OWASP TOP 10

LES DIX VULNERABILITES DE SECURITE APPLICATIVES WEB LES PLUS CRITIQUES

EDITION 2007

© 2002-2007 Fondation OWASP

Ce document est référencé sous license Creative Commons Attribution-ShareAlike 2.5

TABLE DES MATIERES

Table des matières 2

Introduction 3

Résumé 5

Méthodologie 7

A1 – Cross Site Scripting (XSS) 10

A2 – Faille d'injection 13

A3 – Exécution de fichier malicieux 16

A4 – Référence directe non sécurisée à un objet 20

A5 – Falsification de requête inter-site (Cross Site Request Forgery - CSRF) 22

A6 – Fuite d'information et traitement d'erreur incorrect 25

A7 – Violation de gestion d'authentification et de session 27

A8 – Stockage cryptographique non sécurisé 29

A9 – Communications non sécurisées 31

A10 – Manque de restriction d'accès URL 33

Que faire maintenant 35

Références 38

INTRODUCTION

Bienvenue dans le Top 10 2007 de l'OWASP! Cette édition totalement réécrite liste les vulnérabilités d'application web les plus sérieuses, indique comment s'en protéger, et fournit des liens de référence pour plus d'information.

BUT

Le Top 10 de l'OWASP a pour but premier d'éduquer les développeurs, concepteurs, architectes et entreprises sur les conséquences des vulnérabilités de sécurité les plus communes des applications web. Le Top 10 fournit des méthodes de base pour se protéger contre ces vulnérabilités - première étape indispensable à votre programme sécurité de programmation sécurisée.

La sécurité n'est pas un évènement ponctuel. Sécuriser votre code juste une fois ne suffit pas. En 2008, ce Top 10 aura changé, et sans changer une ligne du code de votre application, vous pourrez être vulnérables. Le lecteur est invité à relire le conseil dans Que faire maintenant pour plus d'informations.

Une démarche de programmation sécurisée doit composer avec toutes les étapes du cycle de vie d'un programme. Les applications web sécurisées sont **seulement** possibles quand un SDLC (i.e. Software Development Life Cycle) sécurisé est utilisé. Les programmes sûrs sont sécurisés par conception, pendant le développement et par défaut. Il y a au moins 300 problèmes affectant l'ensemble de la sécurité d'une application web. Ces 300 problèmes sont détaillés dans le Guide de l'OWASP, lecture essentielle et indispensable à toute personne développant des applications web aujourd'hui.

Ce document est d'abord et avant tout un recueil éducatif, pas une norme. Vous êtes invités à ne pas adopter ce document comme politique ou norme sans nous en parler d'abord! Si vous avez besoin d'une politique de programmation sécurisée ou d'un standard, l'OWASP dispose de politiques de programmation sécurisée et des projets de normes en cours d'élaboration. Nous vous invitons à vous joindre à nous ou à nous assister financièrement pour ces efforts.

REMERCIEMENTS

Nous remercions MITRE pour la mise à disposition gratuite pour usage de « Vulnerability Type Distribution au format CVE ». Le projet OWASP Top Ten est piloté et sponsorisé par Aspect Security.

Chef de Projet: Andrew van der Stock (Directeur Exécutif, Fondation OWASP)

Co-auteurs: Jeff Williams (Président, Fondation OWASP), Dave Wichers (Maître de Conférence, Fondation OWASP)

Nous voudrions remercier nos relecteurs:

- Raoul Endres pour son aide à faire évoluer le Top 10 et pour ses précieux commentaires
- Steve Christey (MITRE) pour sa relecture minutieuse de chaque point et l'ajout des données MITRE CWE
- Jeremiah Grossman (White Hat Security) pour sa relecture et sa contribution au succès des moyens automatisés de détection
- Sylvan von Stuppe pour une relecture exemplaire
- Colin Wong, Nigel Evans, Andre Gironda, Neil Smithline pour leurs commentaires par e-mail

RESUMÉ

A1 - Cross Site Scripting (XSS)	Les failles XSS se produisent à chaque fois qu'une application prend des données écrites par l'utilisateur et les envoie à un browser web sans avoir au préalable validé ou codé ce contenu. 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 - Faille d'Injection	Les failles d'injection, en particulier l'injection SQL, sont communes dans les applications web. 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 - Exécution de Fichier Malicieux	Un code vulnérable à l'inclusion de fichier à distance *(RFI - Remote File Inclusion)* permet à des attaquants d'inclure du code et des données hostiles, ayant pour résultat des attaques dévastatrices, telle la compromission totale d'un serveur. Les attaques par exécution de fichier malveillant affectent PHP, XML et toute structure qui accepte des noms de fichiers ou des fichiers des utilisateurs.
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.
A5 - Falsification de requête inter-site (CSRF)	Une attaque CSRF *(Cross Site Request Forgery)* 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. CSRF peut être aussi puissant que l'application web qu'il attaque.
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.
A7 - Violation de Gestion d'Authentification et de Session	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.
A9 - Communications non Sécurisées	La plupart du temps, les applications ne chiffrent pas le trafic réseau quand il est nécessaire de protéger des communications sensibles.
A10 - Manque de Restriction d'Accès 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.

Table 1: Top 10 2007 des vulnérabilités des applications Web

METHODOLOGIE

Notre méthodologie pour le Top 10 2007 a été simple: prendre les Tendances de Vulnérabilité de MITRE pour 2006, et distiller le Top 10 *des problèmes de sécurité applicatifs web*. Les résultats sont classés comme suit :

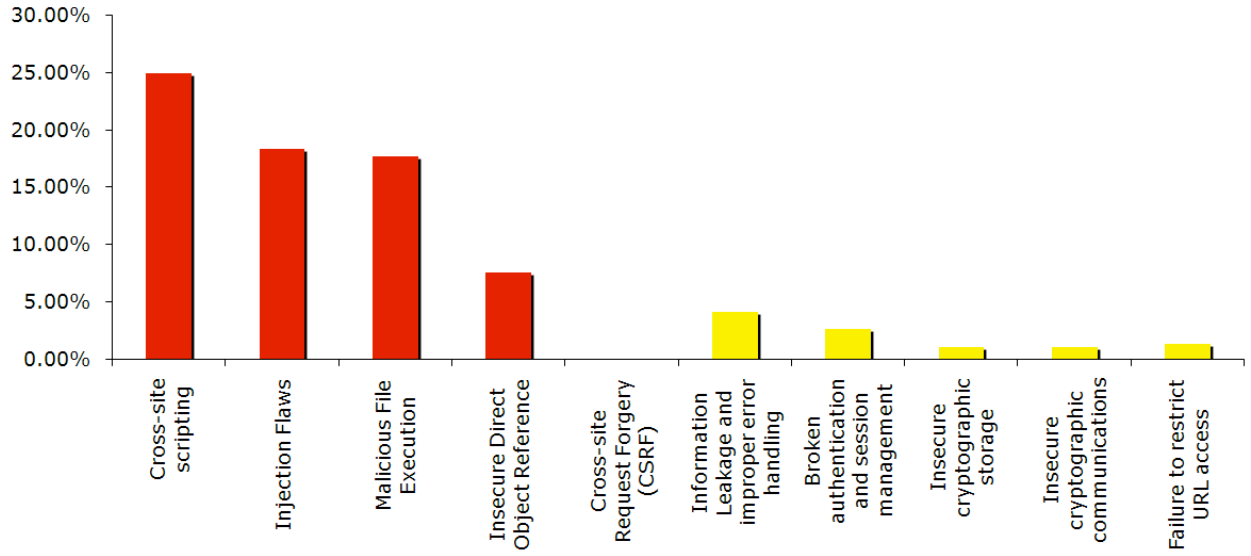

Figure 2: MITRE data on Top 10 web application vulnerabilities for 2006

Bien que nous ayons essayé de préserver une adéquation entre les données de vulnérabilité de MITRE et nos titres de section, nous avons délibérément changé certaines des catégories ultérieures pour refléter plus étroitement les causes premières. Si vous êtes intéressé par les données finales 2006 du MITRE, nous avons inclus une feuille Excel sur le site web du Top 10 de l'OWASP.

Toutes les recommandations de protection fournissent des solutions pour les trois contextes applicatifs web les plus répandus : Java EE, ASP.NET, et PHP. D'autres contextes communs d'application web, tels que Ruby on Rails ou Perl peuvent facilement adapter les recommandations pour répondre à leurs besoins spécifiques.

POURQUOI NOUS AVONS SUPPRIME QUELQUES PROBLEMES IMPORTANTS

La validation de paramètre en « entrée » est un défi important pour toute équipe de développement, et est à l'origine de beaucoup de problèmes de sécurité applicatifs. En fait, plusieurs des autres sujets dans la liste recommandent la validation d'une « entrée » comme faisant partie de la solution. Nous recommandons toujours vivement de créer un mécanisme centralisé de validation d'entrée comme partie intégrante de vos applications web. Pour plus d'information, lire les articles OWASP suivants relatifs à la validation de données :

- http://www.owasp.org/index.php/Data_Validation
- http://www.owasp.org/index.php/Testing_for_Data_Validation

Les débordements de buffers (ou « *débordement de tampon dans la pile* » si nous voulons être précis), **les débordements d'entier** et **les problèmes de format de chaîne** sont des vulnérabilités extrêmement sérieuses pour des programmes écrits en langages tels que C ou C++. La solution à ces problèmes est couverte par la communauté traditionnelle de sécurité applicative non-web, comme le SANS et le CERT, et les fournisseurs d'outils de langages de programmation. Si votre code est écrit dans un langage susceptible de subir des débordements de buffers, nous vous encourageons à lire le contenu sur le débordement de buffer sur le site de l'OWASP:

- http://www.owasp.org/index.php/Buffer_overflow
- http://www.owasp.org/index.php/Testing_for_Buffer_Overflow

Le déni de service (DoS - Denial of Service) est une attaque sérieuse qui peut affecter n'importe quel site quel que soit le langage dans lequel il a été écrit. Le classement de DoS par le MITRE est insuffisant pour figurer dans le Top 10 cette année. Si vous avez des soucis concernant le déni de service, vous devriez consulter le site de l'OWASP et le Guide de Tests:

- http://www.owasp.org/index.php/Category:Denial_of_Service_Attack
- http://www.owasp.org/index.php/Testing_for_Denial_of_Service

La gestion de configuration non sécurisée affecte dans une certaine mesure tous les systèmes, en particulier PHP. Toutefois, le classement par MITRE ne nous permet pas d'inclure ce problème cette année. Quand vous déploierez votre application, vous devriez consulter les versions les plus récentes du Guide de l'OWASP et du Guide de Tests de l'OWASP pour une information détaillée concernant la gestion et le test d'une configuration sécurisée

- http://www.owasp.org/index.php/Configuration
- http://www.owasp.org/index.php/Testing_for_infrastructure_configuration_management

POURQUOI NOUS AVONS AJOUTE QUELQUES PROBLEMES IMPORTANTS

Cross Site Request Forgery (CSRF) - ou *Falsification de requête inter-site* - est la principale nouveauté de cette édition du Top 10 de l'OWASP. Bien que les données brutes le classent en 36e position, nous pensons qu'il est important que les applications commencent leurs efforts de protection dès aujourd'hui, en particulier pour les applications à haute valeurs ajoutée ainsi que celles qui traitent des données sensibles. CSRF est plus répandu que son classement actuel ne le laisserait supposer, et il peut être très dangereux.

Cryptographie Les utilisations peu sécurisées de la cryptographie ne sont pas les problèmes de sécurité applicatifs web #8 et #9 selon les données brutes de MITRE, mais elles sont à l'origine de beaucoup de fuites relatives à la vie privée et de problème de conformité (en particulier avec la conformité PCI DSS 1.1).

