisation sur le serveur par injection de commandes systèmes par exemple.

Une autre technique consiste à fournir un identifiant de session à la victime, par hameçonnage par exemple. L'utilisateur se connecte à l'application Web et s'authentifie. La session est créée en utilisant l'identifiant fourni. L'attaquant peut alors accéder au site en fournissant l'identifiant fixé. L'identifiant peut être généré ou obtenu en émettant une requête vers l'application cible de l'attaque si l'application retourne un identifiant de session pour toute requête avant même l'authentification de l'utilisateur.

Si l'identifiant de session n'est pas généré aléatoirement mais est un nombre incrémenté à chaque ouverture de session, par exemple, l'attaquant peut arriver à deviner un identifiant valide.

III-D-3 - Parade et bonnes pratiques

Concernant les systèmes d'authentification, l'application ne doit accepter que des mots de passe suffisamment forts pour éviter d'être devinés rapidement par la force brute. Une longueur minimale de huit caractères est ce qui est recommandé par l'OWASP pour les applications critiques. De plus il doit comporter au moins un chiffre, une lettre en minuscule et une lettre en majuscule.

Le message affiché lors d'un problème de validité de l'identifiant ou du mot de passe doit être générique et ne doit pas donner d'indice quant à l'origine de l'erreur.

Pour contrer les attaques de force brute, le compte ciblé par l'attaque doit être verrouillé après cinq tentatives consécutives infructueuses de connexion. La procédure de déverrouillage peut être automatique après un laps de temps prédéfini ou manuelle par un administrateur de l'application.

Dans la mesure du possible il est conseillé de ne pas développer son propre mécanisme d'authentification. Il est préférable d'utiliser un système existant éprouvé.

Concernant les identifiants de session, les applications Web doivent limiter la durée de vie d'une session. Une période d'inactivité maximale doit être définie. Si l'utilisateur n'utilise pas l'application pendant ce laps de temps, la session devient inutilisable et l'utilisateur doit se reconnecter. Une session doit également avoir une durée de vie maximale au-delà de laquelle la session expire, même si la période d'inactivité autorisée n'était pas dépassée. Ces précautions permettent de limiter le temps d'action d'un attaquant.

Du côté client, du code JavaScript doit fermer la session lorsque l'utilisateur ferme le navigateur. Cela permet de simuler une action de l'utilisateur pour se déconnecter de l'application. Pour que l'utilisateur évite de perdre des saisies non sauvegardées, du code JavaScript peut prévenir l'utilisateur que sa session va bientôt expirer.

L'identifiant de session doit être généré automatiquement et être suffisamment long pour se prémunir des vols par prédiction.

Pour détecter des attaques de force brute, il faut régulièrement consulter les journaux d'activité à la recherche d'événements inhabituels, comme un nombre important de requêtes utilisant des identifiants de sessions invalides.

III-E - Référence directe non sécurisée à un objet

III-E-1 - Principe

Cette vulnérabilité existe simplement parce que les paramètres de requêtes ne sont pas vérifiés avant traitement. Si le paramètre vulnérable fait référence à un fichier ou à une valeur dans une base de données, il suffit de reconstruire la requête avec une valeur de paramètre normalement interdite pour y avoir accès.

Cette faille peut avoir des impacts importants si un utilisateur mal intentionné obtient par ce biais des accès à des informations et des fonctionnalités pour lesquelles il n'a aucune autorisation.

III-E-2 - Exemples d'attaque

Cette faille est une des bases de la vulnérabilité exploitée par XSS. En effet, dans les exemples d'attaques exposés précédemment (voir paragraphe 3.3.2), les paramètres récupérés ne sont pas vérifiés. Par contre, si ces valeurs avaient été contrôlées, les caractères spéciaux n'auraient pas été autorisés, empêchant ainsi l'envoi de code frauduleux au navigateur.

Si les paramètres sont passés en paramètre d'un lien, un utilisateur malintentionné peut aisément modifier l'adresse pour accéder à des informations auxquelles il n'aurait pas dû avoir accès. L'exemple suivant reprend la table « comptes » (voir paragraphe 3.2.2) qui va être interrogée par un script PHP pour afficher le numéro de carte bancaire de l'utilisateur.

Figure 21 - Script PHP pour l'affichage du numéro de carte

```
<?php
//recuperation des parametres
$nom = $_GET['proprietaire'];
//generation de la requete
$requeteSQL = "SELECT numerocarte FROM comptes WHERE nom = '$nom'"; //execution de la requete
$reponse = mysql_query($requeteSQL);
$resultat = mysql_fetch_assoc($reponse);
//affichage du resultat
echo "<tr><td> Votre numero de carte est :</td><td>";
echo $resultat['numerocarte'];
echo "</td></tr>";
?>
```

Si l'utilisateur malveillant saisit dans son navigateur l'adresse de cette page avec pour paramètre « nom=nom_de_la_victime », il a alors accès au numéro de carte bancaire qu'il n'aurait jamais dû pouvoir voir.

