APPLICATION WEB & cyber-sécurité

Histoire :

En 2012, la société Yahoo a été piratée. À cause d'une mauvaise configuration de la sécurité des bases de données, les mots de passe des utilisateurs ont été rendus publics.

Testez votre adresse mail (intrusion)

https://haveibeenpwned.com/

80 % des attaques sur les applications web sont dues à de mauvaises pratiques de développement, qui sont à l'origine des vulnérabilités découvertes et exploitées.

De plus en plus d'entreprises utilisent des applications web pour traiter des données privées, telles que des informations personnelles (nom, date de naissance, numéro d'identification national, etc.) ou des données bancaires. Toutes ces données représentent un attrait considérable pour un attaquant.

Une fuite de données due à un piratage a un impact majeur sur l'image d'une entreprise. Ces dommages se matérialisent bien souvent par des pertes financière

Remarques :

Les atteintes à la protection des données sur les applications web sont courantes et se produisent même dans les grandes entreprises.

Une attaque sur une application web peut faire perdre à une entreprise beaucoup d'argent et sa réputation.

Le risque est mesuré en comparant la valeur des données au coût de leur sécurisation.

Il existe trois principes de sécurité qui peuvent être appliqués pour garantir la sécurité d'une application ou d'une infrastructure.

La confidentialité. C'est l'assurance que les personnes non autorisées n'accèdent pas à des informations sensibles.

L'intégrité. Elle permet d'être sûr que les données sont fiables et n'ont pas été modifiées par des personnes non autorisées.

La disponibilité. C'est l'assurance qu'il n'y a pas de perturbation d'un service ou de l'accessibilité aux données.

La sécurité de l'information repose sur l'équilibre entre ces trois principes.

Ces principes fondamentaux sont des éléments clés dans l'élaboration des politiques de sécurité en entreprise. Mais il existe également des réglementations imposées pour certains secteurs d'activités qui influencent également les politiques de sécurité des entreprises.

Rappels :

Le RGPD garantit la confidentialité des données pour les résidents de l'UE.

Le traitements des données de santé est réglementé par la CNIL et nécessite une prise en charge particulière.

Exemple : Norme PCI DSS

La norme Payment Card Industry Data Security Standard (PCI DSS) est une norme établie pour toutes les entreprises qui traitent des données bancaires. La sécurisation des données bancaires met l'accent sur la sécurité lors de la transmission, du traitement et du stockage des données.

De plus en plus de plateformes web et de solutions de stockage gèrent les applications pour les entreprises, il est essentiel que le fournisseur soit également conforme à la norme PCI DSS.

Si vous créez une application web qui traite des données bancaires, comme une boutique en ligne ou un site web par abonnement, vous devez chiffrer les transmissions pour garantir la sécurité des données en transit. Les données bancaires transmises en texte clair constituent une violation de la norme PCI DSS, ce qui peut faire l'objet d'une amende.

Découvrez l'Open Web Application Security Project (OWASP)

Il existe une multitude d'attaques permettant de compromettre un système. Nous allons nous concentrer sur les attaques ciblant les applications web et sur la façon de les sécuriser.

Le principe d'une attaque web

Les attaques web tirent parti des failles de sécurité de vos applications web. On parle également de vulnérabilités. Le Top Ten de l'OWASP est un document proposant une classification des attaques web les plus courantes ainsi que les contre-mesures associées.

Lors du développement d'applications, les développeurs ne sont pas forcément au fait des mesures de sécurité à appliquer, et la sécurité représente souvent une contrainte budgétaire et opérationnelle.

C'est pour cette raison qu'il est important de prendre en compte la sécurité dès les premiers développements. En effet, une prise en compte tardive de la sécurité pourra représenter un coût supplémentaire, sans parler du risque encouru en cas d'attaque.

Découvrez l'OWASP

L'Open Web Application Security Project (OWASP) est une organisation à but non lucratif fondée en 2004 pour prévenir de manière proactive les attaques sur les applications web. Il s'agit du premier effort de normalisation des pratiques de développement sécurisé.

En 2001, l'OWASP n'était pas une organisation officielle, mais plutôt un collectif qui préconisait des pratiques de développement sécurisé.

