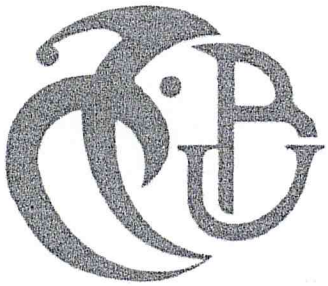
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de L'enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida1

Faculté des Sciences et de la Technologie



Faculté des Sciences

Département d'Informatique

Rapport présenté par:

SAIDI Nassim et REMMIDE Sid Ahmed ElHadi

Thème : Elaboration d'une stratégie de sécurité de l'applicatif de gestion des bases de données.

En vue d'obtenir le diplôme de Master

Domaine: MI

Filière: Informatique

Spécialité : Master en Sécurité des systèmes d'information

Encadreur : Youcef AIT AMARA

Organisme d'accueil : Sonatrach

Promoteur : Fatima BOUMAHDI

Président jury : Ould khaoua Mohamed

Soutenu le : juin 2017

Remerciements

Avant de vous convier à la présentation de ce travail, l'opportunité nous est donnée de témoigner notre gratitude et notre reconnaissance à toutes les personnes qui par leur aide et leurs encouragements nous ont permis de réaliser ce mémoire.

On tient tout d'abord à remercier la Sonatrach en particulier Mr Youcef AIT AMARA qui nous a accueillis dans leur centre durant ces 6 mois, pour son aide précieuse, ses conseils et suggestions et aussi d'avoir durant toute cette année pris de son précieux temps pour nous transmettre les fruits de son expérience.

Nos remerciements s'adressent à tous les enseignants surtout M BOUMAHDI pour tous ses conseils et orientations qui ont contribué à notre formation durant ce stage et les cinq années d'études dans cette université.

On remercie également nos parents pour tous les sacrifices ainsi que leurs soutiens moral et financier.

Nous tenons à remercier tous nos amis et collègues pour leur soutien moral tout au long de la préparation de ce mémoire. Spécialement à tous les étudiants de la promotion sortante sans exception.

Que les membres de ce prestigieux et distingué jury soient assurés de notre gratitude pour nous avoir fait l'honneur d'évaluer notre travail.

Résumé

Avec le développement croissant d'Internet, les applications Web sont devenues de plus en plus vulnérables et exposées à des attaques malveillantes pouvant porter atteinte à des propriétés essentielles telles que la confidentialité, l'intégrité ou la disponibilité des systèmes d'information. Pour faire face à ces malveillances, il est nécessaire de développer des mécanismes de protection et de test qui soient efficaces. La question qui se pose est comment évaluer l'efficacité de tels mécanismes et quels moyens peut-on mettre en œuvre pour analyser leur capacité à détecter correctement des attaques contre les applications web.

Dans ce mémoire nous proposons une approche, basée sur l'analyse des applications web, qui permet d'identifier les vulnérabilités selon une approche de 8 étapes, tout ça en nous exigeant de faire les tests dans le cas d'une boîte noire sur l'application cible. Chaque vulnérabilité identifiée est contrée par une parade ce qui permet de s'assurer que la vulnérabilité est bien arrêtée. L'approche proposée permet également de mettre en évidence différents scénarios d'attaque potentiels incluant l'exploitation de plusieurs vulnérabilités successives en tenant compte explicitement des dépendances entre les vulnérabilités. Cette méthode s'est concrétisée par la mise en œuvre de parades sur chaque vulnérabilité trouvée et a été validée expérimentalement sur notre application web.

Mots-clés : Application Web, Attaque et vulnérabilités, Évaluation, approche

Abstract

With the increasing development of Internet, Web applications have become increasingly vulnerable and exposed to malicious attacks that could affect essential properties such as confidentiality, integrity or availability of information systems. To cope with these threats, it is necessary to develop efficient security protection mechanisms and testing techniques. The question that arises is how to evaluate the effectiveness of such mechanisms and what means can be implemented to analyze their ability to correctly detect attacks against Web applications.

This thesis presents a new approach, based on analyzing web applications, which identify the vulnerabilities of a Web application following a black box analysis of the target application. Each identified vulnerability is actually exploited to ensure that the identified vulnerability

does not correspond to a false positive. The proposed approach can also highlight different potential attack scenarios including the exploitation of several successive vulnerabilities, taking into account explicitly the dependencies between these vulnerabilities. The proposed method led to the development of defense mechanisms and has been validated experimentally on the targeted application.

Table des matières

Liste des tableaux 2

Introduction générale ..... - 1 -

Chapitre I. Les technologies et sécurité Web ..... - 3 -

1. Introduction ..... - 3 -
2. Le protocole HTTP ..... - 3 -
   - 2.1. Les Requête HTTP ..... - 3 -
   - 2.2. Réponses HTTP ..... - 4 -
   - 2.3. Les méthodes HTTP ..... - 5 -
   - 2.4. Les en-têtes HTTP ..... - 6 -
   - 2.5. Cookies ..... - 7 -
   - 2.6. HTTPS ..... - 8 -
   - 2.7. Les proxys HTTP ..... - 9 -
   - 2.8. Authentification HTTP ..... - 9 -
3. Les Fonctionnalités Web ..... - 10 -
   - 3.1. Fonctionnalité côté serveur ..... - 10 -
   - 3.2. Fonctionnalité côté Client ..... - 11 -
   - 3.3. Codage des caractères ..... - 16 -
4. La sécurité des applications web ..... - 18 -
   - 4.1. L'abus de ressources. ..... - 18 -
   - 4.2. La destruction de données ..... - 18 -
   - 4.3. La publication de données confidentielles ..... - 18 -
   - 4.4. Le détournement du site ..... - 18 -
   - 4.5. L'usurpation d'identité ..... - 19 -
5. Conclusion ..... - 19 -

Chapitre II. Les vulnérabilités dans les technologies utilisées pour le développement d'applications web (WAMP/LAMP) ..... - 20 -

1. Introduction ..... - 20 -
2. PHP ..... - 20 -
   - 2.1. Vulnérabilité PHP ..... - 20 -

3. Apache ..... - 26 -
3.1. Vulnérabilité Apache ..... - 26 -
3.2. Vulnérabilité Configuration ..... - 27 -
3.3. Vulnérabilités de logicielle ..... - 32 -
3.3.3. Vulnérabilité de gestion de mémoire ..... - 33 -
3.4. Comment trouver les vulnérabilités d'un serveur web ..... - 33 -
3.5. Sécuriser un serveur apache ..... - 34 -
4. MySQL ..... - 35 -
4.1. Vulnérabilité MySQL ..... - 35 -
5. Windows ..... - 36 -
5.1. Vulnérabilités Windows ..... - 36 -
5.2. Sécuriser Windows ..... - 37 -
6. Linux ..... - 37 -
6.1. Vulnérabilités Linux ..... - 38 -
6.2. Comment Sécuriser Linux ..... - 38 -
7. Conclusion ..... - 39 -
Chapitre III. Approche pour tester l'application Web ..... - 40 -
1. Introduction ..... - 40 -
2. Choix de l'approche ..... - 40 -
3. Les étapes de l'approche ..... - 43 -
1. Mapper le contenu de l'application ..... - 43 -
2. Analyser l'application ..... - 45 -
3. Tester les contrôles de côté client ..... - 46 -
4. Tester le mécanisme d'authentification ..... - 47 -
5. Tester le mécanisme de gestion des sessions ..... - 50 -
6. Tester le contrôle d'accès ..... - 53 -
7. Tester les vulnérabilités basées sur les entrées ..... - 54 -
8. Tester les vulnérabilités de serveur d'application ..... - 58 -
4. Conclusion ..... - 59 -
Chapitre IV. Début des tests ..... - 60 -
1. Introduction ..... - 60 -
1. Phase de Mappage du contenu de l'application ..... - 60 -
2. Analyser l'application ..... - 65 -
3. Tester les contrôles de côté client ..... - 67 -
4. Tester le mécanisme d'authentification ..... - 68 -
5. Tester le mécanisme de gestion des sessions ..... - 71 -

6. **Tester le contrôle d'accès** ..... - 76 -
7. **Tester les vulnérabilités basées sur les entrées** ..... - 77 -
8. **Tester les vulnérabilités de serveur d'application** ..... - 84 -