III-E-3 - Parade et bonnes pratiques

Pour protéger les données les plus confidentielles ou les fonctionnalités les plus avancées, le WASC recommande de demander à l'utilisateur de saisir à nouveau son identifiant et son mot avant de pouvoir y accéder. Ensuite il suffit de se baser sur ces valeurs pour construire les requêtes. Ainsi dans l'exemple de l'affichage du numéro de carte, le paramètre utilisé pour la recherche aurait été celui de la personne qui s'est authentifiée et non celui fourni en paramètre du lien.

III-F - Falsification de requête intersites (CSRF)

III-F-1 - Principe

D. Gollmann [12] montre qu'une attaque par falsification de requête intersites (Cross-Site Request Forgery ou session riding ou CSRF ou XSRF) a un fonctionnement assez proche d'une attaque XSS. La principale différence est que l'utilisateur au travers de son navigateur ne sera pas la victime mais sera celui qui va effectuer une action malveillante sur l'application cible. Une attaque CSRF va exécuter du code malveillant dans une application Web au travers de la session d'un utilisateur connecté.

Comme pour XSS, il existe deux modes opératoires.

Dans une attaque CSRF par réflexion (reflected CSRF), l'attaquant crée une page Web qui comporte un formulaire invisible par exemple. Ce dernier contient un script caché qui lance des actions de l'application. L'attaquant piège l'utilisateur en mettant un lien vers cette page dans un courrier électronique ou sur des réseaux sociaux. Quand l'utilisateur affiche cette page, le navigateur va interpréter le code malicieux et va tenter d'exécuter une fonctionnalité de l'application cible. Si l'utilisateur s'y est récemment connecté, l'application va exécuter la commande sans le consentement de l'utilisateur. Cette attaque fonctionne car les informations d'authentification qui ont préalablement été saisies par l'utilisateur sont envoyées automatiquement par le navigateur au serveur. L'attaquant n'a donc

pas besoin de se connecter à l'application pour exécuter des commandes frauduleuses. Cependant l'attaque ne fonctionne pas si l'utilisateur ne s'est pas connecté.

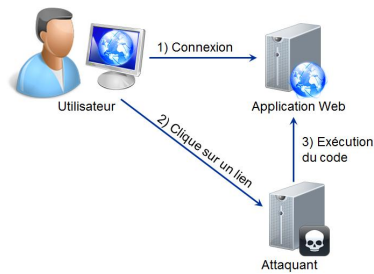


Figure 22 - Principe d'une attaque CSRF par réflexion

Dans une attaque CSRF stockée (stored CSRF), c'est l'application elle-même qui présente le code malicieux à l'utilisateur. Pour ce faire l'attaquant a réussi à insérer du code malicieux dans les données de l'application Web. Chaque fois qu'un utilisateur parcourt la page qui va présenter ce code, le navigateur va l'interpréter et par conséquent va exécuter une commande de l'application. L'application va alors accepter d'exécuter cet ordre comme si la demande provenait de l'utilisateur. Cette attaque a plus de chances de réussir car l'utilisateur s'est déjà connecté et utilise l'application. L'attaquant n'a pas de besoin de piéger un utilisateur.



Figure 23 - Principe d'une attaque CSRF stockée

Une attaque par CSRF rend les défenses contre les attaques XSS inopérantes.

III-F-2 - Exemples d'attaque

L'exemple suivant reprend la table « messages » et le script PHP pour l'insertion de données (voir paragraphe 3.3.2). Si ce script se nomme « add_message.php », le site de l'attaquant pourra utiliser le code suivant pour le faire exécuter par l'utilisateur :

Figure 24 - Code HTML pour réaliser une attaque CSRF

```

<form method="GET" id="reflected_CSRF" name="reflected_CSRF" action="add_message.php">
  <input type="hidden" name="numsjet" value="6">
  <input type="hidden" name="nom" value="CSRF">
  <input type="hidden" name="message" value="action frauduleuse">
</form>
<script>document.reflected_CSRF.submit()</script>
  
```

L'utilisateur en parcourant la page de l'attaquant est alors automatiquement redirigé vers la page « add_message.php » avec les paramètres numsjet=6, nom=CSRF et message=action frauduleuse.

III-F-3 - Parade et bonnes pratiques

Bien que XSS et CSRF soient proches dans le principe, se protéger des attaques XSS ne permet pas de se protéger des attaques CSRF.

Pour se protéger, il faut utiliser uniquement des requêtes POST. Les méthodes GET doivent être bannies. Attention toutefois, dans les servlet Java la méthode « doGet() » fait appel à la méthode « doPost() » en redirigeant l'ensemble

des paramètres. Dans ce cas l'utilisation de requêtes GET fonctionne. C'est pourquoi l'utilisation de POST n'est pas une protection suffisante.

