Les différentes façon de tester ses applications

Au minimum, le test de sécurité des applications Web nécessite l'utilisation d'un analyseur de vulnérabilité, tel que Netsparker ou Acunetix Web Vulnerability Scanner.
Pour les tests authentifiés, un proxy HTTP tel que Burp Suite permet d'essayer de manipuler les connexions des utilisateurs, la gestion des sessions, les workflows des applications, etc.

Validation des tests et vérifications manuelles supplémentaires

Il n'est pas possible de lister exhaustivement tous les contrôles à effectuer, tant les vecteurs d'attaque peuvent être nombreux et diversifiés. La première chose à faire consiste à valider les résultats du scanner de vulnérabilités Web pour voir ce qui est exploitable et ce qui compte dans le contexte de son application et de son entreprise. Mais au-delà, d'autres domaines peuvent mériter un examen :

le mécanisme de connexion et les manipulations de session impliquant des mots de passe, des cookies et des jetons,
l'impact de failles liées à l'utilisateur ou au navigateur Web,
les faiblesses de la logique applicative qui permettent la manipulation manuelle du déroulement des opérations et des champs de saisie spécifiques,
et la politique de gestion des mots de passe, qu'il s'agisse de leur complexité ou de la gestion des compromissions.

Une partie des tests consiste à effectuer des scans de vulnérabilité. Ces scans utilisent des attaques typiques basées sur les dix principaux risques de sécurité de l'OWASP et les vulnérabilités connues des composants du framework utilisé.
Ils permettront d'identifier rapidement une vulnérabilité commune ou une mauvaise configuration.

Le fuzzing

Le fuzzing est une technique utilisée pour tester les zones où les données sont entrées sur votre application web. Il teste les entrées aléatoirement pour s'assurer que les techniques de validation appropriées sont utilisées. Le fuzzing va permettre de tester l'injection SQL et les attaques XSS, par exemple !

L'audit de code

L'audit de code peut être effectué manuellement ou à l'aide d'un outil automatisé.
Il implique souvent une analyse ligne par ligne du code source pour trouver les éléments ou fonctions vulnérables.

Ce test garantit que les fonctions vulnérables communes ne sont pas utilisées dans le code. Cela peut permettre de détecter des injections SQL, par exemple.

Threat Modeling

Un modèle de menace est un processus par lequel des menaces potentielles, telles que les vulnérabilités structurelles (c'est-à-dire inhérentes à la structure de votre application) peuvent être identifiées, énumérées et classées par ordre de priorité.

Voici quelques questions que vous pouvez poser pour créer votre modèle de menace :

Quelles sont les données que vous devez protéger ?

Qui exploitera les données ?

Quelle est la probabilité qu'une vulnérabilité soit exploitée ?

Quels dommages cette exploitation peut-elle causer ?

Quelle est votre protection contre cet exploit ?

Les failles web les plus connues

Voici les dix erreurs-type dénoncées par l'OWASP :

1. **Oubli de valider les entrées des utilisateurs.**
Un classique, qui permet aux pirates de faire accepter des commandes au serveur à travers un formulaire web ou une simple URL, ou d'exécuter des contenus dynamiques (Javascript, par exemple) chez les autres utilisateurs d'un site.

2. **Contrôle d'accès inefficace.**
Mauvaise mise en oeuvre des outils de contrôle d'accès (fichier .htpasswd lisible par tous, mots de passes nuls par défaut, etc...).

3. **Mauvaise gestion des sessions.**
Cela permet aux pirates de "voler des sessions" d'autres utilisateurs (en devinant un numéro de session simple, en dérobant un cookie, ou en allant regarder les fichiers de sessions de PHP).

4. **Cross Site Scripting**.
Un autre grand classique, lui aussi lié à un manque de contrôle des entrées de l'utilisateur. Cette faille touche les sites web qui laissent les internautes publier du code HTML susceptible d'être vu par les autres utilisateurs du site (dans un forum, par exemple). Cela permet d'exécuter des contenus dynamiques sur les navigateurs des internautes, avec les droits associés au site web.