2. **Conclusion** ..... - 89 -

Conclusion générale ..... - 90 -

Annexe : ..... - 91 -

Références ..... - 102 -

Liste des Figures

Figure 1: page par défaut phpinfo.php[1] ..... - 28 -
Figure 2 Exemple de script de session sur Apache Tomcat[1] ..... - 29 -
Figure 3 : liste de répertoire ..... - 30 -
Figure 4 :Etapes mappage le contenu de l'application ..... - 43 -
Figure 5 : Résumé des étapes pour analyser l'application ..... - 45 -
Figure 6: Résumé des étapes pour tester les contrôles de côté client ..... - 46 -
Figure 7: Résumé des étapes Tester le mécanisme d'authentification ..... - 47 -
Figure 8: Résumé des étapes pour tester le mécanisme de gestion des sessions ..... - 50 -
Figure 9: Résumé des étapes pour Tester le contrôle d'accès ..... - 53 -
Figure 10 : Résumé des étapes pour Tester les vulnérabilités basées sur les entrées ..... - 55 -
Figure 11 : Résumé des étapes pour tester les vulnérabilités de serveur d'application ..... - 58 -
Figure 12: Résultats BurpSuite ..... - 60 -
Figure 13 : Examination du code 1 ..... - 61 -
Figure 14 : Examination du code 2 ..... - 61 -
Figure 15 : Résultat du script ..... - 62 -
Figure 16 : Résultat de l'outil Nikto ..... - 63 -
Figure 17 : Résultat recherche par l'Url ..... - 64 -
Figure 18 : information pour la connexion a la base de données MySQL ..... - 67 -
Figure 19 : Code validation des mdp ..... - 68 -
Figure 20 Réponse de serveur quand nous soumettrons un nom d'utilisateurs qui existe ..... - 69 -
Figure 21 Réponse de serveur quand nous soumettrons un nom d'utilisateur qui n'existe pas ..... - 69 -
Figure 22 : code blocage ..... - 70 -
Figure 23: code qui teste l'unicité d'utilisateur ..... - 71 -
Figure 24:code mécanisme de gestion des sessions ..... - 72 -
Figure 25 : Résultat de l'outil Hash-identifier ..... - 73 -
Figure 26: code de terminaison de session ..... - 75 -
Figure 27: test de génération de jetons1 ..... - 75 -
Figure 28:test de génération de jetons2 ..... - 76 -
Figure 29 : Code contrôle d'accès ..... - 76 -

Figure 30 :Examination code XSS ..... - 79 -
Figure 31 : Résultat test XSS reflétée ..... - 80 -
Figure 32 :Résultat test XSS enregistrée ..... - 80 -
Figure 33 : injection Script ..... - 82 -
Figure 34: injection script 2 ..... - 84 -
Figure 35 : code contre l'inclusion des fichiers ..... - 84 -
Figure 36 : interface phpMyAdmin ..... - 85 -
Figure 37 : Recherche de vulnérabilités dans la base de données NIST ..... - 86 -
Figure 38: Recherche de vulnérabilités dans la base de données CVE DETAILS ..... - 86 -
Figure 39 : réponse de requête http en utilisant la méthode OPTION ..... - 87 -
Figure 40: résultat pour la requête GET ..... - 88 -
Figure 41: résultat pour la requête CONNECT ..... - 88 -

Liste des tableaux

Tableau 1 : Récapitulatif des approches et méthodes ..... - 42 -
Tableau 2: Récapitulatif des Url trouvés phase 1 ..... - 65 -
Tableau 3: Tableau qui résume les points d'entrée des données ..... - 66 -
Tableau 4 : vulnérabilités des technologies php, MySQL, Apache et Windows leur impacts ainsi que leur parades ..... - 101 -

Introduction générale

Problématique

La sécurité des applications Web est un problème désormais récurrent. Le nombre de vulnérabilités web recensées s'accroît constamment.

Ces attaques peuvent leur permettre, par exemple, d'obtenir des données confidentielles (numéros de cartes de crédit, mots de passe, etc.) qui sont manipulées par l'application, voire même de modifier ou détruire certaines de ces données. La complexité des technologies utilisées aujourd'hui pour réaliser les applications Web (Java, JavaScript, PHP, Ruby, J2E, etc.) fait qu'il est particulièrement difficile 1) d'empêcher l'introduction de vulnérabilité dans ces applications et 2) d'estimer ou de prévoir leur présence. De plus, la sécurisation des réseaux ainsi que l'installation de pare-feux ne fournit pas de protection satisfaisante contre les attaques Web car ces applications sont par définition publiques et accessibles à tous. Il est donc nécessaire d'auditer régulièrement les applications Web pour vérifier la présence de vulnérabilités exploitables.

Objectifs

Dans notre cas d'étude, la SONATRACH, envisage d'utiliser une application web de gestion des bases de données très confidentielles donc très sensibles. De ce fait, le but de ce travail est d'identifier les menaces et les vulnérabilités, ensuite proposer des parades en fonction de ces menaces et vulnérabilités trouvées. Enfin Développer ces parades pour faire face à chaque menace et vulnérabilité.

L'objectif du mémoire est donc de faire une approche conceptuelle et bien détaillée d'une méthode de pentest permettant d'identifier les vulnérabilités et les faiblesses des applications Web, d'exploiter ces vulnérabilités et de développer une solution pour les contrer.

Structure

Notre mémoire s'organise autour de quatre chapitres :

-Le premier chapitre est conçu pour comprendre de quoi le web est constitué, comment il marche, les protocoles utilisés, technologies pour fournir les Fonctionnalités côté serveur coté client et les types de codage qui sont appliqués. Enfin nous allons voir à quoi nous devrions nous attendre coté sécurité.

-Pour le deuxième chapitre nous avons énuméré toutes les vulnérabilités qui existent dans la pile Php, Apache, Mysql et Windows essentiellement pour les besoins de notre projet et linux à un degré moindre, ensuite nous avons proposé des parades pour s'y protéger.

-Ensuite pour le troisième chapitre nous avons proposé une approche pour tester l'application web et pour cela nous avons tracé 8 étapes bien définies pour tester les failles, les exploiter et implémenter des solutions pour toutes les vulnérabilités découvertes

-Enfin dans le quatrième chapitre nous avons appliqué les étapes de l'approche précédente à l'aide de différents outils et de scripts, recensé les failles qui existaient et mis en œuvre des solutions pour s'y protéger.

Chapitre I. Les technologies et sécurité Web

1. Introduction

Afin de bien comprendre le fonctionnement d'un site web nous allons le décortiquer et étudier son anatomie cela nous permettra d'avoir une idée bien précise de comment marche le web. Dans ce chapitre nous allons présenter le protocole http, ensuite nous présenterons les fonctionnalités web coté serveur, client et codage de caractères, et enfin nous terminerons par énumérer tous les risques à prévenir.

2. Le protocole HTTP

Le protocole http est le protocole de base des communications utilisé pour accéder au Web (World Wide Web) et il est utilisé par toutes les applications web de nos jours. HTTP a été développé initialement pour transférer des ressources statiques basées sur le texte, Il a depuis été étendu et exploité de diverses façons pour lui permettre de supporter les applications complexes distribuées qui sont maintenant courantes.

HTTP utilise un modèle basé sur les messages dans lequel un client envoie une requête et le serveur répond à cette requête. Le protocole est essentiellement sans connexion malgré qu'il utilise le protocole TCP en tant que mécanisme de transport, chaque échange de requêtes et réponses est une transaction autonome et peut utiliser une connexion TCP différente. [1]

2.1. Les Requête HTTP

Tous les messages HTTP (demandes et réponses) se composent d'un ou plusieurs en-têtes, chacun sur une ligne distincte, suivi d'une ligne vierge obligatoire, suivie d'un corps de message facultatif. [2]

Une requête HTTP typique est comme suit :

GET /auth/488/nosDétails.ashx?uid=129 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwaveflash, */*
Referer: https://exemple.net/auth/488/Home.ashx
Accept-Language: en-GB
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate Host: mdsec.net

Connection: Keep-Alive

Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476

La première ligne de chaque requête HTTP se compose de trois éléments, séparés par des espaces: [2]

1. Un verbe qui indique la méthode utilisée, la méthode la plus utilisée est GET.
2. L'URL demandée.
3. La version de protocole HTTP utilisée.

Les autres entêtes dans l'exemple :

- User-Agent est utilisé pour fournir des informations sur le navigateur ou un autre logiciel client qui a généré la requête.
- Host spécifie le nom d'hôte qui est apparu dans l'URL complète à laquelle nous accédons.
- Cookie permet de soumettre des paramètres supplémentaires que le serveur a émis au client.

2.2. Réponses HTTP

Une réponse HTTP typique est comme suit:

HTTP/1.1 200 OK

Date: Tue, 19 Apr 2011 09:23:32 GMT

Server: Microsoft-IIS/6.0 X-Powered-By: ASP.NET

Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc X-AspNet-Version: 2.0.50727

Cache-Control: no-cache

Pragma: no-cache

Expires: Thu, 01 Jan 1970 00:00:00 GMT Content-Type: text/html; charset=utf-8

Content-Length: 1067

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://

www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://

www.w3.org/1999/xhtml" ><head><title>Your details</title>

La première ligne de chaque réponse HTTP se compose de trois éléments, séparés par des espaces: [2]

- La version de protocole HTTP utilisée.
- Un code d'état numérique indiquant le résultat de la demande.
- Une «phrase de motif » textuelle décrivant l'état de la réponse.

Les autres entêtes dans la réponse HTTP : [2]

- Serveur contient une bannière indiquant le logiciel du serveur Web utilisé
- Set-Cookie émet au navigateur un autre cookie, ceci est renvoyé dans l'en-tête Cookie des requêtes ultérieures à ce serveur.
- Pragma indique au navigateur de ne pas enregistrer la réponse dans son cache.
- Content-Type indique que le corps de ce message contient un document HTML.
- Content-Length indique la longueur du corps du message en octets.