Pour les pages qui manipulent des données sensibles, il faut demander à l'utilisateur de s'authentifier à nouveau. Cela permet de s'assurer que l'utilisateur est conscient de l'action et l'approuve.

L'utilisateur doit toujours vérifier que le lien sur lequel il clique est bien celui de l'application qu'il veut utiliser.

III-G - Mauvaise configuration de sécurité

III-G-1 - Principe

Cette faille de sécurité regroupe toutes les vulnérabilités laissées ouvertes aux différents niveaux de l'architecture de l'application Web. Pour chacun des serveurs impliqués dans l'activité de l'application, le problème concerne le système d'exploitation ainsi que les outils installés pour servir l'application.

Pour chacun de ces composants, des failles sont connues du domaine public, ce qui facilite les attaques. S'ils ne sont pas mis à jour, l'attaquant peut exploiter les failles non corrigées.

Pour de nombreux outils, des options sont installées par défaut alors qu'elles ne sont pas nécessaires au bon fonctionnement de l'application. Cette situation offre plus d'opportunités pour un attaquant.

De même de nombreuses applications sont installées avec des comptes créés avec des mots de passe par défaut. Ces comptes et mots de passe sont les cibles privilégiées des usurpations d'identité.

III-G-2 - Exemples d'attaque

J. Scambray, V. Liu et C. Sima [7] donnent un exemple d'attaque. En 2007 une faille est découverte dans l'extension mod_jk du serveur HTTP Apache. Ce module permet de renvoyer les requêtes HTTP reçues par le serveur HTTP au serveur de Servlet Apache Tomcat pour qu'il exécute les Servlets Java. Le problème détecté est un dépassement de mémoire tampon (buffer overflow). Le module ne gérait pas correctement les adresses trop longues contenues dans les requêtes HTTP. Cela permettait à un attaquant de faire ouvrir un port d'écoute spécifique utilisable pour prendre la main sur le serveur. Tant que le correctif n'était pas publié et installé, les systèmes restaient vulnérables. Le seul moyen de se protéger était de désactiver le module s'il n'était pas utilisé.

Les codes sources des applications Web Open Source sont disponibles aussi bien pour les développeurs légitimes que pour les attaquants. En parcourant ces fichiers, il est possible de lire des commentaires laissés par les développeurs indiquant qu'il y a un problème dont ils ont conscience mais qu'ils traiteront plus tard. Dans ce cas l'attaquant n'a plus qu'à exploiter cette faiblesse.

Les pages d'erreurs des serveurs HTTP contiennent par défaut des informations sur l'erreur et sur le serveur HTTP lui-même. En tentant d'accéder à une page inexistante, une erreur de type « 404 page not found » est retournée à l'attaquant, de même que la version du serveur HTTP. Dans ce cas il suffit à l'attaquant d'étudier cette version pour en connaître les vulnérabilités et de les exploiter.

III-G-3 - Parade et bonnes pratiques

L'ANSSI et l'OWASP font les recommandations suivantes.

Il faut désactiver les options inutiles des composants afin de diminuer le nombre de vulnérabilités potentielles que ce soit au niveau du système d'exploitation, du système de gestion de bases de données ou du serveur HTTP.

Il faut mettre à jour les différents composants de l'architecture autant que possible en installant les correctifs dès qu'ils sont publiés. De plus, à l'installation il est préférable de choisir la version anglaise plutôt qu'une autre langue. En effet, lors du développement de correctifs c'est toujours la version anglaise qui est privilégiée, les autres versions étant corrigées plus tardivement.

Après l'installation il faut désactiver voire supprimer tous les comptes inutiles. Le mot de passe des autres comptes doit être modifié dès que possible. Les comptes d'administration par défaut doivent être verrouillés. Il faut préférer l'utilisation de comptes d'administration créés manuellement.

III-H - Stockage de données cryptographiques non sécurisé

III-H-1 - Principe

Cette faille de sécurité englobe toutes les faiblesses liées à la protection du stockage des données. La meilleure protection est la mise en place du chiffrement des informations. Le CLUSIF définit le chiffrement comme « le procédé grâce auquel on peut rendre la compréhension d'un document impossible à toute personne qui n'en possède pas la clé. »

La principale faille concerne les données sensibles, c'est-à-dire les données dont la divulgation, l'altération ou la non-disponibilité peuvent porter préjudice à leur propriétaire, telles que le mot de passe ou l'identifiant de session. Si les données sont présentes en clair ou chiffrées par un algorithme faible, il existe un risque qu'un attaquant puisse les consulter.

III-H-2 - Exemples d'attaque

Certains moteurs de bases de données font payer l'option de chiffrement. Pour des raisons d'économies, les responsables décident de stocker les données en clair dans la base de données. Seules les transmissions de requête HTTP sur le réseau sont chiffrées. Pour se protéger du risque d'incendie sur le site où sont installés les serveurs, les données sauvegardées sur bande sont envoyées sur un autre site une fois par mois. Si un agresseur intercepte cette sauvegarde pendant son transfert, il aura accès aux données en clair.