Ce collectif a pris de l'ampleur et est devenu l'OWASP foundation en 2004, avec une norme éthique pour maintenir une neutralité et l'absence de pressions commerciales.

L'OWASP n'est réglementée par aucune entreprise. Elle propose un référentiel neutre permettant d'accompagner les entreprises dans le processus de sécurisation ou d'audit de sécurité.

Avec l'évolution d'Internet, les plus grandes entreprises ont une présence significative en ligne, ce qui les oblige à fournir et à maintenir des applications web.

La plupart des processus internes utilisent également des applications web. Toute application qui traite des données sensibles peut être attaquée et nécessite donc une sécurité standardisée.

Les entreprises qui ne traitent pas de données sensibles, mais qui ont besoin d'une plateforme web pour leur business, sont également exposées aux attaques web.

Par exemple, une entreprise qui fournit des informations, comme un site d'actualité, peut ne pas avoir besoin des données personnelles de ses clients mais nécessite que le site web soit disponible pour permettre la continuité de service. Si une attaque faisait s'effondrer le site, cela aurait une incidence sur les revenus de l'entreprise.

Le Top Ten de l'OWASP fournit une base de référence avec une liste de contrôles à effectuer pour atténuer les risques les plus courants en matière de sécurité. Cette base de référence est également utilisée pour répondre à des normes réglementaires plus strictes, telles que le RGPD par exemple.

Dans le Top Ten de l'OWASP, les attaques sont classées par ordre d'importance. Les vulnérabilités présentes dans le Top Ten existent dans les applications web de tous les langages et frameworks. Il n'existe pas de langage de programmation exempt de vulnérabilités.

Voici les 10 attaques les plus fréquentes contre les applications web

1 - Injection

Une attaque par injection est une attaque permettant l'injection de code arbitraire dans l'application. Cela se produit lorsque des données non maîtrisées sont exécutées par le

moteur présent sur le backend de l'application. Les données de l'attaquant sont exécutées sans autorisations adéquates.

L'injection de code non autorisée peut permettre à un attaquant d'accéder à des données auxquelles il n'a normalement pas accès.

2 - Piratage de session

Beaucoup d'applications exigent qu'un utilisateur se connecte pour arriver sur des pages auxquelles lui seul a accès. L'application est vulnérable à une attaque si un utilisateur malveillant peut obtenir un accès non autorisé aux mots de passe, clés et jetons pour pirater la session d'un autre utilisateur.

3 - Exposition de données sensibles

Les données stockées ou échangées via une application doivent être protégées pour éviter l'interception par une personne malveillante. Les bases de données qui enregistrent les données personnelles, les données de cartes de crédit, les noms d'utilisateur et les mots de passe représentent une cible de choix pour un pirate. Si ces données ne sont pas chiffrées, elles ne sont pas sécurisées et peuvent être récupérées lorsqu'elles sont en transit par exemple. L'utilisation de techniques de chiffrement et de pratiques de sécurité peut atténuer ce type d'attaques.

4 - Entités externes XML (XXE)

Le format XML permet de faciliter l'échange de données sous forme d'arborescence. Il est largement utilisé sur Internet. Il peut être exploité via l'injection XXE ou XML External Entity. XML External Entity est une attaque contre les applications qui parsent des entrées XML (exemple flux RSS). Cette attaque a lieu lorsque l'analyseur XML est mal configuré et contient une référence à une entité externe.

5 - Contournement du contrôle d'accès.

Cette attaque vise les fonctionnalités des applications web qui nécessitent un contrôle d'accès. Dans ce cas, les pirates peuvent utiliser l'URL pour contourner l'authentification, par exemple. Ce type d'exploitation peut par exemple révéler comment une base de données est organisée.

6 - Security Misconfiguration

Une mauvaise configuration de sécurité est le plus souvent observée dans les en-têtes HTTP qui permettent de donner des indications sur la configuration du serveur, ou via la gestion des exceptions par défaut. Les codes d'erreur et les exceptions courantes peuvent donner à un attaquant un aperçu de l'application.