VULNERABILITES, PAS ATTAQUES

L'édition précédente du Top 10 contenait un mélange d'attaques, de vulnérabilités et de contre-mesures. Cette fois-ci, nous nous sommes essentiellement concentrés sur les vulnérabilités,

bien que la terminologie généralement utilisée combine parfois vulnérabilités et attaques. Si les entreprises emploient ce document pour sécuriser leurs applications et réduisent les risques à leurs business, ceci mènera inéluctablement à une réduction directe de la probabilité des

- **Attaques par Phishing** qui peuvent exploiter l'un de ces failles, en particulier XSS, et affaiblir ou rendre inexistant les contrôles d'authentification ou d'autorisation (A1, A4, A7, A10)
- **Violations de vie privée** dûes à des contrôles faibles de validation, de principe économique et d'autorisation (A2, A4, A6, A7, A10)
- **Vols d'identité** à travers des contrôles cryptographiques insuffisants ou inexistants (A8 et A9), inclusion de fichier (A3) et authentification à distance, principe économique, et contrôles d'autorisation (A4, A7, A10)
- **Compromission de systèmes, altération de données, ou attaques de destruction de donnée** par Injections (A2) and inclusion de fichier à distance (A3)
- **Perte financière** à travers des transactions non autorisées et attaques CSRF (A4, A5, A7, A10)
- **Perte de réputation** par exploitation de l'une des vulnérabilités ci-dessus (A1 ... A10)

Chaque fois qu'une entreprise passera d'un mode de contrôles réactifs, à la réduction proactive des risques applicables à son business, cela améliorera sa conformité aux régimes réglementaires, réduira les dépenses opérationnelles, et si tout va bien permettra des systèmes bien plus robustes et plus sécurisés en conséquence.

INFLUENCES

La méthodologie décrite ci-dessus polarise nécessairement le Top 10 vers des découvertes par la communauté des chercheurs en sécurité Ce modèle de découverte est semblable aux méthodes d' attaques actuelles, en particulier car il se rapporte aux attaquants d'entrée de gamme (« script kiddy »). Protéger votre logiciel contre le Top 10 va assurer un minimum de protection contre les formes d'attaques les plus courantes, mais plus important que tout, va aider à l'amélioration de la sécurité de vos logiciels.

CARTOGRAPHIE

Il y a eu des changements de titres, même où le contenu cadre étroitement au contenu précédent. Nous n'employons plus le schéma de nommage WAS XML car il n'a pas été tenu à jour avec les vulnérabilités, les attaques, et contre-mesures modernes. La table ci-dessous dépeint l'adéquation de cette édition avec le Top 10 2004 et le classement de MITRE:

OWASP Top 10 2007	OWASP Top 10 2004	MITRE 2006 Raw Ranking
A1. Cross Site Scripting (XSS)	A4. Cross Site Scripting (XSS)	1
A2. Faille d'Injection	A6. Faille d'Injection	2
A3. Exécution de fichier malicieux (NEW)		3
A4. Référence directe non sécurisée à un objet	A2. Violation de Contrôle d'Accès (partagé dans le Top 10 2007)	5
A5. Cross Site Request Forgery (CSRF) (NEW)		36
A6. Fuite d'information et traitement d'erreur incorrect	A7. Traitement d'Erreur Incorrect	6
A7. Violation de gestion d'authentification et de session	A3. Violation de gestion d'authentification et de session	14
A8. Stockage cryptographique non sécurisé	A8. Stockage non sécurisé	8
A9. Communications non sécurisées (NEW)	Discuté dans A10. Gestion de Configuration non sécurisé	8
A10. Manque de restriction d'accès URL	A2. Violation de Contrôle d'Accès (partagé dans le Top 10 2007)	14
<enlevé in 2007>	A1. Paramètre non validé	7
< enlevé in 2007>	A5. Débordement de tampon	4, 8, and 10
< enlevé in 2007>	A9. Deni de Service	17
< enlevé in 2007>	A10. Gestion de configuration non sécurisé	29

A1 – CROSS SITE SCRIPTING (XSS)

Cross Site Scripting, plus connu sous le nom de XSS, est en fait un sous-ensemble de l'injection HTML. XSS est le problème de sécurité applicatif web le plus répandu et le plus pernicieux. Les failles XSS se produisent à chaque fois qu'une application prend des données écrites par l'utilisateur et les envoie à un browser web sans en avoir au préalable validé ou codé le contenu.

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, insérer du contenu hostile, effectuer des attaques par phishing, et prendre le contrôle du navigateur de l'utilisateur en utilisant un script malicieux *(Malware Scripting)*. Le script malicieux est habituellement écrit en Javascript, mais n'importe quel langage de programmation supporté par le navigateur de la victime est une cible potentielle pour cette attaque.

ENVIRONNEMENTS AFFECTÉS

Tous les contextes applicatifs web sont vulnérables au Cross Site Scripting.

VULNÉRABILITÉ

Il existe trois types de cross site scripting connus: par réflexion, par stockage, et par injection DOM. L' attaque XSS par réflexion est la plus facile à exploiter - une page renverra à l'utilisateur des données fournies directement à celui-ci:

```
Echo $_REQUEST ['userinput'];
```

L'attaque XSS par stockage prend des données hostiles, les stocke dans un fichier, une base de données ou tout autre système de back end, et ultérieurement, affiche les données à l'utilisateur, non filtrées. Ceci est extrêmement dangereux dans les systèmes tels que CMS, les blogs ou les forums, où un grand nombre d'utilisateurs seront à même de voir les inputs d'autres individus.

Avec les attaques XSS basées sur DOM, le code JavaScript et les variables du site sont manipulés plutôt que les éléments HTML.

Alternativement, les attaques peuvent être ou un mélange ou un hybride de ces trois types. Le danger avec cross site scripting n'est pas tant le type d'attaque, mais plutôt qu'il soit possible. Les comportements non standards ou inattendus du navigateur peuvent présenter des vecteurs d'attaque subtils. XSS est aussi potentiellement accessible à travers tous les composants qu'utilise le navigateur.

Les attaques sont habituellement effectuées en JavaScript, qui est un puissant langage de programmation. L'utilisation de JavaScript permet aux attaquants de manipuler n'importe quel aspect de la page rendue, y compris l'ajout de nouveaux éléments (tel qu'ajouter une demande de login qui forwarde les droits à un site hostile), en manipulant n'importe quel aspect de l'arborescence interne DOM, et supprimant ou changeant la présentation de la page. JavaScript

permet l'utilisation de « XmlHttpRequest », qui est typiquement employé par des sites utilisant les technologies AJAX, même si le site de la victime n'emploie pas AJAX aujourd'hui.

En utilisant XmlHttpRequest, il est parfois possible de contourner la même politique source d'un navigateur - transmettant ainsi les données de la victime vers des sites hostiles, et de créer des vers complexes et des zombies malveillants qui perdurent aussi longtemps que le navigateur reste ouvert. Les attaques AJAX n'ont pas besoin d'être visibles ou de nécessiter une interaction de l'utilisateur pour exécuter des attaques dangereuses de type Cross Site Request Forgery (CSRF) (voir A - 5).

VERIFICATION DE LA SECURITE

Le but est de vérifier que tous les paramètres dans l'application sont validés et/ou encodés avant d'être inclus les pages HTML.

Approches automatisées : Les outils de tests de pénétration automatisés sont capables de détecter l'XSS par réflexion par injection de paramètre, mais échouent souvent à trouver un XSS persistant, particulièrement si le résultat du vecteur XSS injecté est empêché via des contrôles d'autorisation (tel par exemple un utilisateur qui saisit des données hostiles que seuls les administrateurs peuvent parfois voir ultérieurement). Les outils automatisés de balayage de code source peuvent trouver des APIs faibles ou dangereuses mais ne peuvent habituellement pas déterminer si la validation ou le codage a eu lieu, ce qui peut avoir comme conséquence des faux positifs. Aucun type d'outil n'est capable de trouver XSS basé sur DOM, ce qui signifie que les applications basées sur Ajax seront habituellement en danger seulement si un test automatisé a lieu.

Approches manuelles: Si un mécanisme centralisé de validation et de codage est employé, la manière la plus efficace de vérifier la sécurité est d'auditer le code. Si une mise en œuvre distribuée est utilisée, la vérification sera alors considérablement plus longue. Les tests prennent un temps considérable parce que la surface d'attaque de la plupart des applications est relativement importante.

PROTECTION

La meilleure protection contre XSS est une combinaison de validation de type « whitelist » de toutes les données entrantes et du codage approprié de toutes les données produites en sortie. La validation permet la détection d'attaques, et l'encodage empêche toute injection de script de se produire dans le navigateur.

La protection d'une application complète contre XSS exige une approche architecturale cohérente:

- **Validation d' entrée.** Utiliser un mécanisme standard de validation d'entrée pour valider la longueur, le type et la syntaxe de toute entrée saisie, ainsi que les règles de gestion avant d'accepter l'affichage ou le stockage des données. Utiliser une stratégie de validation *« accept known good »*, à savoir essentiellement basée sur l'acceptation de

quelque chose de connu. Rejeter toute entrée invalide plutôt qu'essayer d'assainir des données potentiellement hostiles. N'oubliez pas que les messages d'erreur peuvent également inclure des données invalides

- **Chiffrement robuste des données en sortie.** Assurez-vous que toutes les données écrites par l'utilisateur sont convenablement codées par entité (soit HTML ou XML selon le mécanisme de présentation) avant le rendu, prenant le parti de coder tous les caractères plutôt qu'un sous-ensemble très limité. C'est l'approche de la bibliothèque Anti-XSS de Microsoft, et la prochaine bibliothèque Anti-XSS PHP de l'OWASP. En outre, le codage de caractère pour chaque page produite réduira l'exposition à quelques variantes
- **Spécifier le codage en sortie** (tel que ISO 8859-1 ou UTF 8). Ne laissez pas à l'attaquant l'opportunité de le choisir pour vos utilisateurs
- **N'utilisez pas de validation basée sur « blacklist / blacklistage »** pour détecter XSS en entrée ou pour coder le rendu. La recherche et le remplacement de juste quelques caractères ("<" ">" et d'autres caractères ou expressions semblables telles que "script") est faible et a été attaquée avec succès. Même un tag non vérifié "<b>" est dangereux dans certains contextes. XSS a un nombre surprenant de variantes qui rendent facile le contournement de toute validation basée sur « blacklist / blacklistage »
- **Faites attention aux erreurs canoniques.** Les entrées doivent être décodées et canonisées (i.e. leur format doit être contrôlé) à la représentation interne courante de l'application avant d'être validée. Assurez-vous que votre application ne décode pas deux fois la même entrée. De telles erreurs pourraient être employées pour contourner les schémas de "whitelist" en présentant des entrées dangereuses après qu'elles aient été vérifiées

Recommandations spécifiques de langages:

- **Java: Utilisez des contre-mécanismes en sortie** tels que <bean:write ... >, ou utilisez l'attribut par défaut JSTL escapeXML="true" dans <c:out ... >. N'utilisez PAS <%= ... %> non imbriqué (c'est-à-dire, en dehors d'un mécanisme de rendu correctement codé)
- **.NET: Utilisez la bibliothèque Microsoft Anti-XSS** 1.5 disponible gratuitement sur MSDN. N'assignez pas de formulaires de champs de données directement de l'objet de Requête (the Request Object): `username.Text = Request.QueryString ("username");` sans utiliser cette bibliothèque. Sachez comprendre quels contrôles .NET codent automatiquement les données de sortie
- **PHP: S'assurer que le rendu soit passé par des htmlentities () ou htmlspecialchars ()** ou utiliser la très prochaine bibliothèque PHP Anti-XSS de l'OWASP. Désactiver register_globals, si cela n'est pas déjà fait

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4206
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3966
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5204

REFERENCES

- CWE: CWE-79, Cross-Site scripting (XSS)
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml
- OWASP – Cross site scripting, http://www.owasp.org/index.php/Cross_Site_Scripting
- OWASP – Testing for XSS, http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- OWASP Stinger Project (A Java EE validation filter) – http://www.owasp.org/index.php/Category:OWASP_Stinger_Project
- OWASP PHP Filter Project - http://www.owasp.org/index.php/OWASP_PHP_Filters
- OWASP Encoding Project - http://www.owasp.org/index.php/Category:OWASP_Encoding_Project
- RSnake, XSS Cheat Sheet, http://ha.ckers.org/xss.html
- Klein, A., DOM Based Cross Site Scripting, http://www.webappsec.org/projects/articles/071105.shtml
- .NET Anti-XSS Library - http://www.microsoft.com/downloads/details.aspx?FamilyID=efb9c819-53ff-4f82-bfaf-e11625130c25&DisplayLang=en

A2 – FAILLES D'INJECTION

Les failles d'injection, en particulier l'injection SQL, sont courantes dans les applications web. Il existe différents types d'injection de données : SQL, LDAP, XPath, XSLT, HTML, XML, commandes systèmes, et beaucoup d'autres.