2.3. Les méthodes HTTP

Lorsque nous attaquons des applications web nous traiterons presque exclusivement deux types de méthode HTTP : GET et POST. La méthode GET est conçue pour récupérer les ressources, elle peut être utilisée pour envoyer des paramètres à la ressource demandée dans la chaîne de requête URL. Les URL sont affichées à l'écran et sont enregistrées dans différents endroits, comme l'historique de navigateur web et les journaux d'accès au serveur web, et c'est pourquoi la méthode GET ne doit pas être utilisée pour soumettre des informations sensibles. La méthode POST est conçue pour effectuer des actions, avec cette méthode, les paramètres de requête peuvent être envoyés à la fois dans la chaîne de requête URL et dans le corps du message. Et malgré que les URL soient toujours enregistrées, tous les paramètres envoyés dans le corps du message seront exclus d'enregistrement sur les journaux d'accès.

En plus des méthodes GET et POST, le protocole HTTP prend en charge de nombreuses autres méthodes qui ont été créées à des fins spécifiques. Voilà les méthodes que nous devons avoir une connaissance sur elles [2] :

- La méthode HEAD fonctionne de la même manière qu'une demande GET, sauf que le serveur ne doit pas renvoyer un corps de message dans sa réponse, par conséquent, cette méthode peut être utilisée pour vérifier si une ressource est présente avant de faire une demande GET.
- La méthode TRACE est conçue à des fins de diagnostic. Le serveur doit retourner dans le corps de la réponse le contenu exact du message de requête qu'il a reçu, Cela peut être utilisé pour détecter l'effet de n'importe quel serveur proxy entre le client et le serveur qui peut manipuler la requête.

- La méthode OPTION demande au serveur de signaler les méthodes HTTP disponibles pour une ressource particulière.
- La méthode PUT tente de télécharger la ressource spécifiée sur le serveur.

Il existe d'autres méthodes HTTP qui ne sont pas directement pertinentes pour attaquer les applications Web.

2.4. Les en-têtes HTTP

HTTP prend en charge un grand nombre d'en-têtes, certains en-têtes peuvent être utilisés pour les requêtes et les réponses, la section suivante décrit les en-têtes que nous rencontrerons lorsque nous attaquerons des applications Web [2].

2.4.1. Les en-têtes généraux [2]

- **Connection** indique à l'autre extrémité de la communication si elle doit fermer la connexion TCP une fois la transmission HTTP est terminée ou la maintenir ouverte pour d'autres messages.
- **Content-Encoding** spécifie quel type d'encodage est utilisé pour le contenu de corps de message, tel que gzip, qui est utilisé par certaines applications pour compresser les réponses pour une transmission plus rapide.
- **Content-Length** spécifie la longueur du corps du message, en octets.
- **Content-Type** spécifie le type de contenu dans le corps du message.
- **Transfer-Encoding** spécifie tout codage qui a été effectué sur le corps du message pour faciliter son transfert sur HTTP.

2.4.2 Les en-têtes de requête [2]

- **Accept** indique au serveur quels types de contenu le client est prêt à accepter.
- **Accept-Encoding** indique au serveur quels types de codage de contenu le client est prêt à accepter.
- **Authorization** soumet des informations d'identification au serveur pour l'un des types d'authentification HTTP intégrés.
- **Cookie** soumet des cookies au serveur que le serveur a déjà émis.
- **Host** spécifie le nom d'hôte qui apparaît dans l'URL demandée.
- **If-Modified-Since** spécifie quand le navigateur a reçu la ressource demandée pour la dernière fois, si la ressource n'a pas changé depuis cette date, le serveur peut demander au client d'utiliser sa copie en cache.

- **If-None-Match** spécifie une balise-entité, qui est un identifiant dénotant le contenu du corps du message. le navigateur soumet la balise-entité que le serveur a émis avec la ressource demandée lors de sa dernière réception. Le serveur peut utiliser la balise-entité pour déterminer si le navigateur peut utiliser sa copie en cache de la ressource.
- **Origin** est utilisé dans les requêtes de croisement de domaine Ajax pour indiquer le domaine d'origine de la requête.
- **Referer** spécifie l'URL à partir de laquelle la demande actuelle est originaire.
- **User-Agent** est utilisé pour fournir des informations sur le navigateur ou un autre logiciel client qui a généré la requête.

2.4.3. En-tête de réponse [2]

- **Access-Control-Allow-Origin** indique si la ressource peut être récupérée via des requêtes de croisement de domaine Ajax.
- **Cache-Control** transmet les directives de mise en cache au navigateur.
- **ETag** spécifie une balise-entité. Les clients peuvent soumettre cet identifiant dans les futures requêtes pour la même ressource dans l'en-tête If-None-Match pour notifier au serveur quelle version de la ressource est actuellement stockée dans son cache.
- **Expires** indique au navigateur la durée de validité du contenu du corps du message.
- **Location** est utilisée dans les réponses de redirection pour spécifier la cible de la redirection.
- **Pragma** transmet les directives de mise en cache au navigateur (no-cache par exemple).
- **Server** fournit des informations sur le logiciel du serveur Web utilisé.
- **Set-Cookie** émet des cookies vers le navigateur qu'il les soumettra et renverra au serveur dans les requêtes ultérieures.
- **WWW-Authenticate** est utilisé dans les réponses qui ont un code d'état 401 pour fournir des détails sur les types d'authentification que le serveur prend en charge.
- **X-Frame-Options** indique si et comment la réponse actuelle peut être chargée dans un cadre de navigateur.

2.5. Cookies

Les cookies sont une partie essentielle du protocole HTTP sur lesquels s'appuie la plupart des applications Web [1]. Souvent, ils peuvent être utilisés comme un moyen d'exploiter les vulnérabilités, le mécanisme des cookies permet au serveur d'envoyer des items de données au

client, que le client stocke et soumets à nouveau au serveur dans chaque requêtes ultérieures sans aucune action particulière requise par l'application ou l'utilisateur. [1]

Un serveur émet un cookie à l'aide de l'en-tête de réponse Set-Cookie :

Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc

Le navigateur de l'utilisateur ajoute automatiquement l'en-tête suivant aux requêtes suivantes vers le même serveur :

Cookie: tracking=tI8rk7joMx44S2Uu85nSWc

Les cookies consistent normalement en un pair nom / valeur, mais ils peuvent consister en une chaîne de caractères qui ne contient pas d'espace, en outre plusieurs cookies peuvent être émises en utilisant plusieurs en-têtes de Set-Cookie dans la réponse du serveur, ceux-ci sont soumis au serveur dans le même en-tête Cookie, avec un point-virgule séparant différents cookies individuels.

En plus de la valeur réelle du cookie, l'en-tête Set-Cookie peut inclure l'un des attributs facultatifs suivants, qui peut être utilisé pour contrôler la façon dont le navigateur gère le cookie :

- **Expires** définit une date jusqu'à laquelle le cookie est valide.
- **Domain** spécifie le domaine pour lequel le cookie est valide.
- **Path** spécifie le chemin d'accès URL pour lequel le cookie est valide.
- **Secure** si cet attribut est défini, le cookie ne sera soumis que dans les requêtes HTTPS.
- **HttpOnly** si cet attribut est défini, le cookie ne peut pas être consulté directement via JavaScript.

Chacun de ces attributs de cookie peut avoir un impact sur la sécurité de l'application. L'impact principal est la capacité de l'attaquant à cibler directement les autres utilisateurs de l'application.

2.6. HTTPS

Le protocole HTTP utilise le TCP comme son mécanisme de transport, qui n'est pas chiffré et peut donc être intercepté par un attaquant qui est correctement positionné sur le réseau. HTTPS est essentiellement le même protocole de couche d'application que HTTP mais il est

transféré sur le mécanisme de transport sécurisé le protocole SSL, ainsi nous protégeons la confidentialité et l'intégrité des données passées sur le réseau. [3]

2.7. Les proxys HTTP

Un proxy HTTP est un serveur qui sert d'intermédiaire entre le navigateur client et

Le serveur Web de destination. Lorsqu'un navigateur a été configuré pour utiliser un serveur proxy, il transmet toutes ses requêtes à ce serveur, ensuite le proxy relève les requêtes aux serveurs Web concernés et transmet leurs réponses au navigateur.

Nous devons être conscients de deux différences dans la façon dont HTTP fonctionne lorsqu'un serveur proxy est utilisé [1] :

- Lorsqu'un navigateur envoie une requête HTTP non cryptée vers un serveur proxy, il place l'URL complète dans la requête, y compris le préfixe de protocole http: //, le nom d'hôte du serveur, le serveur proxy extrait le nom d'hôte et le port et les utilise pour diriger la demande vers le serveur Web de destination.
- Lorsque HTTPS est utilisé, le navigateur ne peut pas exécuter le handshake SSL avec le serveur proxy, car cela risque de briser le tunnel sécurisé et de laisser les communications vulnérables aux attaques d'interception, et par conséquence le navigateur doit utiliser le proxy comme un relais de niveau TCP qui transmet toutes les données dans les deux sens entre le navigateur et le serveur Web de destination.