5. **Dépassement de mémoire tampon.**
Une faille vieille comme le monde, qui frappe certains langages de programmation plus que d'autres (le C, par exemple). Si des composants CGI sont (mal) écrits dans ces langages, il peut-être simple de compromettre totalement le serveur par une telle attaque.

6. **Injection de commandes.**
Là encore, la source de la faille est un manque de contrôle des entrées de l'utilisateur. Elle permet au pirate de faire exécuter des commandes au serveur (au système d'exploitation ou à un serveur SQL, par exemple) en les attachant à une entrée web légitime avant que celle-ci ne soit transmise au serveur.

7. **Mauvaise gestion des erreurs.**
Les messages d'erreur utiles aux développeurs le sont souvent aussi pour les pirates ! Il faut donc penser à les supprimer une fois le développement terminé.

8. **Mauvaise utilisation du chiffrement.**
La mise en oeuvre du chiffrement au sein des applications web se révèle ardue. Des développeurs non spécialisés peuvent commettre des erreurs difficiles à déceler et créer ainsi une protection illusoire.

9. **Failles dans l'administration distante.**
C'est la voie royale : si les pages réservées aux administrateurs du site ne sont pas réellement protégées (authentification forte du client, chiffrement, contrôles réguliers...), un pirate peut prendre le contrôle du site sans avoir à pirater le serveur. Une aubaine, en quelque sorte.

10. **Mauvaise configuration du serveur web et des applications.**
Un classique : le serveur web qui permet de lister n'importe quel répertoire, ou les outils de développement qui laissent des versions temporaires des fichiers, lisibles par tout le monde. Avant de mettre un serveur en ligne, il est bon de faire le ménage et de bien comprendre toutes les options de ses fichiers de configuration...

Exploiter et se protéger des failles web connues

Ci-dessous vous aurez accès à des exemples et des solutions pour chacune des dix erreurs type énumérées ci-dessus !

1.

Exemple d’une page qui affiche la page d'accueil ou l'espace admin en fonction du paramètre admin :

http://www.abc.fr/index.php?admin=1

exemple un peu plus répandu :

http://www.abc.fr/images.php?dir=images_2020

Cette page est censée lister les images du dossier images_2020. Or ce chemin est passé en paramètre : l'utilisateur peut alors tout à fait le modifier pour en venir à :

http://www.abc.fr/images.php?dir=../admin/

Il aura alors la liste de vos fichiers d'administration et en fonction du type de l'affichage, il pourra peut-être même voir leurs contenus (mot de passe de la base de données, ...).

Solution

Pour accéder à votre espace d'administration, faites un formulaire de connexion *POST* puis utiliser les sessions plutôt que d'ajouter un paramètres dans l'*URL*.

2.

Exemple et mise en évidence d’un contrôle d’accès inefficace

Supposons que nous disposons d’un serveur Apache local et que nous avons une page index.php très simple comme ceci :
<h1>Bonjour !</h1>

Pour accéder à la page, nous accédons à l’URL http://127.0.0.1/~trance/vuln/ avec un navigateur Web. La requête envoyée est grossièrement la suivante :

GET http://127.0.0.1/~trance/vuln/ HTTP/1.1
Host: 127.0.0.1

La réponse reçue comporte alors le code HTML de la page Web. Jusqu’ici, tout est normal : Apache n’a fait qu’interpréter la requête GET qu’on lui a envoyé. Maintenant, imaginez que nous envoyons une requête mal formée qui n’est pas de type GET, comme :

n1Mp0rTeKwa http://127.0.0.1/~trance/vuln/ HTTP/1.1
Host: 127.0.0.1

Et là, nous constatons qu'Apache ne fait absolument aucune différence entre cette et la requête précédente ! Il n'y a aucune erreur ; il renvoie la même chose. A première vue, cela ne choque pas forcément, mais on s'aperçoit assez vite que l'on peut tirer profit de cette faille pour contourner certaines protections .htaccess.

Solution

On peut se prémunir de ce type de faille par exemple en n'utilisant pas les closes <Limit>.