III-H-3 - Parade et bonnes pratiques

Tous les moyens de stockage de données sensibles doivent être chiffrés. Il faut s'assurer également que la sauvegarde du moyen de stockage ne contient pas les données en clair.

Il ne faut pas utiliser d'algorithme de chiffrement ou de hachage faible, tel que MD5 ou SHA1 ni tenter de créer son propre algorithme. Il faut utiliser des algorithmes reconnus et éprouvés tels AES-256, RSA et SHA-256. Pour des raisons de capacité de calcul, l'ANSSI recommande que la taille minimale des clés symétriques utilisées jusqu'en 2020 soit de 100 bits et de 128 bits au-delà de 2020.

III-I - Défaillance dans la restriction des accès à une URL

III-I-1 - Principe

Cette faille permet à un utilisateur d'accéder à des fonctionnalités de l'application, voire des fichiers et répertoires du serveur HTTP sans y être habilité.

L'attaque par traversée de répertoires permet d'accéder à des fichiers du serveur HTTP notamment ceux contenant les clés privées de chiffrement. Les applications vulnérables ouvrent des fichiers dont le nom est donné en paramètre de la requête HTTP.

Une autre attaque consiste à deviner l'existence de fichiers ou de répertoires. En effet de nombreux outils disposent d'une interface d'administration dont l'adresse d'accès est du type « `http://www.site_vulnérable.fr/admin/admin.php` ». Dans ce cas même si l'utilisateur malintentionné n'y a pas accès au travers de l'application, il peut saisir directement l'adresse pour l'ouvrir. Les applications vulnérables ne demandent pas de s'authentifier avant de pouvoir l'utiliser, la seule protection étant qu'aucun lien n'est mis à disposition pour y accéder, ce qui n'est pas suffisant.

Une attaque équivalente consiste à ne pas spécifier de nom de fichier dans l'adresse, par exemple « `http://www.site_vulnérable.fr/admin/` ». Les serveurs HTTP vulnérables afficheront le contenu du répertoire.

III-I-2 - Exemples d'attaque

L'exemple suivant montre une portion de code vulnérable à l'attaque par traversée de chemin. L'application est construite pour inclure du texte en fonction de la langue du navigateur. Dans ce cas la langue est donnée en paramètre de la requête HTTP, la commande PHP « `include lang_nom_fichier.php` » permet d'inclure le contenu du fichier concerné.

Figure 25 - Script PHP vulnérable à l'attaque par traversée de chemin

```
<?php
$language="entete-en";
if (isset($_GET['lang'])) {
    $language=$_GET['lang'];
}
include ("/usr/local/webapp/template/" . $language . ".php")
?>
```

Si l'attaquant envoie la requête avec le paramètre « `lang=../../../../etc/passwd` », il aura accès au fichier des mots de passe du système et tentera de se connecter au serveur HTTP avec un des comptes ainsi trouvés.

Dans cet exemple l'attaque par traversée de chemin est possible à cause de la faille Référence directe non sécurisée à un objet (voir paragraphe III.E).

III-I-3 - Parade et bonnes pratiques

Pour se protéger des défaillances dans la restriction des accès à une URL, il ne faut pas autoriser les caractères douteux tels que « `/` » et « `\` ».

Pour se prémunir des attaques par traversée de chemin, il faut établir une liste de fichiers utilisables et refuser tout autre fichier. La correction à apporter à l'exemple ci-dessus est la suivante.

Figure 26 - Script PHP non vulnérable à l'attaque par traversée de chemin

```
<?php
$languages=array("entete-en","entete-fr","entete-es");
$language=$languages[1];
if (isset($_GET['lang'])) {
    $tmp=$_GET['lang'];
    if (array_search($tmp, $languages)) {
        $language=$tmp;
    }
}
include ("/usr/local/webapp/template/" . $language . ".php")
?>
```

Tous les répertoires doivent contenir un fichier `index.html`, ce qui évite de pouvoir accéder au répertoire lui-même. De plus, les serveurs HTTP doivent être configurés pour ne pas permettre l'affichage du contenu des répertoires.

Pour éviter les accès à des fonctionnalités sans autorisation, ces dernières doivent être protégées en vérifiant que l'utilisateur a le droit de les utiliser. Ne pas afficher de lien pour y accéder n'est pas une protection suffisante.

III-J - Protection insuffisante de la couche transport

III-J-1 - Principe

Comme évoqué pour le stockage des données sensibles, celles-ci ne doivent apparaître en clair qu'aux personnes autorisées. Sur Internet il existe un risque qu'une requête ou une réponse HTTP soit interceptée. Si elle contient des informations confidentielles transmises en clair, alors l'attaquant pourra les exploiter facilement.