2.8. Authentification HTTP

Le protocole HTTP comprend ses propres mécanismes d'authentification des utilisateurs en utilisant différents techniques d'authentification, y compris les suivants [4] :

- **Basic** est un mécanisme d'authentification simple qui envoie les informations d'identification des utilisateurs comme une chaîne encodée en Base64 dans un en-tête de requête avec chaque message.
- **NTLM** est un mécanisme d'authentification défi-réponse qui utilise une version du protocole Windows NTLM.
- **Digest** est un mécanisme d'authentification défi-réponse qui utilise la somme de contrôle MD5 pour vérifier les identifiants des utilisateurs

Il est relativement rare de rencontrer ces protocoles d'authentification utilisés par les applications Web déployées sur Internet.

3. Les Fonctionnalités Web

En plus du protocole de communication de base utilisé pour envoyer des messages entre le client et le serveur, les applications Web utilisent de nombreuses technologies pour fournir leurs fonctionnalités. Toute application raisonnablement fonctionnelle peut employer des dizaines de technologies distinctes au sein de ses composants serveur et client, et c'est pourquoi nous avons besoins d'une compréhension de base de la façon dont ses fonctionnalités sont implémentées, de la manière dont les technologies utilisées sont conçues, comment elles se comportent et c'est quoi leurs points faibles.

3.1. Fonctionnalité côté serveur

Les applications Web d'aujourd'hui utilisent un nombre équitable de ressources statiques. Cependant, une grande partie du contenu qu'ils présentent aux utilisateurs est générée dynamiquement. Lorsque le navigateur d'un utilisateur demande une ressource dynamique, normalement, il ne demande pas simplement une copie de cette ressource, mais en général, il soumet également différents paramètres qui permettent à l'application côté serveur de générer du contenu adapté à l'utilisateur individuel.

Les requêtes HTTP peuvent être utilisées pour envoyer des paramètres à l'application de quatre façons principales:

- Dans la chaîne de caractère de requête URL.
- Dans le chemin du fichier des URL REST-style.
- Les cookies HTTP.
- Dans le corps des requêtes en utilisant la méthode POST.

En général, les applications Web utilisent une large gamme de technologies du côté du serveur pour offrir leurs fonctionnalités :

- Langages de scripts tels que PHP, VBScript, Perl.
- Plateforme d'application web tels que Java et ASP.NET.
- Serveur Web tels qu'IIS, Apache et Netscape Enterprise.
- Bases de données tels que MS-SQL, MySQL et Oracle
- Autres composants de back-end tels que les systèmes de fichiers, les services Web SOAP et les services d'annuaire

3.1.1. PHP

Le langage PHP a émergé d'un projet de loisir, Il a évolué peu à peu comme un Framework très puissant et riche pour développer des applications Web. Il est souvent utilisé en conjonction avec d'autres technologies libres dans ce que l'on appelle la pile LAMP (Linux, Apache, MySQL, PHP). De nombreuses applications et composants open source ont été développés à l'aide de PHP tels que :

- **Front-End administrative** : PHPMyAdmin.
- **Web mail** : SquirrelMail, IlohaMail.
- **Galerie des photos** : Gallery.
- **Paniers :** osCommerce, ECW-Shop.
- **Wikis** : MediaWiki, WikkaWikki.

À cause de la gratuité, la facilité de PHP, il a souvent été le langage de choix pour de nombreux débutants pour développer des applications Web, en outre, la conception et la configuration par défaut du Framework PHP ont depuis toujours permis aux programmeurs d'introduire inconsciemment des bugs de sécurité dans leur code, de plus, plusieurs défauts ont existé dans la plate-forme PHP elle-même.

3.1.2. SQL

Langage de requête structurée (SQL) est utilisé pour accéder aux données dans une base de données relationnelle comme Oracle, MS-SQL ou MySQL, La grande majorité des applications Web d'aujourd'hui utilisent des bases de données basées sur SQL comme leur dépôt de données de back-end, SQL utilise des requêtes pour effectuer des tâches communes telles que la lecture, l'insertion la mise à jour et la suppression de données. Les applications Web peuvent intégrer une entrée fournie par l'utilisateur dans des requêtes SQL exécutées par la base de données back-end afin d'implémenter la fonctionnalité dont ils ont besoin, si ce processus n'est pas exécuté en toute sécurité, les attaquants peuvent soumettre une entrée malveillante pour interférer avec la base de données et potentiellement lire et écrire des données sensibles.

3.2. Fonctionnalité côté Client

Pour que l'application côté serveur reçoive l'entrée et les actions de l'utilisateur et présente les résultats à l'utilisateur, elle doit fournir une interface utilisateur côté client.

3.2.1. HTML

La technologie de base utilisée pour créer des interfaces Web est le langage de balisage hypertexte, HTML est un langage basé sur un tag qui permet de décrire la structure des documents rendus dans le navigateur. HTML est devenu un langage riche et puissant qui peut être utilisé pour créer des interfaces utilisateur hautement complexes et fonctionnelles.[1]

3.2.2. Hyperliens

Une grande quantité de communication entre le client et le serveur est guidée par l'utilisateur qui clique sur les hyperliens. Dans les applications Web, les hyperliens contiennent fréquemment des paramètres de requête prédéfinis, ce sont des éléments de données que l'utilisateur n'entre jamais; Ils sont soumis parce que le serveur les place dans l'URL cible du lien hypertexte auquel l'utilisateur clique. Par exemple une application web a un nombre de liens vers des articles qui ont la forme suivante:

** Que ce passe-t-il?**

Lorsqu'un utilisateur clique sur ce lien, le navigateur effectue la demande suivante:

GET /articles/8/?redir=/maj/maj29.html HTTP/1.1
Host: siteinternet.net
...

Le serveur reçoit le paramètre **redir** dans la chaîne de requête et utilise sa valeur pour déterminer quel contenu doit être présenté à l'utilisateur.

3.2.3. Les Formulaires

Les formulaires HTML sont le mécanisme habituel permettant aux utilisateurs d'entrer des entrées arbitraires via leur navigateur. Un formulaire typique est comme suit:

<form action="/secure/login.php?app=quotations" method="post">
**username: <input type="text" name="username">
**
password: <input type="password" name="password">
<input type="hidden" name="redir" value="/secure/home.php">
<input type="submit" name="submit" value="log in">
</form>

Lorsque l'utilisateur entre les valeurs dans le formulaire et clique sur le bouton Soumettre, le navigateur effectue une requête comme suit:

POST /secure/login.php?app=quotations HTTP/1.1
Host: siteinternet.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 39
Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd
username=daf&password=foo&redir=/secure/home.php&submit=log+in

Dans cette requête, plusieurs points d'intérêt reflètent la manière dont les différents aspects de la requête sont utilisés pour contrôler le traitement au côté serveur:

- Étant donné que la balise de formulaire HTML contient un attribut spécifiant la méthode POST, le navigateur utilise cette méthode pour soumettre le formulaire et place les données du formulaire dans le corps du message de requête.
- En plus des deux éléments de données que l'utilisateur entre, le formulaire contient un paramètre caché (redir) et un paramètre de soumission (soumettre). Les deux sont soumis dans la requête et peuvent être utilisés par l'application du côté serveur pour contrôler sa logique.
- L'URL cible pour l'envoi de formulaire contient un paramètre prédéfini (app), comme dans l'exemple de lien hypertexte montré précédemment. Ce paramètre peut être utilisé pour contrôler le traitement côté serveur.
- La requête contient un paramètre de cookie (SESS), qui a été délivré au navigateur dans une réponse antérieure du serveur. Ce paramètre peut être utilisé pour contrôler le traitement côté serveur.

La requête précédente contient un en-tête spécifiant que le type de contenu dans le corps du message est x-www-form-urlencoded. Cela signifie que les paramètres sont représentés dans le corps du message en tant que paires nom / valeur de la même manière qu'ils sont dans la chaîne de requête URL.

3.2.4. CSS

Cascading Style Sheets (CSS) est une langue utilisée pour décrire la présentation d'un document écrit dans un langage de balise, dans les applications Web, il est utilisé pour spécifier comment le contenu HTML doit être rendu à l'écran. Les normes Web modernes visent à séparer autant que possible le contenu d'un document de sa présentation, cette séparation présente de nombreux avantages, y compris des pages HTML plus simples et plus petites, une mise à jour plus facile du formatage sur un site Web et une meilleure accessibilité.

Dans les premiers jours de la sécurité des applications Web, CSS a été largement négligé et a été considéré comme n'ayant aucune incidence sur la sécurité. Aujourd'hui, CSS est de plus en plus pertinent à la fois en tant que source de vulnérabilités de sécurité et comme moyen de réaliser des exploits efficaces pour d'autres catégories de vulnérabilités. [4]

3.2.5. JavaScript

Les hyperliens et les formulaires peuvent être utilisés pour créer une interface utilisateur riche qui peut facilement rassembler la plupart des types d'entrée requises par les applications Web, Cependant, la plupart des applications utilisent un modèle plus distribué, dans lequel le côté client est utilisé non seulement pour soumettre des données et des actions utilisateur, mais aussi pour effectuer un traitement réel des données. Cela se fait pour deux raisons principales [1]:

- Il peut améliorer les performances de l'application, car certaines tâches peuvent être effectuées entièrement sur les composants de client, sans qu'il soit nécessaire de faire un tour de requêtes et de réponses au serveur.
- Il peut améliorer la convivialité, car certaines parties de l'interface utilisateur peuvent être mises à jour dynamiquement en réponse aux actions de l'utilisateur, sans avoir besoin de charger une page HTML entièrement nouvelle fournie par le serveur.