3.

Exemple d'une mauvaise gestion de session

http://exemple.com/sale/saleitems;masessionid=45131215=Hawaii

Solution :

Utiliser un framework éprouvé qui gère correctement les sessions, et ne développer ou tester que si réellement nécessaire.

4.

Exemple d'un Cross-Site Scripting (XSS)

Pirate

(1) - Le pirate injecte du code nocif dans un site vulnérable

Victime

(2) - La victime visite la page piégée du site web ciblé
(3) - La page demandée contenant le code injecté par le hacker est affichée au client.
(4) - Le code est exécuté.
(5) - Les données sont envoyées.

Solution

Il faut absolument utiliser les fonctions php `htmlspecialchars()` qui filtre les '<' et '>' ou `htmlentities()` qui filtre toutes les entités html.

Ces fonctions doivent être utilisées sur des entrées utilisateurs qui s'afficheront plus tard sur votre site. Si elle ne sont pas filtrées, les scripts comme ceux que nous avons vus plus haut s'exécuteront avec tout le mal qui s'en suit.

Voici un exemple d'utilisation de cette fonction :

<?php echo htmlspecialchars($_POST['nom']); // echo affiche les données sur un page, du coup on protège l'affichage avec la fonction htmlspecialchars?>
Au possible, il faut placer des cookies avec le paramètre **HttpOnly**, empêchant leur récupération avec JavaScript (Attention elle n'est pas forcément supportée par tous les navigateurs).

5.

Solution au buffer overflow

Pour se prémunir contre de telles attaques, plusieurs options sont offertes au programmeur. Quelques-unes de ces options sont décrites dans les deux sections suivantes.

Protections logicielles

- Bannir de son utilisation les fonctions dites « non protégées ». Préférer par exemple strncpy à strcpy ou alors fgets à scanf qui effectue un contrôle de taille. Les compilateurs récents peuvent prévenir le programmeur s'il utilise des fonctions à risque, même si leur utilisation demeure possible.
- Utiliser des options de compilation permettant d'éviter les dépassements de mémoire tampon menant à la corruption de pile, comme le SSP (Stack-Smashing Protector).
- Se reposer sur les protections offertes par le système d'exploitation, comme l'Address space layout randomization ou la Data Execution Prevention.
- Utiliser des outils externes pendant le développement pour détecter les accès mémoire invalides à l'exécution, par exemple la bibliothèque *Electric Fence* ou Valgrind.
- Utilisation de langages à contexte d'exécution managé implémentant la vérification de bornes des tableaux.

Technique du canari

La technique du canari fait référence aux canaris qui étaient utilisés jadis pour détecter les fuites de grisou dans les mines de charbon3.

Le principe est de stocker une valeur secrète, générée à l'exécution, juste avant l'adresse de retour. Lorsque la fonction se termine, cette valeur est contrôlée. Cette méthode permet d'empêcher l'exécution de code corrompu, mais augmente la taille du code et ralentit donc l'appel de la fonction.

Protections matérielles

- Les microprocesseurs récents, 64 bits notamment, implémentent des protections efficaces.

6.

Exemple Injection SQL

```
uname = request.POST['username']
passwd = request.POST['password']

sql = "SELECT id FROM users WHERE username='" + uname + "' AND
password='" + passwd + "'"

database.execute(sql)
```

Un attaquant a maintenant la possibilité de **manipuler le champ relatif au mot de passe** grâce à une injection SQL, en entrant par exemple *password' OR 1='1*, ce qui mène à la requête SQL suivante :

```
sql = "SELECT id FROM users WHERE username='' AND password='password'
OR 1='1'
```

En agissant ainsi, il peut accéder à l'ensemble des tables utilisateurs de la base de données, **le mot de passe étant toujours valide** (1='1'). S'il se **connecte en tant qu'administrateur**, il peut procéder à toutes les modifications qu'il désire. Autrement, c'est le champ « nom d'utilisateur » qui peut être également manipulé de cette manière.

Solution

Pour se prémunir des injections SQL vous pouvez passer par plusieurs étapes.