L'injection se produit lorsqu'une donnée fournie par l'utilisateur est envoyée à un interpréteur dans le cadre d'une commande ou d'une requête.

Les attaquants dupent l'interpréteur en lui faisant exécuter des commandes fortuites via la soumission de données spécialement formées. Les failles d'injection, permettent aux attaquants de créer, lire, modifier ou effacer toute donnée arbitraire de l'application. Dans le pire des scénarii, ces failles permettent à un attaquant de compromettre complètement l'application et le système sous-jacent même en contournant les environnements de filtrage sérieusement structurés.

ENVIRONNEMENTS AFFECTES

Toutes les applications web utilisant des interpréteurs ou invoquant d'autres processus sont vulnérables aux attaques par injection. Cela inclus tout composant d'un framework qui pourrait utiliser des interpréteurs secondaires.

VULNERABILITE

Si la donnée provenant d'un utilisateur est envoyée à un interpréteur sans aucune validation ou encodage, l'application est vulnérable. Vérifier si la donnée utilisateur est fournie à des requêtes dynamiques, telles que:

```
PHP:

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

Java:

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

VERIFICATION DE LA SECURITE

Le but est de vérifier que les données fournies par l'utilisateur ne peuvent modifier le sens des commandes ou requêtes exécutées par les interpréteurs invoqués par l'application.

Approches automatisées : beaucoup d'outils d'audit de vulnérabilités recherchent les problèmes d'injection, particulièrement l'injection SQL. Les outils d'analyses recherchant l'utilisation inappropriée des API d'interpréteurs sont pratiques, mais ne peuvent pas, la plupart du temps, vérifier que l'encodage et la validation appropriée des paramètres sont mis en place pour se protéger contre cette vulnérabilité. Si l'application intercepte les codes d'erreurs internes au serveur (code 501 et 500), ou les erreurs des bases de données, cela peut gêner les

outils automatisés, mais le code peut quand même comporter un risque. Ces outils automatisés sont aussi capables de détecter des failles d'injections de type LDAP/XML/Xpath.

Approches manuelles : l'approche la plus appropriée est de vérifier le code invoquant des interpréteurs. Le vérificateur devrait vérifier l'utilisation d'une API sûr ou qu'une validation et/ou un codage approprié s'est produit. Les tests peuvent prendre beaucoup de temps, sans pour autant valider la globalité de l'application car le spectre peut être trop large.

PROTECTION

Bannissez l'utilisation des interpréteurs le plus souvent possible. Si vous devez invoquer un interpréteur, la meilleure méthode pour éliminer les failles d'injections est l'utilisation d'API sûrs, telles les requêtes fortement typées et les librairies d'objets de cartographie relationnels (ORM). Ces interfaces gèrent toute évasion de donnée, ou ne requièrent pas d'échappements. Notez que bien que les interfaces sûres résolvent le problème, la validation est toutefois recommandée afin de détecter les attaques.

Utiliser des interpréteurs est dangereux, ainsi vaut-il mieux prendre des précautions supplémentaires, telles les suivantes:

- **Validation des données d'entrée :** utilisez un mécanisme standard de validation des entrées pour valider toutes les données d'entrée pour la longueur, le type, la syntaxe et les règles métiers avant d'accepter l'affichage ou le stockage de donnée. Utilisez une stratégie d' « acceptation des bonnes valeurs ». Rejetez toute entrée non valide plutôt qu'essayer d'assainir les données potentiellement hostiles.. N'oubliez pas que les messages d'erreurs peuvent, eux aussi, contenir des données invalides

- **Utilisez des APIs de requêtes fortement typées** avec des marqueurs de substitution dédiés, même lors d'appels de procédures stockées

- **Respectez le principe du moindre privilège** lorsque vous vous connectez à des bases de données ou tout autre système de back-office

- **Evitez les messages d'erreurs détaillées**, utiles aux attaquants

- **Utilisez des procédures SQL stockées** car elles généralement non vulnérables à l'injection SQL. Faites toutefois attention qu'elle ne le devienne (via l'utilisation de fonctions de type exec() ou par la concaténation d'arguments dans la procédure stockée)

- **N'utilisez pas d'interfaces de requêtes dynamiques** (tel que mysql_query() ou similaire)

- **N'utilisez pas les fonctions d'échappements simples,** comme la fonction PHP addslashes() ou les fonctions de remplacement de caractères comme str_replace("'", "''"). Elles sont faillibles et ont déjà été utilisées avec succès par des attaquants. Pour

PHP, utilisez mysql_real_escape_string() si vous utilisez MySQL, ou utilisez plutôt PDO qui ne nécessite pas d'échappement

- Lorsque vous utilisez des mécanismes d'échappement simple, sachez que **les fonctions d'échappement simple ne peuvent échapper les noms des tables !** Les noms des tables doivent respecter le format SQL, et sont donc complètement inadaptés en tant que données d'entrée utilisateur

- **Attention aux erreurs canoniques.** Les données en entrée doivent être décodées et normalisées à la représentation interne courante de l'application avant d'être validée. Assurez-vous que votre application ne décode pas la même entrée deux fois. De telles erreurs pourraient être utilisées pour contourner les schémas de *whitelist* en présentant des entrées dangereuses après qu'elles aient été vérifiées

Recommandations spécifiques aux langages de programmation

- Java EE – utilisez la fonction fortement typée PreparedStatement, ou des ORM comme Hibernate ou Spring
- .NET – utilisez des requêtes fortements typées, telles SqlCommand avec SqlParameter ou un ORM comme Hibernate
- PHP – utilisez PDO avec des requêtes fortements typées (en utilisant bindParam())

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5121
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4953
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4592

RÉFÉRENCES

- CWE: CWE-89 (SQL Injection), CWE-77 (Command Injection), CWE-90 (LDAP Injection), CWE-91 (XML Injection), CWE-93 (CRLF Injection), etc.

- Classement des menaces du projet WASC:
  http://www.webappsec.org/projects/threat/classes/ldap_injection.shtml
  http://www.webappsec.org/projects/threat/classes/sql_injection.shtml
  http://www.webappsec.org/projects/threat/classes/os_commanding.shtml

- OWASP, http://www.owasp.org/index.php/SQL_Injection

- Guide de l'OWASP, http://www.owasp.org/index.php/Guide_to_SQL_Injection

- Guide d'audit de code de l'OWASP,
  http://www.owasp.org/index.php/Reviewing_Code_for_SQL_Injection

- Guide de tests sécurité de l'OWASP,
  http://www.owasp.org/index.php/Testing_for_SQL_Injection

- Injection SQL, http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

- Injection SQL avancée, http://www.ngssoftware.com/papers/advanced_sql_injection.pdf
- Toujours plus loin dans l'injection SQL avancée, http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf
- Hibernate, gestionnaire de gestion d'objets relationnel (ORM) pour J2EE et .NET, http://www.hibernate.org/
- Requêtes J2EE, http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html
- Comment se protéger de l'injection SQL dans ASP.Net, http://msdn2.microsoft.com/en-us/library/ms998271.aspx
- Les fonctions PDO de PHP, http://php.net/pdo

A3 – EXECUTION DE FICHIER MALICIEUX

L'exécution de fichiers malicieux existe dans beaucoup d'applications. Les développeurs utilisent souvent la possibilité de concaténer des données d'entrée dans des fonctions de gestion de fichiers ou de flux, ou font confiance à des fichiers donnés en entrée. Sur beaucoup de plateformes, les Framework autorisent l'utilisation d'objets externes, comme des URLs ou des références au système de fichier. Lorsque les données ne sont pas correctement vérifiées, cela peut conduire à l'exécution de code distant ou à l'affichage de contenu non voulu lors de l'exécution par le serveur Web.

Cela permet à un attaquant :

- D'exécuter du code à distance
- D'installer des rootkits et compromettre votre système.
- Sur les systèmes Windows, la compromission du système peut être possible via l'utilisation de PHP's SMB file wrappers

Cette attaque est particulièrement répandue sur PHP, et un soin extrême doit être pris avec tout flux ou toute fonction de fichier afin de s'assurer que l'entrée fournie par l'utilisateur n'influence pas les noms de fichiers..

ENVIRONNEMENTS AFFECTÉS

Toute application web est vulnérable à l'exécution malicieuse de fichier si elle accepte des fichiers ou des noms de fichiers donnés par un utilisateur. Des exemples typiques sont : les assemblages .NET qui autorisent des noms de fichiers d'URL en argument, ou le code qui accepte que l'utilisateur fournisse des noms de fichiers locaux.

PHP est particulièrement vulnérable à l'inclusion distante de fichiers (RFI), via les APIs de flux et de manipulation de fichiers.

VULNERABILITE

Un code typique vulnérable:

```
include $_REQUEST['filename'];
```

Not only does this allow evaluation of remote hostile scripts, it can be used to access local file servers (if PHP is hosted upon Windows) due to SMB support in PHP's file system wrappers.

D'autre méthodes d'attaques sont possibles via :

- Le chargement de données hostiles en fichiers de sessions, fichiers de logs ou via des images (type des logiciels de forums)

- L'utilisation de flux de compressions ou audios, tels zlib:// ou ogg:// qui n'inspectent par la configuration de PHP et permettent alors d'accéder à des ressources distantes même si la variable allow_url_fopen ou allow_url_include est désactivée.

- L'utilisation de wrappers PHP, tels que des entrées php:// et autre pour passer des données en POST plutôt que via un fichier.

- L'utilisation de wrappers PHP, comme `data:; base64,PD9waHAgcGhwaW5mbygpOz8+`

Cette liste n'est pas figée (et change régulièrement), il est donc vital d'utiliser une architecture sécurisée et une conception de la sécurité extrêmement robuste lorsque l'on travaille avec les entrées fournies par un utilisateur qui pourraient influencer le choix d'accès ou le nom des fichiers coté serveur.

Bien qu'il n'y ai qu'un exemple en PHP de fourni, cette attaque est possible de manière différente sur .NET et J2EE. Quand aux applications écrites via ces frameworks, il convient de porter une attention particulière à la façon dont les mécanismes de sécurité d'accès sont codés, et vérifier que les noms de fichiers fournis par l'utilisateur ne permettent pas de réduire les contrôles de sécurité.

Par exemple, il est possible que des documents XML fournis par un attaquant comprennent une DTD hostile, forçant alors l'analyser XML à charger une DTD distante, l'analyser, et renvoyer le résultat. Une société de sécurité australienne, a démontré la possibilité d'effectuer un scan de ports derrière les firewalls par cette technique. Voir [SIF01] dans les références de ce chapitre pour plus d'informations.

Les dommages causés directement par cette vulnérabilité sont intimement liés à la robustesse des mécanismes d'isolation du framework. Comme PHP n'est que rarement isolé et ne dispose pas de concept de bac à sable ou d'architecture de sécurité, le dommage peut être pire que les attaques sur d'autres plateformes disposant de mécanismes de confiance, faibles ou élevés ; telle qu'une application Web tournant dans une JVM avec le gestionnaire de sécurité activé et proprement configuré (ce qui est rarement le cas par défaut).

VERIFICATION DE LA SECURITE

Approches automatisées : les scanners de vulnérabilités ont des difficultés pour identifier les paramètres utilisés lors des inclusions de fichier ainsi que leur syntaxe. Les outils d'analyse statique peuvent rechercher l'utilisation d'APIs dangereuses mais ne peuvent vérifier que les mécanismes de protection et de validation des données sont en place pour contrer cette vulnérabilité.

Approches manuelles : Une revue de code peut permettre de déceler le code permettant d'inclure un fichier dans une application, mais il y a aussi de nombreuses erreurs de reconnaissance possibles. Des tests peuvent permettent de détecter ces vulnérabilités, mais l'identification de paramètres bien particuliers et de leur syntaxe peut être difficile.

PROTECTION

Pour prévenir les failles d'inclusion de fichiers, il est nécessaire de faire très attention dans les phases d'architecture et de conception, et aussi lors des tests. En général, une application bien écrite ne doit pas utiliser une donnée entrée par un utilisateur comme nom de fichier pour une ressource du serveur (tel que les images, les documents XML et XSL, ou l'inclusion de scripts), et il est nécessaire de mettre en place des règles de filtrage permettant de limiter les connexions sortantes vers l'Internet ou vers d'autres serveurs internes. Néanmoins, beaucoup d'applications nécessitent d'accepter les données d'entrées d'utilisateurs.

Parmi les considérations les plus importantes:

- **Utilisez une référence indirecte à l'objet (voyez la section AA pour plus de détails)**. Par exemple, lorsque vous avez besoin d'une valeur pour un nom de fichiers, utilisez un hash de ce nom. Au lieu de :

  <select name="language">
  <option value="English">English</option>

  utilisez

  <select name="language">
  <option value="78463a384a5aa4fad5fa73e2f506ecfc">English</option>

  Considérez l'utilisation de salts pour prévenir les attaques par force brute sur la référence indirecte de l'objet. Au choix, utilisez des valeurs d'index comme 1, 2, 3 et vérifier que les valeurs extrêmes sont valides pour détecter la falsification de paramètres.

- Utilisez des mécanismes explicites de vérifications de traces, si votre langage le supporte. Autrement, considérez un schéma de nommage variable pour aider la vérification de traces

```
$hostile = &$_POST; // refer to POST variables, not $_REQUEST

$safe['filename']= validate_file_name($hostile['unsafe_filename']); // make it safe
```

Toute opération basée sur un contenu hostile est immédiatement suspecte:

☒ `require_once($_POST['unsafe_filename'] . 'inc.php');`

☑ `require_once($safe['filename'] . 'inc.php');`

- **Validez fortement les données d'entrées** en utilisant la stratégie d'acceptation uniquement des bonnes valeurs

- **Ajoutez des règles à vos Firewalls** pour empêcher les serveurs web d'effectuer de nouvelles connexions vers des sites web externes ou internes. Pour des systèmes à forte sécurité, isolez le serveur Web dans son propre VLAN ou dans son propre sous-réseau

- **Vérifiez que les fichiers ou noms de fichiers fournis par l'utilisateur** ne puissent se soustraire à d'autres contrôles, tels que altération de données dans l'objet de session, avatars et images, rapports PDF, fichiers temporaires, et ainsi de suite

- **Considérer la mise en place de bac à sable ou** d'autres mécanismes comme la virtualisation pour isoler les applications entre elles

- **PHP: Désactivez allow_url_fopen et allow_url_include** dans le php.ini et essayer de compiler PHP pour ne pas inclure cette fonctionnalité. Très peu d'applications nécessitent ces fonctions et vous devez l'activer application par application

- **PHP: Désactiver register_globals et utilisez E_STRICT pour trouver les variables non initialisées**

- **PHP: Vérifiez que toutes les fonctions de fichiers et de gestion de flux (stream_*) sont** `soigneusement contrôlées`. Vérifier que les données d'entrée utilisateurs ne vont pas être utilisées par des fonctions prenant un nom de fichier en argument, en incluant les fonctions suivantes:

```
include() include_once() require() require_once() fopen() imagecreatefromXXX() file()
file_get_contents() copy() delete() unlink() upload_tmp_dir() $_FILES move_uploaded_file()
```

- PHP: Faites extrêmement attention aux données passées à system() eval() passthru() ou ` (la quote inversée).

- Vérifiez que le gestionnaire de sécurité J2EE est activé, correctement configuré et que les applications demandent bien les permissions.

- Avec ASP.NET, référez vous à la documentation relative à la confiance partielle, et construisez vos applications pour qu'elles soient segmentées en zones de confiance, comme cela la plupart des applications seront créées avec le niveau de confiance le plus restreint.

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0360
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5220
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4722

RÉFÉRENCES

- CWE: CWE-98 (PHP File Inclusion), CWE-78 (OS Command Injection), CWE-95 (Eval injection), CWE-434 (Unrestricted file upload)

OWASP Top 10 2007

- Classification des menaces du WASC: http://www.webappsec.org/projects/threat/classes/os_commanding.shtml
- OWASP Guide, http://www.owasp.org/index.php/File_System#Includes_and_Remote_files
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP PHP Top 5, http://www.owasp.org/index.php/PHP_Top_5#P1:_Remote_Code_Execution
- Stefan Esser, http://blog.php-security.org/archives/45-PHP-5.2.0-and-allow_url_include.html
- [SIF01] SIFT, Web Services: Teaching an old dog new tricks, http://www.ruxcon.org.au/files/2006/web_services_security.ppt
- http://www.owasp.org/index.php/OWASP_Java_Table_of_Contents#Defining_a_Java_Security_Policy
- Microsoft - Programming for Partial Trust,

  http://msdn2.microsoft.com/en-us/library/ms364059(VS.80).aspx

A4 – REFERENCE DIRECTE NON SECURISEE A UN OBJET

Une référence directe existe lorsqu'un développeur affiche une référence vers l'implémentation interne d'un objet, comme un fichier, un répertoire, un enregistrement de base de données, une clef comme un paramètre d'un formulaire ou d'une URL. Un attaquant peut alors manipuler directement la référence à l'objet pour accéder à d'autres objets sans autorisation, même avec des vérifications de contrôle d'accès en place.

Par exemple, dans une application de banque en ligne, il est fréquent d'utiliser le numéro de compte comme clef primaire. Il est alors très tentant d'utiliser le numéro de compte directement dans l'interface Web. Même si le développeur a pris soin de prévenir les injections SQL, si il n'y pas de contrôle d'autorisation en place pour vérifier que le possesseur d'un compte peut le lire, un attaquant peut modifier le paramètre de numéro de compte pour changer ou lire tous les comptes.

Ce type d'attaque est intervenu sur le site de "the Australian Taxation Office's *GST Start Up Assistance"* en 2000, lorsqu'un utilisateur légitime, mais hostile, a simplement changé le numéro ABN (l'identité d'une entreprise de taxi) présent dans l'URL. L'utilisateur a récupéré des données relatives à environ 17000 entreprises du système, et a envoyé un e-mail à chacune avec les détails de l'attaque. Ce type de vulnérabilité est très courant, mais est trop peu testé dans beaucoup d'applications.

ENVIRONNEMENTS AFFECTES

Tous les frameworks d'applications Web sont vulnérables à ces attaques.

VULNERABILITE

Beaucoup d'application exposent des références à des objets internes aux utilisateurs. Un attaquant peut utiliser et modifier un paramètre et violer ainsi les politiques de contrôles d'accès mises en place. Fréquemment, ces références pointent vers des fichiers système ou des bases de données, mais toute autre partie d'une application peut être vulnérable.

Par exemple, si le code autorise l'utilisateur à donner un nom de fichier ou un répertoire, cela peut permettre à un attaquant de sortir de la structure de l'application et d'accéder à d'autres ressources.

```
<select name="language"><option value="fr">Français</option></select>

…

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

Un tel code peut être attaqué via une chaine de type "../../../../etc/passwd%00" via une injection d'octet vide (voir l'OWASP Guide pour plus d'informations) dans le but d'accéder à n'importe quel fichier du système de fichier du serveur Web.

De façon similaire, les références vers les clefs d'une base de données sont fréquemment exposées. Un attaquant peut utiliser ce paramètre en cherchant une clef valide ou en en récupérant une. Elles sont souvent de nature séquentielle. Dans l'exemple ci-dessous, même si l'application ne présente par de lien vers des cartes non autorisées, et que l'injection SQL n'est pas possible, l'attaquant peut changer le paramêtre cartID avec la valeur qu'il veut.

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

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

VERIFICATION DE LA SECURITE

Le but est de vérifier que l'application ne permet par d'accéder directement à une référence d'objet manipulable par un attaquant.

Approches automatisées : Les outils de vérification de sécurité automatisés, ont des difficultés à détecter le paramètre susceptible d'être manipulé ou que la manipulation est possible. Les outils d'analyse statique ne peuvent pas savoir quels paramètres doivent disposer de contrôle d'accès.

Approches manuelles : La revue de code peut permettre d'identifier, de tracer les paramètres critiques susceptibles d'être manipulés. Les tests d'intrusions peuvent aussi vérifier que la manipulation est possible. Ces deux techniques sont très consommatrices de temps.

PROTECTION

La meilleure protection est d'éviter d'exposer une référence directe à un objet à l'utilisateur, grâce à l'utilisation d'un index, une équivalence par référence indirecte ou une autre méthode indirecte facile à valider. Si une référence directe à un objet doit être utilisée, vérifiez que l'utilisateur est autorisé avant de l'utiliser.

La mise en place d'une méthode de référence aux objets d'application est importante

- Evitez d'exposer des références d'objet privé aux utilisateurs, chaque fois que possible, tels les clés primaires ou noms de fichiers
- Validez sans retenue toutes les références aux objets privés, via la méthode d'acceptation des bonnes valeurs
- Vérifiez l'autorisation à tous les objets référencés

La meilleure solution consiste en l'utilisation d'une valeur d'index ou d'une référence permettant de prévenir la manipulation du paramètre.

```
http://www.example.com/application?file=1
```

Si vous devez exposer directement une référence à la structure de la base de données, vérifiez que les commandes SQL et les autres méthodes d'accès à la base ne permettent que la visualisation des enregistrements autorisés :

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

User user = (User)request.getSession().getAttribute( "user" );