JavaScript est un langage de programmation relativement simple mais puissant qui peut être facilement utilisé pour étendre les interfaces Web de plusieurs manières, ce qui n'est pas possible avec seulement du HTML, il est couramment utilisé pour effectuer les tâches suivantes :

- La validation des données saisies par l'utilisateur avant d'être soumise au serveur pour éviter les demandes inutiles si les données contiennent des erreurs.
- Modification dynamique de l'interface utilisateur en réponse aux actions de l'utilisateur.
- Recherche et mise à jour du modèle d'objet de document (DOM) dans le navigateur pour contrôler le comportement du navigateur.

3.2.6. Document Object Model (DOM)

Le modèle d'objet de document (DOM) est une représentation abstraite d'un document HTML qui peut être interrogé et manipulé via l'API DOM.

Le DOM permet aux scripts côté client d'accéder à des éléments HTML individuels par leur identifiant et à traverser la structure des éléments d'une manière automatique (programmable), de plus les données telles que l'URL et les cookies actuels peuvent également être lues et mises à jour. La manipulation du DOM de navigateur est une technique clé utilisée dans les applications basées sur Ajax, comme décrit dans la section suivante.[4]

3.2.7. Ajax

Ajax est une collection de techniques de programmation utilisées du côté client pour créer des interfaces utilisateur qui visent à imiter l'interaction fluide et le comportement dynamique des applications de bureau traditionnelles, avec Ajax, certaines actions utilisateur sont traitées dans le code de script côté client et ne provoquent pas de rechargement complet de la page, au lieu de recharger toute la page, le script exécute une requête «en arrière-plan» et reçoit généralement une réponse beaucoup plus petite qui est utilisée pour mettre à jour dynamiquement une partie de l'interface utilisateur.

La technologie de base utilisée dans Ajax est XMLHttpRequest, après une certaine consolidation des normes, XMLHttpRequest est maintenant un objet JavaScript natif que les scripts côté client peuvent l'utiliser pour effectuer des requêtes «en arrière-plan» sans nécessiter un événement de navigation au niveau de la fenêtre, malgré son nom, XMLHttpRequest permet à la fois d'envoyer des contenus arbitraires dans des requêtes et les recevoir dans réponses de serveur.

Historiquement, l'utilisation d'Ajax a introduit de nouveaux types de vulnérabilités dans les applications Web. D'une manière plus générale, elle a aussi augmenté la surface d'attaque sur une application typique en introduisant plus de cibles potentielles d'attaque sur le côté du serveur et du client.[1]

3.2.8. État et sessions

Les applications doivent suivre l'état de l'interaction de chaque utilisateur avec l'application sur plusieurs requêtes, par exemple, une application d'achat peut permettre aux utilisateurs de parcourir un catalogue de produits, d'ajouter des éléments à un panier, de procéder à la caisse et de fournir des détails personnels et de paiement, pour rendre possible ce type de fonctionnalité, l'application doit conserver un ensemble de données d'état générées par les actions de l'utilisateur sur plusieurs requêtes, ces données sont normalement détenues dans une structure côté serveur l'ensemble de ces données est appelé une session. Lorsqu'un utilisateur effectue une action, l'application côté serveur met à jour les détails pertinents au

sein de la session de l'utilisateur et lorsque l'utilisateur veut afficher plus tard le contenu de son panier, les données de la session sont utilisées pour renvoyer les informations correctes à l'utilisateur.

Étant donné que le protocole HTTP est sans état, la plupart des applications ont besoin d'un moyen de ré-identifier les utilisateurs individuels sur plusieurs requêtes pour l'ensemble correct des données d'état à utiliser pour traiter chaque requête, normalement, cela est réalisé en émettant à chaque utilisateur un jeton qui identifie de manière unique la session de cet utilisateur. Ces jetons peuvent être transmis en utilisant n'importe quel type de paramètre de requête, mais la plupart des applications utilisent les cookies HTTP.[4]

3.3. Codage des caractères

Les applications Web utilisent plusieurs schémas de codage pour leurs données, Le protocole HTTP et la langue HTML sont historiquement basés sur le texte, et différents schémas de codage ont été conçus pour s'assurer que ces deux mécanismes peuvent gérer en toute sécurité des caractères inhabituels et des données binaires.

Lorsque nous attaquons une application Web, nous devons souvent coder des données en utilisant un schéma pertinent pour s'assurer qu'elles sont traitées de la manière que nous souhaitons. En outre, dans de nombreux cas, nous pouvons manipuler les systèmes de codage utilisés par une application pour provoquer un comportement que ses concepteurs n'avaient pas l'intention.

3.3.1. Codage des URL

Les URL sont autorisées à contenir uniquement les caractères imprimables dans le jeu de caractères US-ASCII (Ceux dont le code ASCII est compris entre 0x20 et 0x7e), en outre, plusieurs caractères dans cet intervalle sont limités parce qu'ils ont une signification particulière dans le schéma d'URL lui-même ou dans le protocole HTTP.

Le schéma de codage d'URL est utilisé pour coder tous les caractères problématiques dans le jeu de caractères ASCII afin qu'ils puissent être transportés en toute sécurité via HTTP, la forme codée par URL de n'importe quel caractère est le préfixe% suivi du code ASCII à deux chiffres du caractère exprimé en hexadécimal. [5]

Voilà quelques exemples de codage d'URL :

- %3d — =
- %25 — %

- %20 — Espace
- %0a — Nouvelle ligne
- %00 — Octet Nul

3.3.2. Codage Unicode

Unicode est une norme de codage de caractères conçue pour supporter tous les systèmes d'écriture du monde, il peut être utilisé pour représenter des caractères inhabituels dans les applications Web, l'encodage Unicode 16 bits fonctionne de manière similaire à l'encodage d'URL, pour la transmission sur HTTP la forme d'un caractère codé en Unicode-16 est préfixe %u suivi par le code Unicode de caractère exprimé en hexadécimal [1]:

- %u2215 — /
- %u00e9 — é

UTF-8 est une norme de codage de longueur de variable qui emploie un ou plusieurs octets pour exprimer chaque caractère, pour la transmission sur HTTP, la forme de codage UTF-8 pour les caractères multi octets est d'utiliser chaque octets exprimé en hexadécimal précédé de préfixe % :

- %c2%a9 — ©
- %e2%89%a0 — ≠

Le codage Unicode est intéressant si nous voulons attaquer des applications web il peut parfois être utilisé pour vaincre les mécanismes de validation des entrées d'utilisateurs.

3.3.3. Codage HTML

L'encodage HTML est utilisé pour représenter les caractères problématiques afin qu'ils puissent être incorporés en toute sécurité dans un document HTML. Des différents caractères ont une signification particulière en tant que méta caractères dans le code HTML et sont utilisés pour définir la structure d'un document plutôt que son contenu, pour utiliser ces caractères en toute sécurité dans le cadre du contenu du document, il faut les coder en HTML par exemple [4] :

- " — "
- ' — '
- & — &
- < — <
- > — >

Lorsque nous attaquons une application Web, nous utilisons le codage HTML pour chercher l'existence des vulnérabilités XSS.

3.3.4. Encodage Base64

L'encodage Base64 permet de représenter en toute sécurité toutes les données binaires en utilisant uniquement des caractères ASCII imprimables, il est couramment utilisé pour coder les pièces jointes de courrier électronique pour une transmission sécurisée sur SMTP et il est également utilisé pour coder les informations d'identification des utilisateurs dans l'authentification HTTP de base. [1]

4. La sécurité des applications web

Créer un site web peut sembler être une opération simple, mais les risques engendrés sont pourtant réels de voir son site détourné de son utilisation initiale. Ci-dessous nous allons décrire les risques à prévenir :

4.1. L'abus de ressources.

Cette attaque consiste tout simplement à accaparer tout ou partie des ressources d'un site web pour en bloquer le fonctionnement. Cela conduit au déni de service.

Les abus de ressources se produisent suite à une importante sollicitation de la mémoire, du processeur, des connexions aux bases de données ou bien de la bande passante.

Ce type d'abus est généralement facile à identifier, car il pose immédiatement des problèmes à l'administrateur.[19]

4.2. La destruction de données

Il s'agit de s'attaquer aux données, en utilisant une faille de l'application web. Par exemple l'injection SQL, il est possible d'effacer le contenu d'une table, ou bien de modifier des données dans cette table, et de rendre l'ensemble du site web inutilisable. [19]

4.3. La publication de données confidentielles

Il s'agit d'un accès par un pirate à des données auxquelles il ne devrait pas y accéder : par exemple, lire le profil d'un utilisateur qui souhaite rester confidentiel. [19]

4.4. Le détournement du site

Détourner un site revient à s'en servir dans un but différent de son objet initial. [19]

4.5. L'usurpation d'identité

Le respect de l'anonymat est toujours un débat récurrent, mais de nombreux sites requièrent une forme d'identification avant de donner un accès plus complet à leurs services. Exemple les sites commerciaux.