7 - Cross-Site Scripting (XSS)

Les failles XSS se produisent chaque fois qu'une application inclut des données non fiables dans une nouvelle page web sans validation ou échappement. Les failles XSS permettent aux attaquants d'exécuter des scripts dans le navigateur de la victime, ce qui peut détourner des sessions utilisateur, altérer des sites web ou rediriger l'utilisateur vers un site malveillant.

8 - Désérialisation non sécurisée (Insecure Deserialisation)

Une vulnérabilité de type “insecure deserialisation” permet à un utilisateur malveillant d'accéder et de modifier les fonctionnalités de l’application ciblée.

9 - Utilisation de composants présentant des vulnérabilités connues

Même si votre application est sécurisée, vous devez vous assurer que le framework, les bibliothèques, les appels API et la plateforme que vous utilisez ne sont pas vulnérables.

Lorsqu'une nouvelle vulnérabilité est découverte, un correctif est généralement proposé. Il faudra alors l’appliquer pour garantir la sécurité de l’application.

10 - Manque de surveillance et de monitoring

Pour garantir la sécurité d’une application, il est nécessaire de surveiller et de monitorer les connexions. De nombreux serveurs vulnérables servent de rebond aux attaquants. La mise en place de monitoring permettra de détecter une anomalie sur le serveur.

Notons que 80 % des applications présentent des vulnérabilités communes telles que décrites dans l'OWASP.

Injection

Cette vulnérabilité permet à un attaquant d’injecter des données non maîtrisées qui seront exécutées par l’application et qui permettent d’effectuer des actions qui ne sont normalement pas autorisées.

Ces injections peuvent par exemple être des requêtes SQL pour manipuler la base de données, du code JavaScript ou HTML.

À quel endroit un attaquant effectue une injection ?

Dans une application web, il existe des champs ou des formulaires. Ce sont des composants des applications permettant la saisie de l’utilisateur. Ces champs sont généralement la cible d’injections. Ils se présentent sous la forme d'un formulaire HTML ou d'un champ texte. Ce que vous avez tapé est ensuite traité. Si vous tapez un nom d'utilisateur et un mot de passe, ils seront alors envoyés à la base de données pour pouvoir vous

Le type d'attaque par injection le plus connu est l'injection SQL.

La faille SQLi, abréviation de SQL Injection, soit injection SQL en français, est un groupe de méthodes d'exploitation de faille de sécurité d'une application interagissant avec une base

de données. Elle permet d'injecter dans la requête SQL en cours un morceau de requête non prévu par le système et pouvant en compromettre la sécurité.

Exemple

Considérons un site web dynamique qui dispose d'un système permettant aux utilisateurs possédant un nom d'utilisateur et un mot de passe valides de se connecter. Ce site utilise la requête SQL suivante pour identifier un utilisateur :

SELECT uid FROM Users WHERE name = '(nom)' AND password = '(mot de passe hashé)';

L'utilisateur Dupont souhaite se connecter avec son mot de passe « truc » hashé en MD5. La requête suivante est exécutée :

SELECT uid FROM Users WHERE name = 'Dupont' AND password = '45723a2af3788c4ff17f8d1114760e62';

Attaquer la requête

Imaginons à présent que le script exécutant cette requête ne vérifie pas les données entrantes pour garantir sa sécurité. Un hacker pourrait alors fournir les informations suivantes :

Utilisateur : Dupont';--

Mot de passe : n'importe lequel

La requête devient :

SELECT uid FROM Users WHERE name = 'Dupont'; -- ' AND password = '4e383a1918b432a9bb7702f086c56596e';

Les caractères -- marquent le début d'un commentaire en SQL. La requête est donc équivalente à :

SELECT uid FROM Users WHERE name = 'Dupont';

L'attaquant peut alors se connecter sous l'utilisateur Dupont avec n'importe quel mot de passe. Il s'agit d'une injection de SQL réussie, car l'attaquant est parvenu à injecter les caractères qu'il voulait pour modifier le comportement de la requête.