String query = "SELECT * FROM table WHERE cartID=" + cartID + " AND userID=" + user.getID();
```

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0329
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4369
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0229

RÉFÉRENCES

- CWE: CWE-22 (Path Traversal), CWE-472 (Web Parameter Tampering)
- Classification des menaces du WASC:
  http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
  http://www.webappsec.org/projects/threat/classes/insufficient_authorization.shtml
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_business_logic
- OWASP Testing Guide,
  http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP, http://www.owasp.org/index.php/Category:Access_Control_Vulnerability
- GST Assist attack details, http://www.abc.net.au/7.30/stories/s146760.htm

A5 – CROSS SITE REQUEST FORGERY (CSRF)

Le Cross-Site Request Forgery (CSRF) n'est pas une nouvelle attaque, mais elle est néanmoins simple et dévastatrice. Une attaque de type CSRF force le navigateur d'une victime ayant une session ouverte sur une application web vulnérable à effectuer une requête sur cette dernière application. Cela implique que le navigateur de la victime va effectuer une action sur un site vulnérable à la place de la victime.

Cette vulnérabilité est extrêmement répandue, toute application web qui

- ne dispose pas de vérifications d'autorisation avant d'effectuer des actions vulnérables
- va effectuer une action si un nom d'utilisateur par défaut est donne dans la requête (ex : http://exemple.com/admin/effectuerAction.ctl?username=admin&password=admin)
- autorise des requêtes basées uniquement sur des informations d'authentification envoyés automatiquement comme les cookies de session si l'utilisateur a une session active dans l'application, ou la fonctionnalité « Se souvenir de moi » si l'utilisateur n'est pas identifié dans l'application, ou un token Kerberos si cela fait parti d'un Intranet avec des identifications sur Active Directory

est alors a risque. De nos jours, la plus part des applications web ne reposent uniquement sur des informations d'authentification envoyées automatiquement comme les cookies de sessions, authentifications HTTP basiques, adresses IP source, certificats SSL, etc.

Cette vulnérabilité est aussi connue sous d'autres noms y compris « Session Riding », « One-Click Attacks », « Cross-Site Reference Forgery », « Hostile Linking » and « Automation Attack », L'acronyme XSRF est aussi fréquemment utilisé. OWASP et MITRE se sont accordés a utiliser le nom Cross-Site Request Forgery et CSRF.

ENVIRONNEMENTS AFFECTES

Tous les Framework d'applications web sont vulnérables à CSRF.

VULNERABILITE

Une attaque CSRF typique contre un forum pourrait prendre la forme de forcer l'utilisateur à invoquer certaines fonctions telles que la page de déconnexion de l'application. La balise suivante dans n'importe quelle page web vue par la victime génèrera une requête qui la déconnectera du forum:

```
<img src="http://www.example.com/logout.php">
```

Si une banque en ligne permettait à son application de procéder à des requêtes, par exemple des transferts de fonds, une attaque similaire pourrait permettre:

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

Dans sa présentation Hacking Intranet Sites from the outside effectuée au BlackHat 2006, Jeremiah Grossman, a démontré qu'il était possible de forcer l'utilisateur à effectuer des modifications dans son routeur DSL sans son consentement ; et même si l'utilisateur ne sait pas que son routeur DSL a une interface web. Jeremiah a utilisé le compte par défaut du routeur pour effectuer son attaque.

Toutes ces attaques fonctionnent car les informations d'authentification de l'utilisateur (typiquement un cookie de session) sont envoyées automatiquement lors de la requête effectuée par le navigateur, même si l'attaquant n'a pas spécifié ces informations.

Si le tag contenant l'attaque peut être envoyé à une application vulnérable, la probabilité de trouver des victimes connectées s'en trouve alors significativement accrue, de façon similaire à l'augmentation du risque entre les failles XSS par stockage ou par réflexion. Une faille de type XSS n'est pas nécessaire pour qu'une attaque CSRF réussisse, bien que toute application présentant des failles XSS soit susceptible d'être vulnérable à CSRF car une telle attaque peut exploiter la faille XSS pour voler des informations qui ne sont pas envoyées automatiquement et qui peuvent être en place pour protéger contre une attaque CSRF. De nombreux vers applicatifs ont utilisé une combinaison de ces deux techniques (CSRF et XSS).

Lorsque vous élaborez des défenses contre les attaques CSRF, vous devez également mettre l'accent sur l'élimination des vulnérabilités XSS dans votre application car celles-ci pourraient être exploitées pour contourner la plupart des défenses contre CSRF que vous pourriez mettre en place.

VERIFICATION DE LA SECURITE

Le but est de vérifier que l'application se protège de CSRF en générant et requérant certains tokens d'authentification qui ne sont pas envoyés automatiquement par le navigateur.

Approche automatisée : De nos jours, peu de scanners automatiques peuvent détecter CSRF, même si la détection de CSRF serait possible pour des scanners relativement puissants. En revanche, si votre scanner trouve une faille de type XSS et que vous n'avez pas de protection contre CSRF, vous êtes très probablement susceptible d'être vulnérable à une attaque CSRF classique et répertoriée.

Approche manuelle : Les tests d'intrusion (Penetration Tests, ou pen-tests, en anglais) sont des moyens rapides et efficaces pour vérifier que des protections contre CSRF sont en place. Pour vérifier la robustesse et la fiabilité du mécanisme utilisé, une vérification de code est la manière la plus efficace.

PROTECTION

Les applications doivent être certaines qu'elles ne dépendent pas uniquement d'informations ou tokens envoyés automatiquement par un navigateur. La seule solution est d'utiliser un token personnalisé que le navigateur ne pourra « mémoriser » et donc pas envoyer automatiquement par une attaque CSRF.

Les stratégies suivantes devraient être inhérentes à toutes les applications web :

- **S'assurer qu'il n'y a pas de vulnérabilité de type XSS dans l'application** (cf. A1 - Cross Site Scripting)

- **Insérer des tokens uniques et aléatoires dans chaque formulaire et URL** qui ne seront pas envoyés automatiquement par le navigateur. Par exemple,