L'identité que l'on utilise sur un site web devient donc un instrument important de sécurité, d'autant plus que différents droits sont attachés à cette identité. Parfois même, l'identité est la seule protection possible sur un site. Il suffit de voler l'identifiant et le mot de passe d'une personne pour prendre sa place. Dans d'autres cas, c'est simplement le cookie ou la session d'un utilisateur qui suffit. [19]

5. Conclusion

Dans cette première partie, nous avons fournit une base théorique sur le fonctionnement d'un site web, tous ses fondements et fondations ainsi que les risques qu'on encoure. Tout cela nous montre que la sécurisation d'un site web est très compliquée puisque faut comprendre son fonctionnement et étudier toutes les failles qui existent.

Chapitre II. Les vulnérabilités dans les technologies utilisées pour le développement d'applications web (WAMP/LAMP)

1. Introduction

Selon le NIST, une vulnérabilité est une faiblesse dans un système, une application, un réseau qu'elle est sujet d'exploitation ou d'utilisation malicieuse ou abusive. Les applications web sont vulnérables à la soumission d'une saisie arbitraire par les utilisateurs. Donc chaque interaction d'utilisateur avec l'application doit être considérer comme malicieuse sauf si on prouve le contraire, le fait de ne pas résoudre ce problème correctement peut laisser les applications vulnérable à l'attaque de nombreuses façons. Comme pour tout type d'application, une application web dépend sur les autres couches de la pile technologique qui la supporte, y compris le serveur web, le réseau, et le système d'exploitation. Un attaquant peut cibler tous ces composants, par conséquence si un attaquant peut compromettre la technologie sur laquelle une application dépend, il pourra compromettre toute l'application.

Dans ce chapitre, nous passerons en revue les risques encourus par les applications web, qu'elles soient PHP, Mysql, Apache ou bien Windows et linux. Enfin nous présenterons les protections possibles à mettre en place pour s'en protéger.

2. PHP

PHP est le langage de programmation coté serveur le plus utilisé, avec plus de 80 % des serveurs web qui le déploie selon le W3 Tech.[1]

PHP est à la fois un langage et un Framework web, et comme tous les langages web, PHP a un grand nombre de bibliothèques qui contribuent à la sureté de code.

PHP est un langage 'développé' plutôt que délibérément conçu, et par conséquence le développement des applications PHP non sécurisé est très commun, donc si nous voulons utiliser PHP en toute sécurité, nous devrions être conscients de toutes ses failles.

2.1. Vulnérabilité PHP

PHP est un langage de programmation très populaire. Chaque développeur ou hébergeur doit comprendre les vecteurs d'attaque utilisés par les pirates informatique contre les applications PHP. [20]

2.1.1. XSS

XSS (Cross Site Scripting) permet l'injection de code html-javaScript dans un script PHP, en exploitant des variables mal protégées.

Il existe deux types de XSS [20] :

Le XSS réfléchi (non permanent)

Elle est appelée non permanente car elle n'est pas enregistrée dans un fichier ou dans une base de données. Elle est donc éphémère.

Le XSS stocké (permanent)

La faille permanente est la faille XSS la plus sérieuse car le script est sauvegardé dans un fichier ou une base de données. Il sera donc affiché à chaque ouverture du site.

Comment s'en protéger

La solution la plus adaptée contre cette faille est d'utiliser la fonction htmlspecialchars() . Cette fonction permet de filtrer les symboles du type <, & ou encore ", en les remplaçant par leur équivalent en HTML. Par exemple :

- Le symbole & devient &
- Le symbole " devient "
- Le symbole ' devient '
- ...

2.1.2. CSRF

Le CSRF (*Cross Site Request Forgeries*) exploite la confiance qu'un site a en un de ses utilisateurs authentifiés. L'attaque produit des modifications sur le site. Ces modifications sont celles que peut réaliser l'utilisateur authentifié (post dans un forum, modification de préférences, ...). Le pirate écrit la requête HTTP d'attaque. Le serveur web victime de l'attaque la reçoit et la traite.

Les conséquences peuvent varier, effectivement un attaquant pourrait effectuer des opérations plus ou moins dangereuses. Si la victime est un administrateur ou utilisateur privilégié, les conséquences peuvent inclure l'obtention d'un contrôle complet sur l'application Web - la suppression ou le vol de données, la désinstallation du produit, ou de l'utiliser pour lancer d'autres attaques contre tous les utilisateurs du produit. Parce que l'attaquant a l'identité de la victime, la portée de CSRF est limitée seulement par les privilèges de la victime.[20]

Comment s'en protéger :

Il n'existe malheureusement pas de protection parfaite contre la CSRF. Néanmoins La façon la plus rependue étant l'utilisation de :

-Authentification pat jeton (token)

-Un petit captcha

-Demande de confirmation

2.1.3. La faille CRLF

La faille CRLF (Carriage Return Line Feed) est la plus souvent utilisée pour récupérer le mot de passe de quelqu'un, grâce à la fonction mail() de la page "mot de passe oublié" présent sur un très grand nombre de sites. Il nous suffit de connaitre l'adresse mail de la victime et d'utiliser la faille pour nous mettre en copie de ce mail.[20]

Exemple :

```
<form action="password_oublie.php" method="POST">
   <input type="text" name="mail">
   <input type="submit" value="Envoyer">
</form>
```

Le hacker va alors essayer de se mettre en copie du mail. Pour cela il lui suffit d'insérer un retour à la ligne dans le champ input en utilisant la chaine "%0A". nous obtenons quelque chose comme : **victime@service.com%0Apirate@service.com**

Comment s'en protéger

Le moyen le plus efficace pour s'en protéger est tout bonnement de supprimer les retours à la ligne lors du traitement

```
<?php
// On récupère la valeur du input
$chaine_utilisateur = $_POST['mail'];
// On supprime les retour à la ligne
$chaine_secure = str_replace(array("\n","\r",PHP_EOL),'',$chaine_utilisateur);
?>
```

2.1.4. Injection

Principe

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

Les attaques d'injection consistent à injecter un code malicieux qui sera traité par un autre système ou interprété Par l'application courante. Ces attaquent incluent les:

- injections SQL ;

- injections LDAP ;

- injections XPath ;

Injections SQL : L'attaque par injection SQL consiste à injecter du code SQL qui sera interprété par le moteur de base de données. Le code malicieux le plus répandu est d'ajouter une instruction pour faire en sorte que la requête sous-jacente soit toujours positive.[20]

Comment s'en protéger

Il est possible d'utiliser PHP Data Objects (PDO) pour éviter ces erreurs, mais quand ce n'est pas possible il convient juste d'utiliser les fonctions **mysqli_real_escape_string()** la fonction protège des caractères spéciaux

2.1.5. La faille include

La faille include est également très dangereuse. Comme son nom l'indique, elle exploite une mauvaise utilisation de la fonction include() . Cette fonction est très souvent utilisée pour exécuter du code PHP se situant dans une autre page, et particulièrement pour la connexion à une base de données [20]. Par exemple :

<?php include('config.php'); ?>

La page config.php contient les variables avec les renseignements de connexion à la base de données. Avant tout, il faut savoir que cette faille est extrêmement dangereuse, car le hacker peut se servir de cette négligence pour faire ce qui lui plaît comme modifier, supprimer, changer les droits...

Il existe de très nombreux types de faille include() , et de très nombreux moyens d'exploiter cette fonction [20] .

Nous allons citer deux types :

La faille include à distance

C'est la faille la plus rencontrée, et la plus facilement exploitable.

Imaginons une page index.php contant :

<?php include($_GET['fichier']); ?>

Maintenant un exemple d'URL exploitant cette partie du code :

www.monsite.com/index.php?fichier=default.php

Dans ce cas la page default.php est incluse dans la page index.php. Imaginons alors que quelqu'un s'amuse à changer default.php par www.php.com. Le site Php s'exécutera alors sur le serveur de votre site et s'affichera au beau milieu de votre page index.php. Un hacker peut ainsi inclure un backdoor sur votre serveur, et là ça devient un peu plus embêtant. [20]

La faille include en local

Il est également possible d'utiliser la faille include pour inclure des fichiers qui se trouvent sur le serveur du site et ainsi de les exploiter. Nous allons également pouvoir nous balader dans les différents répertoires du site voire même dans les répertoires du serveur ! Une personne malintentionnée va donc pouvoir retrouver le fichier contenant les passwords (ou autre), sans trop de difficultés. Il pourra également forcer l'affichage de votre code PHP et donc voir les mots de passe pour la base de données qui sont en clair dans votre code.[20]

Corriger la faille Include

Cette faille est déjà plus complexe à protéger, dans la mesure où on ne peut pas complètement interdire l'inclusion sur le site (car sinon on ne pourra pas inclure nos propres fichiers). Il faut donc savoir au préalable ce que l'on veut inclure, rédiger un htaccess qui veille sur les fichiers importants.[20]

2.1.6. La faille upload

<input type="file" /> Cette balise est utilisée sur de nombreux sites qui proposent à leurs utilisateurs d'avoir une photo de profil ou d'inclure une image dans un message sur le forum. Le problème, c'est que la balise upload permet d'uploader n'importe quel fichier. Un utilisateur malintentionné pourrait sans problème s'amuser à uploader un fichier PHP malveillant (web shell par exemple) qui lui permettrait de prendre le contrôle total de votre site.