Etape 1 : superviser la saisie automatique des applications

Etape 2 : s'assurer de la protection complète du serveur.

Etape 3 : renforcer les bases de données et utiliser des codes sûrs.

7.

Solution a la mauvaise gestion des erreurs

Il faut utiliser display_error = off pour ne pas afficher les erreurs à l'utilisateur.

8.

Solution a la mauvaise utilisation du chiffrement

Faciliter l'utilisation des bibliothèques de chiffrement, mais également des développeurs qui doivent implémenter un chiffrement dans leurs applications qui se renseignent convenablemen.

9.

Solution faille dans l'administration distante

Faire un espace administrateur en béton.

10

Solution a la mauvaise configuration d'un serveur web et des applications

Eviter les répertoires et les fichiers temporaires, car ce sont des répertoires accessibles en lecture à tous.

Les sources à suivre pour rester informé sur les nouvelles failles

Accédez ci-dessous a de multiples sources pour rester informé sur les nouvelles failles !

CVE (Common Vulnerabilities and Exposures) est la référence mondiale en matière de failles et vulnérabilités. Malheureusement, elle n'est pas nécessairement la plus agréable en ce qui concerne la navigation et la recherche.

Vigilance.fr : est l'une des sources francophones les plus utiles, avec des bulletins technologiques en français, une base assez complète et plusieurs flux RSS pour suivre les publications plus facilement.

CERT-FR, le centre gouvernemental de veille, d'alerte et de réponse aux attaques informatiques, qui dépend de l'ANSSI, est la base de données officielle française autour des vulnérabilités. C'est aussi une source indispensable de bonnes pratiques.

La base de données Security Focus est l'une des plus complètes qui soient. Sa fonction de recherche avancée permet de sélectionner un éditeur, puis un logiciel et enfin un numéro de version et d'afficher toutes les vulnérabilités connues s'y rapportant. En cliquant sur l'une des occurrences remontées, vous pouvez connaître non seulement si celle-ci est exploitée (autrement dit si des exploits sont en circulation) et les solutions à implémenter pour s'en protéger.

Exploit-db.com est le site de référence pour tout savoir sur les exploits en vogue pour pénétrer les défenses des entreprises. Le site recueille et regroupe tous les exploits rencontrés sur Internet (« In The Wild ») ou découverts par des chercheurs du monde entier.

Le NVD (National Vulnerability Database) est la base d'information officielle du gouvernement américain avec des fonctions de recherche avancée et une mise en évidence des failles les plus critiques.

L'Enisa (European Union Agency for Network and Information Security) est avant tout une source de bonnes pratiques et de conseils pour protéger ses systèmes d'information.

Information Security Stack Exchange est un site assez différent du reste de notre sélection. C'est en premier lieu une base de questions et réponses pour les professionnels de la sécurité.

Source capitale pour toutes les entreprises dont l'activité dépend en partie de Windows et Office, l'ancienne base de bulletins de sécurité de Microsoft a récemment été modernisée et regroupée au sein du Security TechCenter. Son Security Update Guide s'affirme comme l'une des sources d'information les plus claires et les plus lisibles autour des failles de l'univers Microsoft.

Le blog du National CyberSecurity Institute est certes en anglais, mais il offre une vue pédagogique plutôt que technique sur toutes les attaques du moment.

LiveHacking est la référence du Hacking éthique. C'est une mine de données et d'informations présentées sous un axe très hacker.

L’Infosec Institute est un site toujours pratique pour suivre les attaques les plus répandues du moment.

Le blog de MalwareBytes est également une source d’information essentielle pour tous les acteurs de la sécurité et les RSSI. Il publie notamment chaque semaine un billet « A week in security » qui résume toute l’actualité en matière de malwares et d’exploits. L’information à forte teneur pédagogique est accessible au plus grand nombre. Particulièrement adapté pour tous ceux qui n’ont ni le besoin, ni le temps d’investiguer le sujet au quotidien, ce blog permet d’avoir facilement une vue d’ensemble sur les menaces phares du moment.