```
<form action="/transfer.do" method="post">

<input type="hidden" name="8438927730" value="43847384383">

…
</form>
```

  et ensuite vérifier que le token envoyé par le navigateur est correct pour l'utilisateur en cours. De tels tokens peuvent être uniques pour chaque fonction ou page pour l'utilisateur, ou simplement uniques à la session globale ; tout dépend de la granularité choisie. Plus le token a une granularité fine (unique par fonction, etc.), plus la protection est importante, mais cela est plus compliqué à élaborer et à maintenir.

- **Pour des données sensibles ou des transactions financières, ré-authentifiez ou utilisez la signature de transaction** afin de s'assurer que la requête est authentique. Mettez en place des mécanismes externes tels le courrier électronique ou le contact téléphonique afin de vérifier les demandes ou en notifier l'utilisateur.

- **N'utilisez pas de requêtes GET (URLs) pour les données sensibles ou pour effectuer des transactions de valeur.** Utiliser uniquement la méthode POST lorsque vous recevez des données de l'utilisateur. En revanche, l'URL peut contenir le token aléatoire car ceci crée une URL unique, ce qui rend une attaque CSRF presque impossible à exécuter.

- **POST à lui seul n'est pas une protection suffisante.** Vous devez également l'utiliser en combinaison avec des jetons aléatoire, hors phase d'authentification ou de réauthentification, afin de vous protéger correctement contre CSRF.

- **Pour ASP.NET,** utilisez ViewStateUserKey. (Voire la référence). Ceci fournit un type de contrôle similaire à un jeton aléatoire comme décrit ci-dessus.

Bien que ces suggestions aident à réduire radicalement votre exposition, les attaques CSRF

évoluées peuvent contourner bon nombre de ces restrictions. La méthode la plus efficace reste l'utilisation de jetons uniques, et l'élimination de toutes les vulnérabilités XSS dans votre application.

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0192
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5116
- L'interview de Samy (MySpace): http://blog.outer-court.com/archive/2005-10-14-n81.html
- Une attaque qui utilise QuickTime pour effectuer des attaques CSRF http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9005607&intsrc=hm_list

RÉFÉRENCES

- CWE : CWE-352 (Cross-Site Request Forgery)
- WASC Threat Classification : Pas de sujet direct, mais ce qui suit en est très proche http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
- OWASP CSRF, http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- OWASP Testing Guide, https://www.owasp.org/index.php/Testing_for_CSRF
- OWASP CSRF Guard, http://www.owasp.org/index.php/CSRF_Guard
- OWASP PHP CSRF Guard, http://www.owasp.org/index.php/PHP_CSRF_Guard
- RSnake, “What is CSRF?”, http://ha.ckers.org/blog/20061030/what-is-csrf/
- Jeremiah Grossman, présentation et démonstration de “Hacking Intranet sites from the outside” http://www.whitehatsec.com/presentations/whitehat_bh_pres_08032006.tar.gz
- Microsoft, ViewStateUserKey détails, http://msdn2.microsoft.com/en-us/library/ms972969.aspx#securitybarriers_topic2

A6 – FUITE D'INFORMATION ET TRAITEMENT D'ERREUR INCORRECT

Les applications peuvent involontairement dévoiler des informations sur leur configuration, fonctionnement interne, ou violer la vie privée par toute sorte de problèmes applicatifs. Les applications peuvent également dévoiler des informations sur leur état interne par l'intermédiaire du temps qui leur est nécessaire au traitement de certaines opérations ou via différentes réponses à des entrées différentes, par exemple afficher le même message d'erreur avec des numéros d'erreurs différents. Les applications web dévoileront souvent des informations sur leur état interne à cause de messages d'erreurs détaillés ou de débogage. Ces informations peuvent souvent être utilisées pour déclencher, voire même automatiser des attaques plus puissantes.

ENVIRONNEMENTS AFFECTES

Tous les Framework d'application web sont vulnérables aux fuites d'informations et aux traitements d'erreurs incorrects.

VULNERABILITE

Les applications génèrent fréquemment des messages d'erreurs et les affichent aux utilisateurs. Ces messages d'erreurs sont souvent très utiles pour les attaquants car ils peuvent dévoiler d'importants détails sur l'implémentation ou des informations utiles pour exploiter des vulnérabilités. Il y a de nombreux exemples communs :

- Les traitements d'erreurs détaillés qui génèrent un affichage de messages d'erreur comprenant trop d'information comme le niveau de la pile, les requêtes SQL qui ont échoué ou d'autres informations de débogage

- Les fonctions qui produisent des résultats différents basés sur des entrées différentes. Par exemple, essayer d'authentifier un même nom d'utilisateur avec des mots de passes différents devrait produire le même message d'erreur avec « nom d'utilisateur inconnu », et « mauvais mot de passe ». Cependant, de nombreux systèmes retournent des codes d'erreur différents

VERIFICATION DE LA SECURITE

Le but est de vérifier que l'application ne dévoile pas d'information via des messages d'erreurs ou d'autres moyens.

Approche automatisée : Les scanners de vulnérabilités vont généralement générer des messages d'erreurs. Les outils d'analyse statique peuvent rechercher l'utilisation d'APIs susceptibles de dévoiler de l'information, mais ne seront pas capables de vérifier la signification de ces messages.

Approche manuelle : Un audit de code peut rechercher les traitements d'erreurs incorrects et autres modèles de fuite d'information, mais cela prend tu temps. Le test générera aussi des messages d'erreurs, mais savoir quels messages d'erreurs ont été couverts est un challenge.

PROTECTION

Les développeurs devraient utiliser des outils comme WebScarab (OWASP) pour faire générer des erreurs à leur application. Les applications qui n'ont pas été testées de cette façon généreront certainement des erreurs inattendues. Les applications devraient aussi intégrer une architecture standard de traitement d'exceptions afin d'empêcher toute fuite d'information non désirée à destination des attaquants.

Prévenir la fuite d'informations requiert une vraie discipline. Les méthodes suivantes ont prouvé leur efficacité :

- **Assurez-vous que toute l'équipe de développement partage la même approche relative au traitement d'exceptions**

- **Désactivez ou limiter le traitement d'erreur détaillé**. En particulier, ne pas afficher les informations de débogage aux utilisateurs, l'état de la pile ou des informations relatives aux chemins

- **Assurez que les « path » sécurisés qui ont des résultats multiples retournent des messages d'erreur similaires ou identiques** approximativement dans le même temps. Si ce n'est pas possible, envisager d'imposer un temps de réponse aléatoire pour toutes les transactions pour cacher cette information à l'attaquant

- Différentes couches peuvent retourner des erreurs fatales ou une exception, telle la couche base de données, le serveur web sous-jacent (IIS, Apache, etc.). Il est crucial que **les erreurs venant de chaque couche soient dûment vérifiées et configurées pour empêcher les messages d'erreur d'être exploités par des intrus**

- Il faut savoir que les Framework courants retournent des erreurs HTTP différentes suivant que l'erreur est dans votre code ou dans celui du Framework lui-même. Il est intéressant de créer un gestionnaire par défaut d'erreur qui retourne un message d'erreur correctement nettoyé pour la plupart des utilisateurs en production pour tous les « path » d'erreur

- Modifier le traitement d'erreurs par défaut pour qu'il retourne tout le temps une erreur « 200 » (OK) réduit la possibilité qu'un outil automatisé aurait de déterminer si une erreur sérieuse s'est produite. Bien que ce soit de la « sécurité par l'obscurité », cela fournit un niveau de défense supplémentaire

- Certaines grandes entreprises ont choisi d'inclure des codes d'erreurs aléatoires ou

uniques à travers toutes leurs applications. Ceci peut aider le help desk à trouver une solution correcte à un problème particulier, mais cela peut aussi permettre aux attaquants de déterminer exactement comment une application s'est terminée

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4899
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3389
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0580

RÉFÉRENCES

- CWE: CWE-200 (Information Leak), CWE-203 (Discrepancy Information Leak), CWE-215 (Information Leak Through Debug Information), CWE-209 (Error Message Information Leak), autres.
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/information_leakage.shtml
- OWASP, http://www.owasp.org/index.php/Error_Handling
- OWASP, http://www.owasp.org/index.php/Category:Sensitive_Data_Protection_Vulnerability

A7 – VIOLATION DE GESTION D'AUTHENTIFICATIONS ET DE SESSIONS

La gestion de sessions avec une authentification appropriée est un mécanisme critique pour une application web. Le plus souvent, les failles dans ce genre de mécanismes se traduisent par une incapacité de l'application à protéger les informations relatives aux droits ainsi que les tokens de sessions par leur cycle de vie. Ces failles peuvent mener au détournement de comptes utilisateur ou administratif, altérer les contrôles d'autorisation et de responsabilité, et provoquer une violation de la vie privée.

ENVIRONNEMENTS AFFECTES

Tous les Framework d'applications web sont vulnérables à des failles dans la gestion d'authentifications et de sessions.

VULNERABILITE

Les faiblesses dans le mécanisme principal d'authentification ne sont pas rares mais les faiblesses sont plus souvent introduites par des fonctions auxiliaires d'authentification telle la déconnexion, la gestion de mot de passe, le timeout, les questions secrètes et les mises à jour des informations relatives au compte.

VERIFICATION DE LA SECURITE

Le but est de vérifier que l'application authentifie correctement les utilisateurs et protège correctement les identités tout autant que les droits associés.

Approche automatisée : Les outils de balayage de vulnérabilité mettent un temps très long à détecter des vulnérabilités dans des schémas de gestion d'authentification et de session personnalisés. Dans le même esprit, les outils d'analyse statique ne sont pas non plus susceptibles de détecter des problèmes de gestion d'authentification dans un code personnalisé.

Approche manuelle : L'audit de code ainsi que des tests, spécialement en association, sont très efficaces pour vérifier que les mécanismes d'authentification, de gestion de session, et de fonctions auxiliaires sont correctement implémentés.

PROTECTION

L'authentification repose sur la sécurité des communications et du stockage des informations d'identification. Assurez-vous d'abord qu'SSL est la seule option pour toutes les parties authentifiées de l'application (cf. A9 – Communications non sécurisées) et que toutes les informations d'identifications sont stockées sous forme de hash ou encryptées (cf. A8 – Stockage cryptographique non sécurisé).

Prévenir les failles d'authentification demande une planification rigoureuse et de se plier à de nombreuses règles, dont parmi les plus importantes :

- **Utiliser uniquement des mécanismes de gestion de sessions intégrés.** N'utiliser sous aucun prétexte des mécanismes secondaires ou faits maison

- **N'acceptez pas des identifiants de sessions, nouveaux, préréglés ou invalides** à partir de l'URL ou dans la requête. Ce genre d'attaque porte le nom de « Session Fixation »

- **Limitez ou supprimez de votre code les cookies personnalisés relatifs à l'authentification ou la gestion de session** tels que les fonctionnalités de type « Remember me » ou les systèmes d'Authentification Unique (Single Sign-On - SSO) faits maison. Ceci ne s'applique pas aux SSO ou solutions d'authentifications éprouvées

- **Utilisez un mécanisme d'authentification unique** avec un niveau de sécurité et un nombre de facteurs appropriés. Assurez-vous que ce mécanisme n'est pas facilement sujet au « Spoofing » ou aux attaques par rejeux (Replay Attacks). Faites en sorte que ce mécanisme ne soit pas excessivement complexe, car il pourrait alors devenir sujet à sa propre attaque

- **N'autorisez pas le démarrage du processus d'authentification à partir d'une page non cryptée.** Commencez toujours le processus d'ouverture de session à partir d'une seconde page chiffrée avec un nouveau jeton de session afin d'empêcher le vol de session ou d'informations, les attaques par Phishing ou les attaques de type « Session Fixation »

- Envisagez de **générer un nouveau jeton de session (Token) à chaque changement d'authentification** ou de niveau de privilège

- **Assurez-vous que toutes les pages disposent d'un lien de déconnexion**. La déconnexion doit totalement détruire tout état de session côté serveur et cookies côté client. Considérez aussi le facteur humain : ne demandez pas confirmation car les utilisateurs fermeront juste l'onglet ou la fenêtre plutôt que se déconnecter proprement

- **Utiliser une période de timeout** qui déconnecte automatiquement tout utilisateur après une période d'inactivité

- **Utiliser uniquement des fonctions d'authentification annexes sécurisées** (questions et réponses, réinitialisation du mot de passe) car ces fonctions traitent des informations importantes au même titre que le sont les noms d'utilisateurs et mots de passe ou jeton de session. Utiliser un hash à sens unique aux réponses pour empêcher les attaques par révélation

- **N'exposez aucun identifiant de session ou identifiant valide dans les URL ou les logs (aucune réécriture de session ou enregistrement du mot de passe de l'utilisateur dans les journaux d'évènements)**

- **Vérifier l'ancien mot de passe lorsque l'utilisateur change pour un nouveau mot de passe**

- **Ne pas se baser sur des informations falsifiables comme forme d'authentification unique** telles les adresses IP ou masques de plage d'adresses, DNS ou Reverse DNS, en-têtes de Referrer ou identique

- **Soyez prudent lors de l'envoi d'informations confidentielles à une adresse email** en tant que mécanisme de réinitialisation de mot de passe (cf. RSNAKE01). Utilisez uniquement des nombres aléatoires à durée limitée pour réinitialiser l'accès et envoyer un e-mail de confirmation dès que le mot de passe a été réinitialisé. Attention si vous permettez à des utilisateurs auto-enregistrés de changer leur adresse e-mail - envoyer un message à l'adresse e-mail précédente avant de décréter le changement

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6229
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6528

RÉFÉRENCES

- CWE: CWE-287 (Authentication Issues), CWE-522 (Insufficiently Protected Credentials), CWE-311 (Reflection attack in an authentication protocol), autres.