Exemple exploitation d'une faiblesse

-La double extension

Imaginons un site qui autorise uniquement les fichiers .jpg. Rien ne nous empêche de renommer le fichier en "backdoor.php.jpg" et de l'envoyer. Il est également possible de renommer son fichier en tant que "backdoor.php\0.jpg". Le "\0" indique au serveur qu'il arrive en bout de chaine. Tout ce qui suit ne sera donc pas interprété. Le fichier sera enregistré en tant que "backdoor.php" et aura passé haut la main le test d'extension.[20]

Comment s'en protéger

On peut resumer les grands points à considérer pour mettre en place un script d'upload sécurisé en 8 règles : [20]

1. Renommez le fichier
2. Ne pas enregistrer à la racine de votre site
3. Vérifiez la taille du fichier
4. Ne pas se fier aux extensions
5. Effectuez un scan anti-malware
6. Gardez le contrôle des permissions (CHMOD)
7. N'autorisez l'upload qu'aux utilisateurs inscrits et authentifiés
8. Limite le nombre de fichiers qu'un utilisateur peut mettre en ligne

2.1.7. Mauvaise configuration de sécurité

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

Si les composants ne sont pas mis à jour, l'attaquant peut exploiter les failles non corrigées.

Pour de nombreux outils, des options sont installées par défaut alors qu'elles ne sont pas nécessaires au bon fonctionnement de l'application.

De même de nombreuses applications sont installées avec des comptes créés avec des mots de passe par défaut. Ces comptes et mots de passe sont les cibles privilégiées des usurpations d'identité. [20]

Parade et bonnes pratiques

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

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

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

Après l'installation il faut désactiver voire supprimer tous les comptes inutiles. Le mot de passe des autres comptes doit être modifié dès que possible. Les comptes d'administration par défaut doivent être verrouillés. Il faut préférer l'utilisation de comptes d'administration créés manuellement.[20]

3. Apache

Le serveur web est un élément Essentiel des applications web, le serveur web apache est souvent placé au bord du réseau, par conséquence il devient l'un des services les plus vulnérable à l'attaque.

3.1. Vulnérabilité Apache

Comme avec les autres composants sur lesquels s'exécute une application Web, le serveur Web représente une zone importante de surface d'attaque par laquelle une application peut être compromise. Si le serveur web est compromis la sécurité de l'application web sera gravement dégradée, en donnant accès à des listes de répertoire, au code source pour les pages exécutables, à la configuration sensible et aux données d'exécution et à la possibilité de contourner les filtres d'entrées.

La localisation des vulnérabilités de serveur web inclue souvent la reconnaissance et la recherche, et pour cela les scanners de vulnérabilités peuvent être très efficaces à localiser les différentes failles. [1]

3.2. Vulnérabilité Configuration

Les vulnérabilités de configuration apache sont :

3.2.1. Identificateur par défaut

Plusieurs serveurs web contiennent des interfaces administratives qui peuvent être accessible au public, souvent les interfaces administratives ont des identificateurs par défaut qui sont bien connus, et ne sont pas nécessairement changer lors de l'installation.

Dans le cas de apache on a (utilisateur : Admin mot de passe : (rien) | utilisateur : tomcat mot de passe : tomcat | utilisateur : root mot de passe : root). [1]

3.2.2. Contenu par défaut

La plupart des serveurs d'application sont livrés avec une gamme de contenu et de fonctionnalité par défaut que nous pouvons utiliser pour attaquer soit le serveur lui-même soit l'application cible. Voici quelques exemples de contenu par défaut qui peuvent être intéressants. [1]

3.2.2.1. Fonctionnalité de debug

Les fonctionnalités de diagnostique utilisées par les administrateurs sont des outils de grande valeur pour les attaquants, car elles peuvent contenir des informations sur la configuration de serveur, l'état de serveur et applications en cours d'exécution sur le serveur.

La figure 1 nous montre la page par défaut phpinfo.php, qui existe sur plusieurs installations d'Apache, cette page exécute la fonction phpinfo() et retourne le résultat de cette fonction.[1]

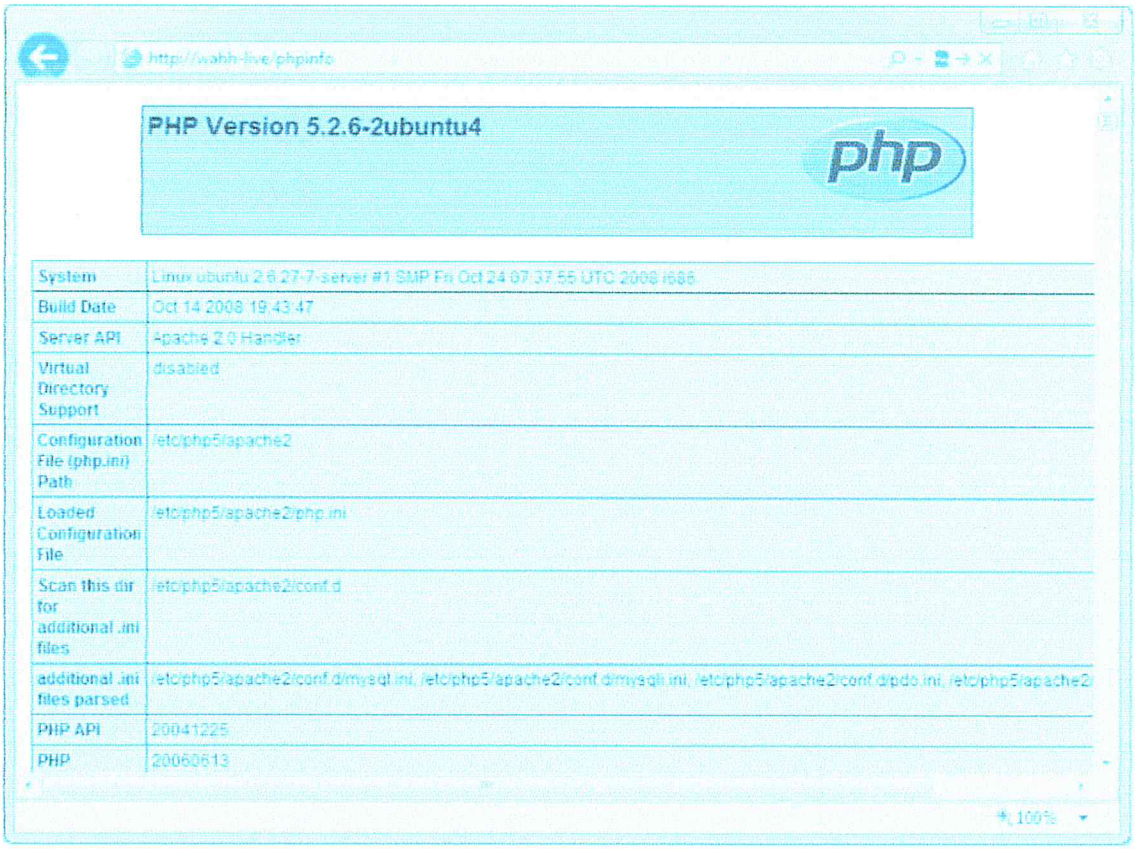


Figure 1: page par défaut phpinfo.php[1]

3.2.2.2. Les Exemples de fonctionnalité

Par défaut plusieurs serveurs web contiennent des exemples de scripts et de pages afin de démontrer comment certaines fonctions de serveur d'application et d'API peuvent être utilisées.

Typiquement elles sont censées être inoffensives et ne fournissent aucune opportunité pour un attaquant, toutefois dans la pratique, ce n'est pas le cas pour deux raison :

- Plusieurs exemples de script contiennent des vulnérabilités
- De nombreux exemples de scripts implémentent des fonctionnalités qui sont directement utilisées par un attaquant

Un exemple de deuxième problème est l'exemple de session dans Apache Tomcat comme le montre la figure 2, Si une application exécutée sur le serveur stocke des données sensibles dans la session d'un utilisateur, alors un attaquant peut voir ce script et interfère avec le traitement de l'application en modifiant ça valeur. [1]