Tous les réseaux de l'architecture de l'application Web sont concernés, depuis le navigateur de l'utilisateur jusqu'au stockage des données, en passant par le serveur Web.

III-J-2 - Exemples d'attaque

L'attaque du type « Homme du milieu » (ou « Man-in-the-Middle ») est une des attaques les plus répandues pour accéder aux données d'une application [7]. Si un attaquant réussit à compromettre un serveur proxy, il pourra intercepter toutes les communications. Si en plus ce serveur est responsable du chiffrement des flux HTTP, il aura accès aux données les plus sensibles qui devaient être chiffrées.

Si une partie de l'application seulement est protégée par chiffrement, alors l'application complète est vulnérable. Souvent, seule la page de connexion contenant le formulaire de saisie de l'identifiant et du mot de passe est chiffrée. Si l'utilisateur après s'être authentifié retourne sur des pages non chiffrées, alors des informations, telles que le nom de l'utilisateur ou l'identifiant de session, peuvent être transmises en clair de page en page, exposant ainsi toute l'application à des attaques d'usurpation d'identité (voir paragraphe III.D).

III-J-3 - Parade et bonnes pratiques

Si une application Web manipule des données sensibles il faut mettre en place du chiffrement SSL pour TOUTES les pages. De plus, les mots de passe et les identifiants de session ne doivent à aucun moment transiter en clair. Pour cela, il est possible de configurer le serveur Web pour rediriger automatiquement toutes les requêtes HTTP vers les pages chiffrées.

La protection de la couche « transport » vient en complément de la protection du stockage des données. Ainsi si les données stockées sont chiffrées, il faut s'assurer que tous les moyens de communication le soient aussi. Par exemple, la politique de sécurité pour les données médicales exige que le médecin du travail et le patient soient les seuls autorisés à consulter ces informations. Pour cela, au niveau de l'application, il faut s'assurer que l'utilisateur est soit la personne concernée, soit le praticien. De plus, les informations ne doivent apparaître en clair à aucun autre moment que pour l'affichage. Cela concerne les flux de communication entre l'utilisateur et les différents composants de l'architecture tels que le serveur HTTP ou la base de données, mais aussi les moyens de stockage tels que les fichiers, les bases de données ou les sauvegardes de ces derniers.

III-K - Redirection et renvois non validés

III-K-1 - Principe

Les redirections d'adresse sont utilisées dans les applications Web pour effectuer un changement de page en fonction d'un paramètre.

L'utilisation de la redirection est particulièrement utilisée pour les attaques par hameçonnage (ou phishing). En cliquant sur un lien utilisant une page de redirection, l'utilisateur est automatiquement emmené vers une autre page. Cette redirection peut être utilisée dans le cadre d'une attaque CSRF (voir paragraphe III.F).

III-K-2 - Exemples d'attaque

Le script suivant est un exemple de page de redirection en utilisant la redirection par code HTML.

Figure 27 - Script PHP de redirection automatique

```
<?php
$nouvelleAdresse='http://nouvelle.adresse.fr/index.php';
if (isset($_GET['adresse'])) {
    $nouvelleAdresse=$_GET['adresse'];
}
echo '<!DOCTYPE html>'. "\n",
'<html xmlns="http://www.w3.org/1999/xhtml">'. "\n",
'<head>'. "\n",
'<meta charset="UTF-8" />'. "\n",
//Redirection HTML : '<meta http-equiv="refresh" content="0; url='".$nouvelleAdresse.'" />'. "\n",
'<title>Redirection</title>'. "\n",
'<meta name="robots" content="noindex,follow" />'. "\n",
'</head>'. "\n",
"\n",
'<body>'. "\n",
//au cas où la redirection ne fonctionne pas :
'<p><a href="'.$nouvelleAdresse.'">Redirection</a></p>'. "\n",
'</body>'. "\n",
'</html>'. "\n";
?>
```

Si un attaquant a connaissance d'une page de redirection vulnérable, il peut l'utiliser dans un lien pour faire rediriger l'utilisateur vers une page Web.

Figure 28 - Lien pour rediriger l'utilisateur vers une page malveillante

```
<a href="http://www.application-securisee.com/redirect.php?www.attaquant.com/attaque.php">
http://www.application-securisee.com</a>
```

Dans ce cas l'utilisateur est leurré. Le lien pointe bien vers l'application, mais va le rediriger vers une page malveillante.

III-K-3 - Parade et bonnes pratiques

La redirection ne doit renvoyer qu'à des pages locales. Dans ce cas les caractères spéciaux doivent être prohibés.

En cas de changement d'adresse ou de déplacement d'une page, il vaut mieux utiliser la redirection paramétrée dans le serveur HTTP. Par exemple avec Apache il est possible de placer un fichier « .htaccess » pour paramétrer les redirections automatiques.