Supposons maintenant que l'attaquant veuille non pas tromper le script SQL sur le nom d'utilisateur, mais sur le mot de passe. Il pourra alors injecter le code suivant :

Utilisateur : Dupont

Mot de passe : ' or 1 --

L'apostrophe indique la fin de la zone de frappe de l'utilisateur, le code « or 1 » demande au

script si 1 est vrai, or c'est toujours le cas, et -- indique le début d'un commentaire.

La requête devient alors :

SELECT uid FROM Users WHERE name = 'Dupont' AND password = '' or 1 --';

Ainsi, le script programmé pour vérifier si ce que l'utilisateur tape est vrai, il verra que 1 est vrai, et l'attaquant sera connecté sous la session Dupont.

Sécurisez une application contre l'injection SQL

Pour garantir une protection contre l'injection SQL, il est possible d'utiliser un pare-feu d'application web ou WAF, pour Web Application Firewall en anglais. Ce pare-feu se place entre l'utilisateur et l'application web et permet de vérifier et d'intercepter les données envoyées. Toutefois il est également possible de sécuriser l'application directement dans le code.

Validez les entrées

La validation des entrées est une excellente pratique en tant que développeur web. Elle limite ce que l'utilisateur peut mettre dans la zone de texte.

Cela n'empêchera pas l'injection, mais c'est une mesure que vous pouvez mettre en place pour limiter des attaques de base.

Utilisez une requête préparée !

Ecrire vos requêtes SQL en paramétrant les variables. C'est ce qu'on appelle une requête préparée.

```
select * from personne where num = ?
```

La variable num est paramétrée, elle est donc désormais passée dans une méthode pour exécuter la requête au lieu de se connecter directement à la base de données.

```
ps.setInt(1, num);
```

OWASP ZAP et son mode "attaque"

Test des applications web & audits tests d'intrusions

Lors de projets de création d'applications (applications web ou applications mobile smartphones Android iphone IOS) une phase de pentesting doit être intégrée. (bug bounty comme certains l'appellent).

Après les tests unitaires, les tests d'intégration, les tests d'intrusions sont à prendre en

comptes.

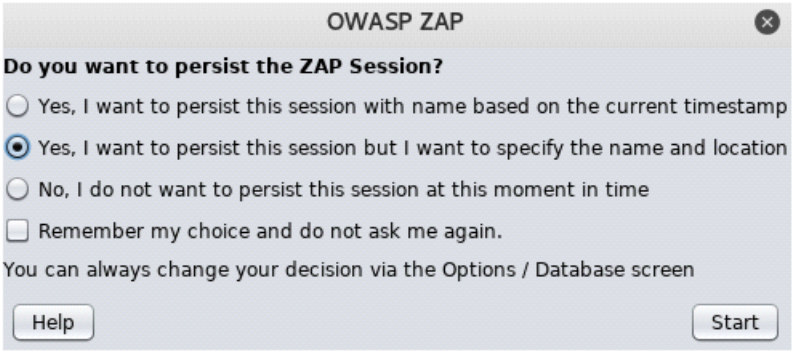
Ces tests d'intrusions vont nous permettre de trouver les failles et les vulnérabilités du programme, avant que le programme passe en production. Cela évite qu'ensuite des hackers mal intentionnés (les black hats, ou chapeaux noirs) trouvent des failles et les exploitent.

il est interessant d'utiliser aussi le logiciel OWASP ZAP pour pentester à chaque étape/jalon du projet d'une application.

OWASP ZAP est un outil Java permettant de tester la sécurité des applications Web. Il possède une interface graphique intuitive et de puissantes fonctionnalités permettant de d'attaquer d'applications Web, ...

Lancement d'OWASP ZAP

On sauvegarde la session avec le nom du site scanné : « ... » dans le répertoire « ... » . Le logiciel crée 5 fichiers d'audits dans ce répertoire

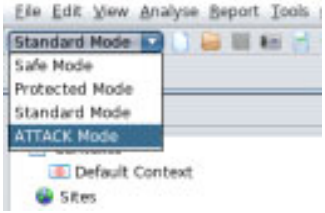


Puis "**start**"

Nous ne testerons que la fonction "Attaque" dans la grande fenêtre de gauche. Dans ce mode, OWASP ZAP se rend de manière agressive sur le site Web que nous avons désigné et commence à rechercher des vulnérabilités.