- WASC Threat Classification:
  http://www.webappsec.org/projects/threat/classes/insufficient_authentication.shtml
  http://www.webappsec.org/projects/threat/classes/credential_session_prediction.shtml
  http://www.webappsec.org/projects/threat/classes/session_fixation.shtml
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Authentication
- OWASP Code Review Guide,
  http://www.owasp.org/index.php/Reviewing_Code_for_Authentication
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_authentication
- RSNAKE01 - http://ha.ckers.org/blog/20070122/ip-trust-relationships-xss-and-you

A8 – STOCKAGE DE DONNEES CRYPTOGRAPHIQUES NON SECURISE

La protection des données sensibles avec des moyens cryptographiques est devenue une partie maîtresse de la plupart des applications Web. Ne pas chiffrer des données sensibles est très répandu. Les applications qui utilisent fréquemment des moyens de chiffrement contiennent une cryptographie mal conçue, soit en utilisant des procédés de chiffrement inappropriée ou faisant de graves erreurs en utilisant des procédés de chiffrement forts. Ces vulnérabilités peuvent mener à une divulgation de données sensibles et à des violations de conformité.

ENVIRONNEMENTS AFFECTES

Tous les contextes d'applications web sont vulnérables au stockage de données cryptographiques peu sûr.

VULNERABILITE

La prévention des failles cryptographiques demande une planification soigneuse. Les problèmes les plus courants sont :

- Pas de chiffrement des données sensibles
- Utilisation d'algorithmes de chiffrement « maison »
- Utilisation incorrecte d'algorithmes robustes
- Utilisation récurrente d'algorithmes connus pour leurs faiblesses (MD5, SHA-1, RC3, RC4, etc.)
- Clés codés en dur, stockage de clés dans des zones non protégées

VERIFICATION DE LA SECURITE

Le but est de vérifier que l'application chiffre correctement les informations sensibles lors du stockage.

Approches automatiques : Les outils de scan de vulnérabilités ne peuvent pas du tout vérifier le moyen de stockage cryptographique. Les outils d'analyse de code peuvent détecter l'utilisation d'API cryptographiques connues, mais ne peuvent pas détecter si leur utilisation est effectuée de manière correcte ou si le chiffrement est effectué par un composant externe.

Approches manuelles : Comme l'analyse automatique, le test (boîte noire) ne peut pas vérifier le moyen de stockage cryptographique. L'audit de code est la meilleure manière de vérifier que l'application chiffre de manière correcte les données sensibles et que le mécanisme et la gestion de la clé de chiffrement sont correctement implémentés. Cela peut dans certains cas impliquer l'étude de la configuration de certains systèmes externes.

PROTECTION

L'aspect le plus important est de s'assurer que tout ce qui doit être chiffré l'est actuellement. Vous devez ensuite vous assurer que le mécanisme de chiffrement est implémenté de manière correcte. Comme il y a beaucoup trop de manières incorrectes d'utiliser la cryptographie, les recommandations suivantes doivent être considérées comme une partie de votre stratégie de test pour s'assurer que les données cryptographiques sont manipulées de manière sécurisée :

- **Ne pas créer d'algorithmes de chiffrement**. Utiliser seulement des algorithmes reconnus publiquement tels qu'AES, cryptographie à clé publique RSA, et SHA-256 ou mieux pour la fonction de hachage
- **Ne pas utiliser d'algorithmes faibles**, tels MD5 / SHA1. Favoriser des alternatives plus sûres, comme SHA-256 ou mieux
- **Générer les clés hors-ligne et stocker les clés privées avec une extrême précaution**. Ne jamais communiquer les clés privées à travers des canaux peu sûrs.
- **Assurez-vous que l'infrastructure d'accréditations comme les bases de données d'identités ou les détails d'accès des files d'attentes MQ sont correctement sécurisés** (via des permissions et des contrôles renforcés sur le système de fichiers)
- **Assurez-vous que les données chiffrées sur disque ne sont pas faciles à décrypter**. Par exemple, le chiffrement de base de données est inutile si le pool de connexion à la base fournit un accès non chiffré.
- Conformément à l'exigence 3 du « PCI Data Security Standard », vous devez protéger les données du titulaire de la carte. La conformité au standard PCI DSS est obligatoire d'ici à 2008 pour les commerçants et toute personne effectuant des traitements avec les cartes de crédit. **La bonne pratique est de ne jamais stocker de données inutiles**, telles que l'information contenue dans la bande magnétique ou le numéro de compte principal (PAN – Primary Account Number). Si vous stockez le PAN, la conformité aux exigences DSS est importante. Par exemple, vous n'êtes JAMAIS autorisés à stocker le numéro « CVV » (i.e. Card Verification Value, correspondant au trigramme situé au dos de la carte) quelles que soient les circonstances. Pour plus d'information, veuillez vous référer au guide « PCI Guidelines and implement controls » si nécessaire.

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1664
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1101 (Applicable à la plupart des serveurs J2EE)

RÉFÉRENCES

- CWE: CWE-311 (Failure to encrypt data), CWE-326 (Weak Encryption), CWE-321 (Use of hard-coded cryptographic key), CWE-325 (Missing Required Cryptographic Step), autres.
- WASC Threat Classification: No explicit mapping
- OWASP, http://www.owasp.org/index.php/Cryptography
- Guide OWASP, http://www.owasp.org/index.php/Guide_to_Cryptography
- OWASP, http://www.owasp.org/index.php/Insecure_Storage
- OWASP, http://www.owasp.org/index.php/How_to_protect_sensitive_data_in_URL's
- PCI Data Security Standard v1.1, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Bruce Schneier, http://www.schneier.com/
- CryptoAPI Next Generation, http://msdn2.microsoft.com/en-us/library/aa376210.aspx

A9 – COMMUNICATIONS NON SECURISEES

Les applications ont souvent des défaillances lors du chiffrement du trafic réseau quand il est nécessaire de protéger des communications sensibles. Le chiffrement (habituellement SSL) doit être utilisé pour toutes les connexions authentifiées, et plus spécifiquement lors d'accès aux pages de sites WEB sans oublier les connexions vers les serveurs back-office. Sinon, l'application expose en clair les éléments d'authentification ou de session. En plus, le chiffrement doit être utilisé à chaque fois qu'il y a des informations sensibles, comme des informations de cartes de crédit, de cartes de santé ou lorsque des informations de santé sont transférées. Les applications qui permettent le retour arrière ou qui peuvent être contraintes de sortir d'un mode de chiffrement peuvent être attaquées par des pirates.

Le standard PCI (Payment Card Industry) exige que toutes les informations relatives aux cartes de crédit échangées sur internet doivent être chiffrées.

ENVIRONNEMENTS CONCERNES

Tous les frameworks d'application WEB sont vulnérables aux communications non sécurisées.

VULNERABILITE

Avoir des défaillances dans le chiffrement de communications sensibles signifie qu'un attaquant qui peut écouter le trafic d'un réseau peut avoir accès aux informations qui circulent, incluant tous les éléments d'authentification ou les données sensibles échangées. Ayez à l'esprit que les différents réseaux sont plus ou moins susceptibles d'être écoutés. Toutefois, il est important de réaliser qu'une machine hôte pourrait être éventuellement compromise sur presque chaque réseau, et que les attaquants installeront rapidement un « sniffer » pour capturer les éléments d'authentification des autres systèmes.

Utiliser SSL pour les communications avec des utilisateurs est primordial, car il est vraiment probable qu'ils utilisent des réseaux non sécurisés pour accéder aux applications. Parce que http inclut des éléments d'authentification ou de sessions dans chaque requête, tous les trafics liés à l'authentification doivent passer par SSL, et pas seulement la requête d'authentification.

Il est aussi important de chiffrer les communications avec les serveurs back-office. Bien que ces réseaux sont généralement plus sécurisés, les informations et les éléments d'authentification qu'ils véhiculent sont plus sensibles et plus importantes. Donc, l'utilisation de SSL pour communiquer avec les serveurs back-office est aussi primordiale.

Chiffrer des données sensibles, comme les informations de cartes de crédit ou les numéros de sécurité sociale, est devenu une exigence réglementaire dans le domaine de la finance et de la vie privée pour beaucoup d'organisations. Négliger l'utilisation de SSL pour des connexions transportant ce type d'information crée un risque d'absence de conformité à ces règles.

VERIFICATION DE LA SECURITE

Le but est de vérifier que l'application chiffre correctement toutes les communications authentifiées et sensibles.

Approches automatisées : les outils de recherche de vulnérabilités peuvent vérifier que SSL est utilisé au niveau du front-office, et ils peuvent trouver des vulnérabilités relatives à l'utilisation de SSL. Toutefois, ces outils n'ont pas accès aux connexions back-office et ils ne peuvent pas vérifier qu'elles sont sécurisées. Des outils d'analyse statiques peuvent permettre d'analyser certains échanges avec les systèmes back-office, mais ils ne comprendront probablement pas la logique spécifique à chaque système.

Approches manuelles : le test peut vérifier que SSL est utilisé et trouver les vulnérabilités correspondantes au niveau du front office, mais les approches automatiques sont sans doute plus efficaces. La revue de code est relativement efficace pour vérifier l'utilisation correcte de SSL pour les connexions back-office.

PROTECTION

La protection la plus adaptée est l'utilisation de SSL pour toutes les connexions authentifiées et à chaque fois qu'une donnée sensible est transmise. La configuration de SSL pour les applications WEB nécessitent de maîtriser pas mal de détails, donc il est important de bien comprendre et analyser votre environnement. Par exemple, IE 7.0 affiche une barre verte quand des certificats SSL de haute confiance sont utilisés mais ce n'est pas un contrôle approprié pour prouver l'utilisation sûre de la cryptographie.

- Utiliser SSL pour toutes les connexions qui sont authentifiées ou qui transmettent des données sensibles, comme des éléments d'authentification, les détails d'une carte de crédit, d'une carte de santé ou autres informations privées
- Assurer que les communications entre les composants d'une infrastructure, comme entre des serveurs WEB et des serveurs de bases de données, sont protégées de façon appropriée grâce à l'utilisation d'une couche de transport sécurisée ou d'un chiffrement au niveau protocole pour les éléments d'authentification et les données intrinsèques
- Au niveau des exigences de niveau 4 du standard de sécurité PCI, vous devez protéger les informations du porteur de carte bancaire qui transitent. La conformité au « DSS PCI » est obligatoire, à partir de 2008, pour les commerçants, et toute personne qui traite des informations de cartes bancaires. En général, les accès aux systèmes par des clients, des partenaires, le personnel et l'équipe administrative doivent être chiffrés en utilisant SSL ou un mécanisme similaire. Pour plus d'informations, consulter le guide d'utilisation du « DSS PCI » et implémenter les contrôles nécessaires.

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6430
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4704
- http://www.schneier.com/blog/archives/2005/10/scandinavian_at_1.html

RÉFÉRENCES

- CWE: CWE-311 (Défaillance dans le chiffrement de données), CWE-326 (Chiffrement faible), CWE-321 (Utilisation de clé cryptographique hard-codée), CWE-325 (absence d'étape de chiffrement nécessaire), autres.
- Classification de menaces WASC : mapping non explicite
- Guide de test OWASP, Test de SSL/TLS https://www.owasp.org/index.php/Testing_for_SSL-TLS
- Guide OWASP, http://www.owasp.org/index.php/Guide_to_Cryptography
- Foundstone - SSL Digger, http://www.foundstone.com/index.htm?subnav=services/navigation.htm&subcontent=/services/overview_s3i_des.htm
- NIST, SP 800-52 Aides pour la sélection et l'utilisation d'implémentations de couche de transport sécurisée (TLS), http://csrc.nist.gov/publications/nistpubs/800-52/SP800-52.pdf
- NIST SP 800-95 Guide pour sécuriser les services WEB, http://csrc.nist.gov/publications/drafts.html#sp800-95

A10 – DEFAILLANCE DANS LA RESTRICTION DES ACCES URL

Généralement, la seule protection pour l'accès à une URL est que les liens vers la page ne sont pas affichés pour des utilisateurs non autorisés. Par contre, un attaquant motivé, compétent, ou tout simplement très chanceux, peut trouver et accéder ces pages, invoquer les fonctions correspondantes et voir les données. La sécurité « par obscurité » n'est pas suffisante pour protéger les fonctions sensibles et les données dans une application. Les vérifications des contrôles d'accès doivent être effectuées avant qu'une requête d'accès aux fonctions sensibles soit autorisée, ce qui assure que l'utilisateur est autorisé à accéder à cette fonction.

ENVIRONNEMENTS CONCERNÉS

Tous les frameworks d'applications WEB sont vulnérables aux défaillances dans la restriction des accès URL.

VULNERABILITE

La première méthode d'attaque pour cette vulnérabilité est appelée "forced browsing », qui regroupe la recherche de liens, des techniques d'attaque par force brute pour trouver des pages non protégées. Les applications fournissent souvent un mécanisme de contrôle d'accès pour faire évoluer et étendre un code de base, résultant en un modèle complexe qui est difficile à comprendre pour les développeurs et spécialistes de la sécurité. Cette complexité rend plus que probable l'occurrence d'erreurs et de pages non traitées qui restent exposées aux attaques.

Des exemples communs de ces vulnérabilités sont :

- Des URLs "cachées" ou "spéciales", accessibles uniquement aux administrateurs ou aux utilisateurs privilégiés au niveau de la couche présentation, mais accessibles à tous les utilisateurs si ils savent qu'elles existent comme /admin/adduser.php ou /approveTransfer.do. C'est particulièrement répandu dans le code qui gère les menus.
- Les applications autorisent souvent des accès à des fichiers « cachés » comme du XML statique ou des rapports système générés, la sécurité reposant sur « l'obscurité » pour les cacher.
- Le code qui renforce la politique de contrôle d'accès mais qui n'est pas à jour ou qui est insuffisant. Par exemple, imaginez /approveTransfer.do qui était initialement accessible pour tous les utilisateurs mais qui, depuis que les contrôles exigés par SOX ont été mis en place, ne soit plus accessible qu'aux approbateurs. Une correction pourrait être de ne pas présenter la page aux utilisateurs non autorisés mais qu'aucun contrôle d'accès ne soit mis en place lorsque la page est demandée.