Figure 29 - Exemple de redirection automatique configurée au niveau du serveur HTTP

```
<IfModule mod_alias.c>
#redirection automatique d'une page vers une nouvelle adresse
Redirect permanent /dossier01/script_1.html http://nouvelle.adresse.fr/dossier03/script_1.php
#redirection automatique d'un ensemble de pages
RedirectMatch permanent /dossier01/(.*)\.html$ http://nouvelle.adresse.fr/dossier02/$1.php
#redirection automatique d'un dossier vers une nouvelle adresse
Redirect permanent /dossier01 http://nouvelle.adresse.fr/dossier02
#redirection automatique de toute l'application Web vers une nouvelle adresse
Redirect permanent / http://nouvelle.adresse.fr/
</IfModule>
```

IV - Bonnes Pratiques

IV-A - Règles de développement

Il est possible de se protéger de la plupart des attaques expliquées précédemment en suivant quelques règles de développement.

IV-A-1 - Toutes les données doivent être vérifiées

Les valeurs saisies dans un formulaire doivent être validées au niveau du navigateur avec du code JavaScript, car le client n'est pas une source fiable. Les valeurs doivent également être contrôlées au niveau du serveur au moment de la récupération des paramètres, tout comme les paramètres de requête. Il n'est pas certain que ce soit l'utilisateur de l'application qui envoie la requête HTTP. Ainsi pour chaque valeur :

- vérifier que le type correspond à celui attendu ;
- pour plus de sécurité, caster les données avant de les mettre dans des variables ;
- coder les caractères spéciaux avec le code HTML correspondant ;
- vérifier la présence de tous les arguments attendus ;
- pour les nombres, contraindre la valeur entre deux bornes ;
- pour les listes, vérifier que la valeur appartient à la liste des valeurs autorisées (select, radio, checkbox...) ;
- contraindre la longueur de la valeur saisie avec une taille minimale et une taille maximale ;
- vérifier la valeur avec une expression régulière ;
- n'accepter que les lettres de l'alphabet et/ou les chiffres par défaut, tous les autres caractères devant être refusés. Dans le cas où d'autres caractères doivent être autorisés, ils doivent être limités à une liste prédéfinie ou être remplacés par les codes HTML ;
- vérifier si la valeur nulle doit être acceptée ;
- définir le jeu de caractères de la donnée.

Même les données envoyées vers l'utilisateur doivent être vérifiées, avec au minimum les actions suivantes :

- coder les caractères spéciaux avec le code HTML correspondant ;
- définir le jeu de caractères de la page.

IV-A-2 - Privilégier l'utilisation des requêtes POST

Cela permet de ne pas prendre en compte des paramètres frauduleux dans les adresses.

IV-A-3 - Utiliser les requêtes paramétrées

Les requêtes SQL ne doivent pas être construites dynamiquement. Ainsi, les requêtes SQL ne peuvent pas être parasitées comme c'est le cas dans le cas de l'injection SQL (voir paragraphe III.B).

IV-A-4 - Ne pas réinventer la roue

Dans la mesure du possible, il est préférable d'utiliser des outils existants plutôt que de développer des fonctionnalités souvent présentes dans les applications, comme les systèmes d'authentification ou de réinitialisation de mot de passe.

IV-A-5 - Sécuriser l'accès aux données

Toutes les pages d'une application Web doivent s'assurer que l'utilisateur s'est authentifié préalablement. Pour les fonctionnalités manipulant des données sensibles, l'application doit demander à l'utilisateur de s'authentifier à nouveau avant de les afficher.

Cela permet de s'assurer qu'il n'y a pas eu usurpation d'identité. Les messages d'erreur renvoyés doivent être génériques pour ne pas aiguiller les attaquants potentiels.

Les données sensibles doivent être chiffrées dans les bases de données.

La protection des mots de passe, des identifiants de session et des cookies permet de se prémunir contre l'usurpation d'identité. Pour cela, le mot de passe doit avoir au moins huit caractères, voire dix pour les accès aux données sensibles. Il doit également contenir au moins trois types de caractères, tels que des chiffres, des lettres minuscules, des lettres majuscules ou des caractères spéciaux. Il doit avoir une période de validité de maximum quatre-vingt-dix jours. Au-delà, il doit être changé. Le compte de l'utilisateur doit être verrouillé, au moins temporairement, si cinq tentatives de connexion consécutives ont échoué. Le mot de passe doit être chiffré ou haché avec un algorithme fort. Seule sa valeur chiffrée sera comparée lors de l'authentification de l'utilisateur.

Les identifiants de session doivent avoir une durée de vie limitée au-delà de laquelle il n'est plus utilisable. De même, après une période d'inactivité prédéfinie, l'identifiant de session doit être invalidé. En cas de fermeture du navigateur, l'identifiant de session doit être supprimé. Ces actions sont équivalentes à un système de déconnexion automatique.