Selectionner le mode Attaque "Attack mode" d'OWASP ZAP

Nous avons le choix entre safe mode (si le site est en production, pour ne pas le perturber), protected mode, standard mode, ou attack mode (attention, ce mode est très violent et risque de crasher le site en production) .



Rédiger un rapport

Les rapports sont très importants pour les réunions de restitution. Le consultant s'appuie dessus pour transmettre l'information.

Nous pouvons générer des rapports: XML, Markdown, Json

Report/Generate HTML report

On va l'appeler par exemple **NOMDUSITEVISITE**.html

Le rapport généré est responsive design, consultable sur tablette, smartphone

voici les 10 meilleurs outils de tests d'intrusion:

https://www.comparitech.com/net-admin/best-penetration-testing-tools/

Faille XSS

Une faille XSS consiste à injecter du code directement interprétable par le navigateur Web, comme, par exemple, du JavaScript ou du HTML.

Cette attaque ne vise pas directement le site comme le ferait une injection SQL mais concerne plutôt la partie client c'est-à-dire vous (ou plutôt votre navigateur). Ce dernier ne fera aucune différence entre le code du site et celui injecté par le pirate, il va donc l'exécuter sans broncher.

Les possibilités sont nombreuses : redirection vers un autre site, vol de cookies, modification du code HTML de la page, exécution d'exploits contre le navigateur : en bref, tout ce que ces langages de script vous permettent de faire.

Exemple :

```
<!DOCTYPE html>
<html lang="fr">
  <head>
   <meta charset="utf-8" />
  </head>
  <body>
    <h1>Mon super moteur de recherche</h1>
    <form type="get" action="">
      <input type="text" name="keyword" />
```

```
        <input type="submit" value="Rechercher" />
      </form>
   </body>
</html>
```

Ce bout de code HTML représente un champ qui nous servira de petit moteur de recherche.

Si vous essayez de rechercher un mot vous n'obtiendrez aucun résultat, c'est tout à fait normal.

Testons donc ce formulaire avec le mot **test**

Rien à signaler.

Testons donc ce formulaire avec **<h1>test</h1>**

???????

Analysons le code obtenu.

```
<!DOCTYPE html>
<html lang="fr">
   <head>
```

```
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Mon super moteur de recherche</h1>
    Résultat(s) pour le mot-clé : <h1>test</h1>
    <form type="get" action="">
      <input type="text" name="keyword" value="<h1>test</h1>" />
      <input type="submit" value="Rechercher" />
    </form>
  </body>
</html>
```

Et plus particulièrement cette ligne.

Résultat(s) pour le mot-clé : <h1>test</h1>

<h1>test</h1> c'est le mot-clé que nous avons entré dans le formulaire. Le navigateur reçoit le résultat et tombe sur <h1>test</h1>.

Pour lui il s'agit d'une balise HTML, il va donc l'interpréter comme telle d'où le mot test en tant que titre.

Testons avec, par exemple, <h1 style="color:red;"><u>test</u></h1> pour vérifier notre théorie (le mot recherché devrait alors être affiché en tant que titre, souligné et en rouge).

Bingo : nous sommes bien en présence d'une faille XSS car le navigateur interprète le code que nous avons injecté.

Injection de code JavaScript

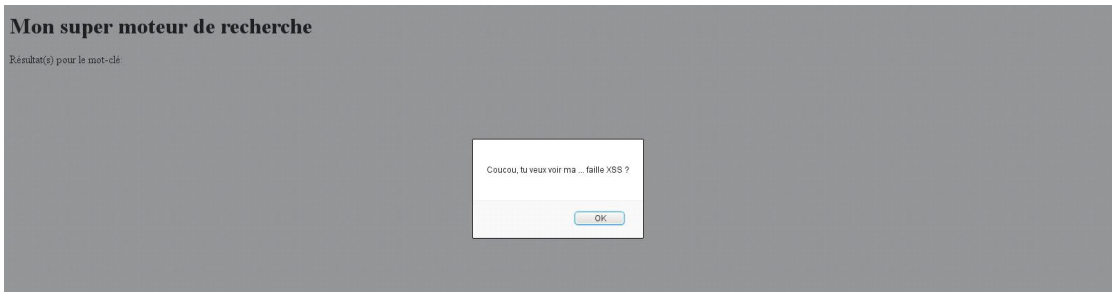
Dans les exemples précédents nous avons injecté du code HTML, ceci dit le HTML n'est pas le seul langage directement compréhensible par un navigateur Web : le JavaScript en est un