- Le code qui évalue les privilèges, stocké sur le client et non sur le serveur, comme dans "attack on MacWorld 2007", qui a validé que « Platinium » valait 1700$ par javascript dans le browser au lieu de le faire côté serveur...

VÉRIFICATION DE LA SÉCURITÉ

Le but est de vérifier que le contrôle d'accès est renforcé continuellement tant au niveau de la couche présentation qu'au niveau de la logique métier pour toutes les URLs dans l'application.

Approches automatiques : les scanners de vulnérabilités et les outils d'analyse statique ont des difficultés pour vérifier les contrôles d'accès aux URLs, pour différentes raisons. Les scanners de vulnérabilités ont des difficultés pour découvrir les pages cachées et déterminer celles autorisées pour chaque utilisateur, alors que les moteurs d'analyse statique peinent à identifier les contrôles d'accès personnalisés dans le code et ont du mal à relier la couche présentation avec la logique métier.

Approches manuelles : l'approche la plus efficace et la plus précise est d'utiliser une combinaison de revues et de test sécurité pour vérifier les mécanismes de contrôle d'accès. Si le mécanisme est centralisé, la vérification peut être relativement efficace. Si le mécanisme est distribué à travers tout le code de base, la vérification peut prendre beaucoup plus de temps. Si le mécanisme est étendu de façon externe, la configuration doit être examinée et testée.

PROTECTION

Prendre le temps de planifier les autorisations en créant une matrice pour mettre en correspondance les rôles et les fonctions des applications est une étape clé pour assurer la protection des accès URL non restreints. Les applications WEB doivent renforcer le contrôle d'accès à chaque URL et chaque fonction métier. Il n'est pas suffisant de mettre un mécanisme de contrôle d'accès au niveau de la couche présentation et de laisser la logique métier non protégée. Il n'est pas non plus suffisant de réaliser une vérification unique pendant le processus qui assure que l'utilisateur est authentifié et après ne pas vérifier les étapes suivantes. Autrement, un attaquant peut simplement sauter l'étape où l'authentification est vérifiée, et forger les valeurs de paramètres nécessaires pour passer à l'étape suivante.

Mettre en place le contrôle d'accès sur les URL nécessite de suivre un planning sérieux. Parmi les points importants à ne pas oublier, il y a :

- **Assurez-vous que la matrice de contrôle d'accès faire partie de la couche métier, de l'architecture et de la conception de l'application,**
- **Assurez-vous que toutes les URLs et les fonctions métier sont protégées par un mécanisme de contrôle d'accès efficace** qui vérifie le rôle de l'utilisateur et les droits associés avant chaque traitement. Assurez-vous que c'est fait à chaque étape, et non seulement au début de chaque processus multi-étapes.

- **Effectuez un test de pénétration** avant chaque déploiement ou livraison de code pour vous assurer que l'application ne peut pas être utilisée à mauvais escient par des attaquants mal intentionnés.

- **Prêtez une attention toute particulière à l'inclusion de fichiers et de librairies,** et plus particulièrement s'ils ont une extension « exécutable » comme .php.

Quand c'est possible, ils doivent être stockés en dehors du répertoire web racine. Vous devez vérifier qu'ils ne peuvent pas être accédés directement, par exemple en vérifiant une constante qui peut simplement être créée par la librairie appelante.

- **Ne supposez pas que les utilisateurs ne connaissent pas les URLs ou les APIs spéciales ou cachées.** Assurez-vous toujours que les actions d'administration ou nécessitant des privilèges importants sont protégées.
- **Bloquez l'accès à tous les types de fichier que votre application n'utilisera jamais.** Idéalement, le filtre doit suivre l'approche « tout ce qui est connu est bon » et seulement les types de fichiers autorisés que vous avez décidé d'utiliser, comme par exemple html, pdf, php. Cela va alors bloquer les différents essais d'accès aux fichiers de logs, aux fichiers XML, etc., dont vous n'avez jamais planifié d'utilisation directe.
- **Être à jour à niveau concernant la protection antivirus et des patches** des composants comme les moteurs de traitement XML, les moteurs de traitement de mots, les moteurs de traitement d'images, etc. qui manipulent des données utilisateur

EXEMPLES

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0147
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0131
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1227

RÉFÉRENCES

- CWE: CWE-325 (Requête directe), CWE-288 (Authentification détournée par la méthode de "chemin alternatif"), CWE-285 (Contrôle d'accès absent ou illogique)
- Classification des menaces WASC: http://www.webappsec.org/projects/threat/classes/predictable_resource_location.shtml
- OWASP, http://www.owasp.org/index.php/Forced_browsing
- Guide OWASP, http://www.owasp.org/index.php/Guide_to_Authorization

QUE FAIRE MAINTENANT

Le Top 10 de l'OWASP marque juste le commencement de votre journée de sécurisation des applications web.

> *Les six milliards de personnes dans le monde peuvent être divisées en deux groupes: Groupe 1, ceux qui savent pourquoi tout bon fournisseur de logiciel vend des produits avec des bogues connus, et Groupe 2, ceux qui l'ignorent. Ceux du Groupe 1 tendent à oublier ce qu'était la vie avant que notre optimisme récent n'ait été corrompu par la réalité. Parfois, nous rencontrons une personne du Groupe 2… choquée qu'un fournisseur de logiciel puisse vendre un produit avant que chaque bogue récent ne soit « fixé » (i.e. réparé).*
>
> *Eric Sink, Guardian, 25 Mai 2006*

La plupart de vos utilisateurs et clients sont dans le Groupe 2. La façon dont vous allez traiter ce problème est une occasion d'améliorer votre code et l'état de la sécurité des applications web en général. Des milliards de dollars sont perdus chaque année, et plusieurs millions de personnes souffrent d'usurpation d'identité et de fraude en raison des vulnérabilités citées dans le présent document.

POUR LES ARCHITECTES ET DESIGNERS

Pour sécuriser correctement vos applications, vous devez savoir ce que vous sécurisez (classification des « assets », i.e. des biens), connaître les menaces et les risques d'insécurité, et adresser ces derniers d'une manière structurée. La conception d'une application non triviale exige une bonne dose de sécurité.

- **Assurez-vous que vous appliquez « juste assez » de sécurité en fonction de la modélisation des risques et de la classification des actifs**. Toutefois, comme la conformité aux lois (SOX, HIPAA, Bâle, etc.) représentent une charge croissante, il peut être approprié d'investir plus de temps et de ressources afin de satisfaire au minimum aujourd'hui, en particulier si les pratiques sont bien connues et sont considérablement plus dures que le minimum
- **Posez des questions sur les besoins métiers**, en particulier pré-requis non fonctionnels manquants
- Travaillez sur l' OWASP Secure Software Contract Annex avec votre client
- **Encouragez un design plus sûr** – intégrez la notion de défense en profondeur et des constructions plus simples en employant la modélisation de menace (voir [HOW1] dans les ouvrages de référence)
- **Assurez-vous que vous avez considéré les notions de confidentialité, intégrité, disponibilité, et de non-répudiation**

- **Assurez-vous que vos designs / conceptions sont compatibles avec votre politique sécurité et aux standards,** tels que COBIT ou PCI DSS 1.1

POUR LES DEVELOPPEURS

Beaucoup de développeurs prennent déjà bien en compte les bases de sécurisation des applications web. Pour s'assurer une maîtrise efficace de la sécurité applicative web, le domaine exige de la pratique. N'importe qui peut détruire (i.e. effectuer des tests de pénétration) - cela ne prend qu'un master pour construire des logiciels sécurisés. Aspirez à devenir un maître.

- Envisagez d'adhérer à l'OWASP et d'assister à des reunions avec les équipes locales (i.e. « Local Chapter ». Le *French Chapter* est l'équipe officielle représentant l'OWASP en France)
- **Demandez des formations de programmation sécurisée** si vous avez un budget formation. Demandez un budget formation si vous n'en avez pas
- **Concevez vos dispositifs de façon sécurisée** - considérez la défense en profondeur et la simplicité dans le design
- **Adopter les standards de codage** qui encouragent les conceptions de code plus sûrs
- **Refactorisez le code existant pour utiliser des conceptions plus sûres** dans vos plateformes choisies, comme des requêtes paramétrées
- **Passer en revue** le Guide de l'OWASP **et commencer à appliquer des contrôles spécifiques à votre code.** À la différence de la plupart des guides de sécurité, il est conçu pour vous aider à concevoir un logiciel sécurisé, pas pour le casser
- **Testez votre code pour identifier tout défaut de sécurité** et faites en sorte que cette étape fasse partie intégrante de votre procédure de test
- **Consultez les ouvrages de références**, et regardez si l'un d'entre-eux est applicable à votre environnement

POUR LES PROJETS OPEN SOURCE

L'Open Source constitue un challenge particulier pour la sécurité des applications web. Il y a littéralement des millions de projets open source, projets personnels d'un développeur à des projets majeurs tels Apache, Tomcat, et des applications web à grande échelle, telle que PostNuke.

- Envisagez d'adhérer à l'OWASP et d'assister à des reunions avec les équipes locales

- Si votre projet comprend plus de 4 développeurs, **envisagez de faire en sorte qu'au moins l'un d'entre eux devienne la personne de référence Sécurité**

- **Concevez vos dispositifs de façon sécurisée** - considérez la notion de défense en profondeur et la simplicité dans le design

- **Adopter les standards de codage qui encouragent les conceptions de code plus sûrs**

- **Adopter une politique de divulgation responsable** afin de veiller à ce que les défauts et problèmes de sécurité sont correctement traités
- **Consultez les ouvrages de références**, et regardez si l'un d'entre-eux est applicable à votre environnement

POUR LES PROPRIETAIRES D'APPLICATION

Les propriétaires d'application dans les implémentations commerciales sont souvent soumis à des contraintes de temps et de ressources. Les propriétaires d'application devraient:

- Travailler sur l'OWASP Secure Software Contract Annex avec les fournisseurs de logiciels

- **Assurez-vous que les pré-requis business incluent des exigences non-fonctionnelles (i.e. NFRs, Non-Functional Requirements) telles que les pré-requis de sécurité**

- **Encouragez les conceptions qui incluent des dispositifs sécurisés par défaut**, défense en profondeur et simplicité dans le design

- **Employez (ou formez) des développeurs qui ont une réelle / forte expérience sécurité**

- **Testez les défauts de sécurité** tout au long du projet: conception, construction, test, et déploiement

- **Allouez des ressources, du budget et du temps dans le plan projet pour remédier aux problèmes de sécurité**

POUR LES CADRES DIRIGEANTS

Votre entreprise doit avoir un cycle de vie de développement sécurisé (SDLC, Secure Development Life Cycle) en place qui convienne à votre organisation. La réparation des vulnérabilités coûte beaucoup moins cher en phase de développement plutôt qu'une fois les produits vendus. Un SDLC raisonnable n'est pas seulement composé des tests du Top 10, il inclue:

- Pour les logiciels sur étagère, **assurez-vous que les politiques d'achat et les contrats comportent des exigences de sécurité**
- Pour du code personnalisé, **adopter des principes de codage sécurisé dans vos politiques et standards**
- **Former vos développeurs aux techniques de programmation sécurisée** et veiller à ce qu'ils maintiennent ces compétences à jour
- **Incluez des outils d'analyse de code appropriés de sécurité dans votre budget**
- **Notifiez vos fournisseurs logiciels de l'importance de la sécurité pour votre résultat final**
- **Formez vos architectes, designers, et commerciaux aux fondamentaux de la sécurité des applications web**
- **Considérez l'utilisation d'auditeurs de code tiers**, qui peuvent fournir une évaluation indépendante
- **Adopter des pratiques de divulgation responsables** et mettez un processus en place pour répondre correctement aux rapports de vulnérabilités relatifs à vos produits

REFERENCES

PROJETS OWASP

L'OWASP est le premier site pour la sécurité des applications web. Le site de l' OWASP héberge beaucoup de projets, forums, blogs, présentations, outils, et publications. L' OWASP héberge deux conférences de sécurité des applications web majeures par an, et dispose de plus de 80 chapitres locaux.

Les projets OWASP suivants sont les plus susceptibles d'être utiles:

- OWASP Guide to Building Secure Web Applications
- OWASP Testing Guide
- OWASP Code Review Project (en développement)
- OWASP PHP Project (en développement)
- OWASP Java Project
- OWASP .NET Project

OUVRAGES

Utilisez ces références pour trouver le secteur approprié dans votre librairie locale et sélectionner quelques titres (y compris potentiellement un ou plusieurs de ceux qui suivent) qui répondent à vos besoins:

- [ALS1] Alshanetsky, I. "*phplarchitect's Guide to PHP Security*", ISBN 0973862106
- [BAI1] Baier, D., "*Developing more secure ASP.NET 2.0 Applications*", ISBN 978-0-7356-2331-6
- [GAL1] Gallagher T., Landauer L., Jeffries B., "*Hunting Security Bugs*", Microsoft Press, ISBN 073562187X
- [GRO1] Fogie, Grossman, Hansen, Rager, "*Cross Site Scripting Attacks: XSS Exploits and Defense*", ISBN 1597491543
- [HOW1] Howard M., Lipner S., "*The Security Development Lifecycle*", Microsoft Press, ISBN 0735622140
- [SCH1] Schneier B., "*Practical Cryptography*", Wiley, ISBN 047122894X
- [SHI1] Shiflett, C., "*Essential PHP Security*", ISBN 059600656X
- [WYS1] Wysopal et al, *The Art of Software Security Testing: Identifying Software Security Flaws*, ISBN 0321304861

OWASP Top 10 2007

SITES WEB

- OWASP, http://www.owasp.org
- MITRE, Énumération des Faiblesses Communes – Tendances de Vulnérabilité, http://cwe.mitre.org/documents/vuln-trends.html
- Web Application Security Consortium, http://www.webappsec.org/
- Top 20 du SANS Institute, http://www.sans.org/top20/
- Conseil des Standards Sécurité PCI, éditeurs des standards PCI, appropriés à tous les organismes pour le traitement ou l'exploitation des données de carte de crédit, https://www.pcisecuritystandards.org/
- PCI DSS v1.1, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Build Security In, US CERT, https://buildsecurityin.us-cert.gov/daisy/bsi/home.html