Les cookies doivent être protégés en positionnant deux attributs : « **Secure** » permet d'interdire l'envoi du cookie sur un canal non chiffré, « **HTTPOnly** » permet d'en interdire l'accès à JavaScript.

IV-B - Configuration des composants serveur

Pour des raisons de capacité de calcul, la taille minimale des clés symétriques utilisées jusqu'en 2020 doit être de 100 bits et de 128 bits au-delà de 2020.

Si le caractère confidentiel des données nécessite la mise en place du chiffrement des flux, toutes les pages doivent être protégées et pas seulement celles manipulant ces données.

Les outils installés doivent être mis à jour avec le correctif le plus récent.

Les options inutilisées des outils installés doivent être supprimées ou désactivées. Les comptes créés lors de l'installation des outils doivent être au minimum verrouillés ; le plus sûr étant de les supprimer.

Le serveur HTTP ne doit pas afficher le contenu d'un répertoire.

IV-C - Audit

Pendant la phase de développement, les tests unitaires permettent de vérifier le comportement d'une fonction de l'application. Ils peuvent être utilisés pour s'assurer que les règles de développement citées précédemment sont respectées. En complément, les outils d'intégration continue, tels que « Jenkins » ou « Hudson », permettent de vérifier à chaque modification de code qu'elle n'engendre pas de régression. L'utilisation conjointe de cette panoplie d'outils facilite les audits de code et permet même de les automatiser lors des phases de maintenance.

De plus, l'OWASP offre des outils de test du code et du comportement des applications Web. Ainsi, l'outil « Code Crawler » permet d'analyser le code d'applications .NET et Java. « WebScarab » agit comme un serveur proxy et permet à l'auditeur d'analyser les échanges HTTP pour chercher des failles de sécurité. « Zed Attack Proxy » est un scanner qui permet de détecter automatiquement certaines vulnérabilités. Quant au WASC, il ne fournit pas d'outil,

mais un guide pour comparer les différentes offres commerciales ou non d'automatisation de détection des problèmes liés à la sécurité des applications Web.

Par ailleurs, en condition opérationnelle, les fichiers de journalisation des différents composants de l'architecture doivent faire l'objet d'un audit régulier afin de s'assurer que l'application ou le serveur n'a pas été victime de tentative d'attaque par la force brute.

V - Conclusion

V-A - Constat

Depuis la version de 2004, le classement de l'OWASP a peu évolué. En effet, sept problèmes répertoriés en 2010 étaient déjà présents en 2004 comme le montre le tableau ci-dessous.

Risques	2004	2007	2010
Injection de commandes	A6	A2	A1
Faillies Cross Site Scripting (XSS)	A4	A1	A2
Violation de Gestion d'Authentification et de Session	A3	A7	A3
Référence directe à un objet non sécurisée	A1	A4	A4
Cross Site Request Forgery (CSRF)		A5	A5
Gestion de configuration non sécurisée	A10		A6
Stockage non sécurisé	A8	A8	A7
Manque de restriction d'accès URL	A2	A10	A8
Violation de Contrôle d'Accès			
Communications non sécurisées		A9	A9
Redirection et renvoi non validés			A10
Débordement de tampon	A5		
Mauvaise gestion des erreurs	A7	A6	
Déni de Service	A9		
Exécution de fichier malicieux		A3	

Cela signifie que le Web 2.0 a apporté du confort à l'utilisateur mais n'a pas apporté des failles plus dangereuses que celles déjà présentes. En effet, AJAX permet d'étendre les possibilités des attaques. Ainsi, le vol d'informations par CSRF prend une nouvelle forme. Tout comme l'attaque CSRF classique expliquée dans le paragraphe 3.6, un utilisateur ouvre une page Web frauduleuse. Ensuite l'attaquant profite du mode asynchrone de la fonction « [XMLHttpRequest](#) » pour parcourir la page de l'application affichée, voler le cookie et envoyer les données à

l'attaquant. Les protections contre CSRF restent efficaces, mais se protéger de la seconde phase de l'attaque est plus délicat. Par ailleurs AJAX permet d'outrepasser la protection interdomaines offerte par les navigateurs.

Bien que de nouvelles failles émergent avec le Web 2.0, les applications Web sont toujours vulnérables aux failles les plus anciennes. Cela démontre que le comportement des développeurs n'a pas changé. Pressés par des contraintes de temps, ils ne font pas l'effort d'appliquer quelques règles de base qui permettent de se défendre contre les attaques les plus dangereuses.

V-B - Perspectives

2012 va marquer l'arrivée de nouveaux standards : HTML 5 et CSS 3. Leur objectif est de combler les lacunes de leurs prédécesseurs utilisés depuis bientôt quinze ans. Les développeurs espèrent qu'avec ces nouvelles versions, l'utilisation de JavaScript diminue, allégeant ainsi le code des applications Web et diminuant par la même occasion le nombre des vulnérabilités. Le coût de mise à jour des applications Web actuelles risque de freiner l'adoption de ces nouvelles normes. Les organismes de sécurité devront également s'impliquer pour mettre à disposition des développeurs de nouvelles bonnes pratiques.