autre et va nous permettre de pousser notre exploitation un peu plus loin car le fait est que le HTML est un langage de balisage alors que JavaScript est un vrai langage de programmation, ce qui va nous permettre de faire des choses très intéressantes. ^^

Pour faire comprendre au navigateur que nous voulons qu'il interprète le code comme du JavaScript nous plaçons nos instructions entre la balise <script></script>. Nous verrons aussi par la suite comment on peut éventuellement se passer de cette balise dans le cas où le site vulnérable la filtrerait.

Un cas classique est de tenter d'afficher une fenêtre d'alerte.

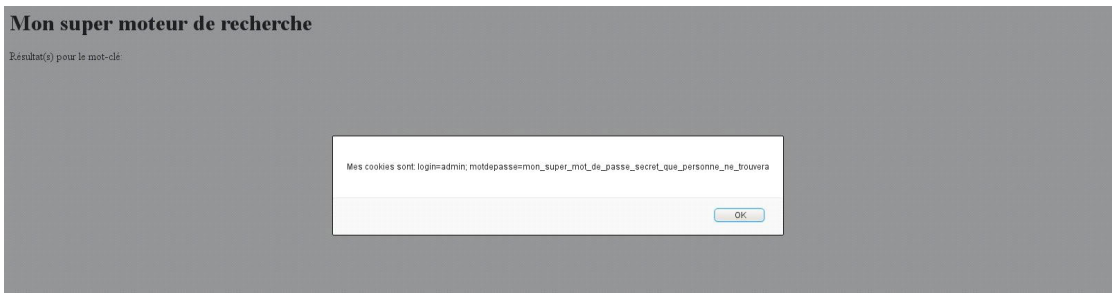
<script>alert("Coucou tu veux voir ma… faille XSS ?");</script>



Etape Suivante :

Affichons nos cookies.

Def : Un cookie est un petit fichier, stocké localement, qui permet de retenir une information (exemple : un login/mot de passe afin d'éviter de devoir chaque fois le retaper).




```
<!DOCTYPE html>

<html lang="fr">

  <head>

    <meta charset="utf-8" />

    <script>

      document.cookie = 'login=admin;'

      document.cookie =
```

```
'motdepasse=mon_super_mot_de_passe_secret_que_personne_ne_trouvera;'
    </script>
  </head>
  <body>
    <h1>Mon super moteur de recherche</h1>
    <?php
    if(!empty($_GET['keyword']))
    {
       echo "Résultat(s) pour le mot-clé : ".$_GET['keyword'];
    }
    ?>
    <form type="get" action="">
       <input type="text" name="keyword" />
       <input type="submit" value="Rechercher" />
    </form>
  </body>
</html>
```

Pour récupérer les cookies, on peut utiliser la fonction **document.cookie.**

<script>window.alert("Mes cookies sont : " + document.cookie);</script>

On affiche nos cookies certes, mais localement… alors comment un pirate peut-il, à distance, les récupérer ?

Il suffit, par exemple, de rediriger la personne vers un site du pirate tout en prenant soin de lui passer les cookies en paramètres. Le site du pirate n'aura plus qu'à les récupérer.

S'en protéger

Pour se protéger des XSS il faut remplacer les caractères qui pourraient éventuellement être compris par le navigateur comme des balises par leur entité HTML.

En faisant cela, le navigateur affichera textuellement le caractère et ne cherchera plus à l'interpréter.

<h1>test</h1> donnera donc le message <h1>test</h1> et non plus le mot « test » en tant

que titre.

En Angular, pour nous en proteger, nous pourrons utiliser les formulaires de type “template driven et la directive ngModel. (Cours Angular)