VI - Bibliographie







- [1] L. Shklar et R. Rosen. *Web Application Architecture: Principles, Protocols and Practices*. John Wiley & Sons Ltd, 372p, 2003
- [2] S. Gulati. Under the hood of the Internet: Connector: the Internet protocol, part one: the foundations. *Magazine Crossroads* 6, article 3, 2000
- [3] G. Florin et S. Natkin. *Support de cours RESEAUX ET PROTOCOLES*. CNAM, 884p, 2007
- [4] J. Governor, D. Hinchcliffe et D. Nickull. *Web 2.0 Architectures*. O'Reilly Media, 248p, 2009
- [5] C. Porteneuve et T. Nitot. *Bien développer pour le Web 2.0 : Bonnes pratiques Ajax*. Eyrolles, 673p, 2008
- [6] M. Contensin. Sécurité des applications Web dans *formation PHP/MySQL* chapitre 6. CNRS, 2007
- [7] J. Scambray, V. Liu et C. Sima. *Hacking Exposed Web Applications: Web Application Security Secrets and Solutions*. Osborne/McGraw-Hill, 482p, 2010
- [8] H. Dwivedi, A. Stamos, Z. Lackey et R. Cannings. *Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions*. Osborne/McGraw-Hill, 258p, 2007
- [9] Z. Su et G. Wassermann. The essence of command injection attacks in Web applications. Dans *POPL'06 Conference*, ACM SIGPLAN Notices, p372-384, 2006


[10] Y.-W. Huang, C.-H. Tsai, T.-P. Lin, S.-K. H., D.T. Lee et S.-Y. Kuo. A testing framework for Web application security assessment. Dans *Computer Networks*, pages 739-761, 2005

[11] A. Kiezun, P. J. Guo, K. Jayaraman, M. D. Ernst. Automatic Creation of SQL Injection and Cross-Site Scripting Attacks. Dans *ICSE*, pages 199-209, 2009

[12] D. Gollmann. Securing Web applications. Dans *Information Security Technical Report*, chapitre 1-9, Elsevier, 2008

Les sites Web cités ci-dessous sont considérés comme des sources fiables par les organismes évoqués précédemment.

-  <http://csrc.nist.gov/publications/> consulté le 27/11/2011.
-  <http://erratasec.blogspot.com/> consulté le 26/01/2012.
-  <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/> consulté le 27/11/2011.
-  <http://ha.ckers.org/> consulté le 27/11/2011.
-  <http://googlewebmastercentral.blogspot.com/2009/01/open-redirect-urls-is-your-site-being.html> consulté le 27/11/2011.
-  <http://jeremiahgrossman.blogspot.com/2010/01/wasc-threat-classification-to-owasp-top.html> consulté le 26/01/2012.
-  <http://references.modernisation.gouv.fr/> consulté le 21/11/2011.
-  <http://www.certa.ssi.gouv.fr/> consulté le 21/11/2011.
-  <http://www.cgisecurity.com/> consulté le 27/11/2011.
-  <http://www.checkmarx.com/> consulté le 27/11/2011.
-  <http://www.clusif.asso.fr/> consulté le 21/11/2011.
-  <http://www.cnil.fr/> consulté le 03/12/2011.
-  <http://www.dgdr.cnrs.fr/fsd/> consulté le 19/11/2011.
-  <http://www.dwheeler.com/secure-programs/> consulté le 27/11/2011.
-  <http://www.ietf.org/> consulté le 27/11/2011.
-  <http://www.legifrance.gouv.fr/> consulté le 10/12/2011.
-  <http://www.ouah.org/chambet.html> consulté le 27/11/2011.
-  <https://www.owasp.org/> consulté le 19/11/2011.
-  <http://www.pcmag.com/article2/0,4149,11525,00.asp> consulté le 10/12/2011.
-  <http://www.references.modernisation.gouv.fr/rgs-securite> consulté le 27/11/2011.
-  <http://www.resinfo.org/> consulté le 20/11/2011.
-  <http://www.rsa.com/> consulté le 10/12/2011.
-  http://www.schneier.com/blog/archives/2005/10/scandinavian_at_1.html consulté le 27/11/2011.
-  <http://www.ssi.gouv.fr/> consulté le 21/11/2011.
-  <http://www.w3.org/> consulté le 21/11/2011.
-  <http://www.webappsec.org/> consulté le 21/11/2011.
-  <http://yehg.net/lab/> consulté le 27/11/2011.

Source des icônes libres de droits d'utilisation :  <http://findicons.com>

VII - Remerciements

Je tiens à remercier **ClaudeLELOUP** et **Erielle** pour la relecture orthographique attentive de cet article.