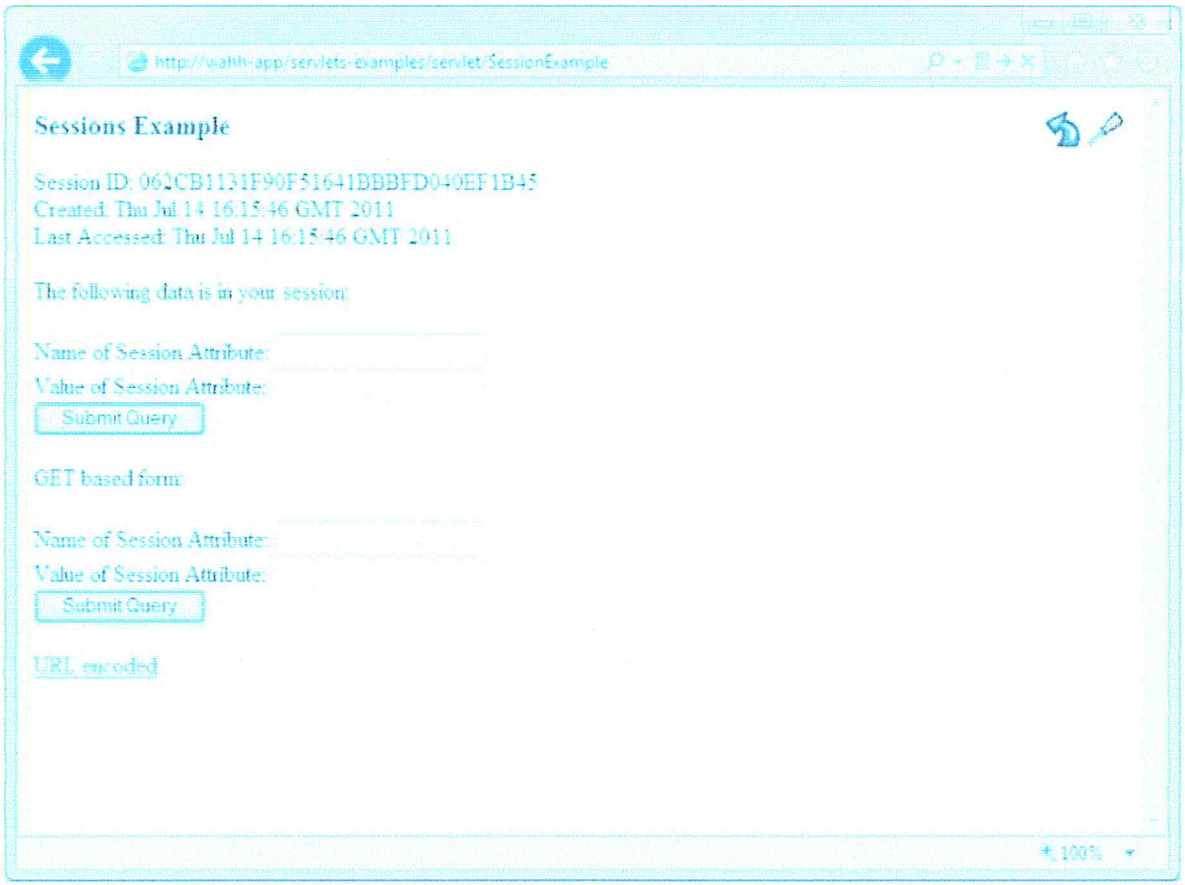


Figure 2 Exemple de script de session sur Apache Tomcat[1]

3.2.3. Liste du répertoire

Quand un serveur web reçoit une requête pour un répertoire plutôt qu'un fichier réel, il peut répondre de l'une des trois façons suivantes [1] :

- Il peut répondre par une ressource par défaut dans le répertoire, comme index.html par exemple
- Il peut retourner une erreur, comme code http 403 par exemple qui indique que la requête n'est pas permit
- Il peut retourner une liste qui montre le contenu d'un répertoire, figure 3

Figure 3 : liste de répertoire

L'obtention de la liste de répertoire peut nous aider à monter une attaque pour deux raison [1] :

- De nombreuses applications n'appliquent pas un contrôle d'accès approprié sur leurs fonctionnalités et ressources et s'appuient sur l'ignorance de l'attaquant des URL utilisées pour accéder aux informations sensibles
- Des fichiers et des répertoires sont souvent laissés involontairement dans la racine web comme les logs, fichiers de sauvegarde, script obsolète

3.2.4. Méthode WebDAV

WebDAV (Web-based Distributed Authoring and Versioning) est un protocole (extension de http), ce protocole ajoute de nombreuses méthodes qui peuvent être utilisées pour la manipulation des fichiers sur le serveur web, toute fois ils peuvent constituer un moyen pour attaquer une application, voici quelque méthodes qui peuvent être dangereuse [6] :

- PUT télécharge le fichier attaché à la destination spécifiée

- DELETE supprime la ressource spécifiée
- COPY copie la ressource spécifiée vers la location donnée dans l'entête destination
- MOVE déplace la ressource spécifiée vers la location donnée dans l'entête destination
- SEARCH recherche des ressources dans un chemin de répertoire
- PROPFIND récupère des informations sur une ressource comme la taille, l'auteur et le type

La méthode PUT est particulièrement dangereuse, car nous pouvons télécharger des scripts de porte dérobée sur le serveur web, ainsi nous pouvons contrôler complètement l'application.

3.2.5. Le serveur d'application en tant que proxy

Les serveurs web sont des fois configurés en tant que proxy inverse ou proxy direct, si le serveur est configuré comme proxy direct, il serait peut être possible d'exploiter le serveur pour effectuer diverses attaques :

- Un pirate peut utiliser le serveur pour attaquer des systèmes tiers sur internet et le trafic malicieux s'apparentera pour la cible qu'il est envoyé à partir de serveur compromis
- Un pirate peut utiliser le serveur pour se connecter à des hôtes arbitraires sur le réseau interne de l'organisation, ainsi il peut atteindre des cibles qu'ils ne sont pas accessible directement par internet.
- Un pirate peut utiliser le serveur pour se connecter à d'autres services qui sont exécutés sur le serveur ainsi contourné les restrictions de pare-feu, et potentiellement exploité la relation de confiance pour contourner l'authentification.

Nous pouvons utiliser deux méthodes pour amener un proxy direct à effectuer des connexions directes.

Premièrement nous pouvons envoyer des requêtes http contenant des URL qui inclue le nom de l'hôte et optionnellement le numéro de port.

GET http://Site.com:80/ HTTP/1.0
HTTP/1.1 200 OK
...

Si le serveur est configurer à avancer les requêtes pour l'hôte spécifier, nous aurons le contenu de la part de l'hôte.

Deuxièmement nous pouvons utiliser la méthode CONNECT pour spécifier le nom de l'hôte cible et le numéro de port.

CONNECT site.com:443 HTTP/1.0
HTTP/1.0 200 Connection established

Si le serveur répond de cette manière, il est en train d'avancer nos requêtes en tant que proxy.

3.3. Vulnérabilités de logicielle

Parmi les vulnérabilités de logiciel apache :

3.3.1. Déni de service

Les services réseaux sont susceptibles aux attaques de déni de service, qui tente à compromettre la disponibilité de service ciblé, il est impossible de se prémunir totalement contre ce type d'attaques, mais on peut quand même diminuer les dégâts.

Parmi les vulnérabilités qui existent dans la version 2.4.7 d'Apache sont :

3.3.1.1. Le module mod_status d'apache

La condition de concurrence dans le module mod_status dans le serveur http Apache permettre aux attaquants distants de causer un déni de service, ou possiblement l'obtention des informations d'identifications sensible ou encore l'exécution d'un code arbitraire, via une requête fabriquée qui déclenche une mauvaise gestion de tableau de bord dans la fonction status_handler dans modules/generators/mod_status.c et la fonction lua_ap_scoreboard-_worker dans modules/lua/lua_request.c. Cette vulnérabilité causera une divulgation considérable d'information et la modification de quelques fichiers de système, et enfin une réduction de performance et de disponibilité de serveur, toutefois pour exploiter ce vulnérabilité nous devrons avoir accès au htaccess.[7]

3.3.1.2. Le module mod_cgid

Le module mod_cgid dans le serveur http Apache n'a pas un mécanisme de timeout, ce qui permet aux attaquants de causer un déni de service via une requête de script CGI qui ne lit pas de son descripteur de ficher stdin.[8]

3.3.2. Exécution de code arbitraire

L'exécution de code arbitraire est une vulnérabilité qui donne à l'attaquant une possibilité d'exécuter un code arbitraire en l'injectant dans un système fonctionnaire.

Parmi les vulnérabilités qui existe dans le serveur apache :

3.3.2.1. mod_rewrite.c

mod_rewrite.c dans le module d'apache mod_rewrite avant la version 2.2.25 écrit les données dans un fichier log sans le nettoyage des caractères non imprimables, ce qui peut permettre à un attaquant d'exécuter des commandes arbitraire via une requête http contenant une séquence d'échappement pour un émulateur de terminal, ce qui est très compliqué mais si l'attaque est réussite, le pirate peut divulguer des informations considérables et peut être il pourra modifier certain fichier de système, et enfin il pourra interrompre le service.[9]

3.3.3. Vulnérabilité de gestion de mémoire

Le débordement de tampon est parmi les failles les plus graves qui peuvent affecter n'importe quel type de logiciel, puisqu'il permet à l'attaquant de contrôler l'exécution d'un processus vulnérable. Archiver une exécution de code arbitraire sur un serveur web va permettre à l'attaquant de contrôler les applications hébergées par ce serveur.

Parmi les vulnérabilités qui existent dans le serveur apache est la vulnérabilité de module mod_status expliqué précédemment.[1]

3.4. Comment trouver les vulnérabilités d'un serveur web

Si nous sommes chanceux, notre cible contiendra quelques vulnérabilités décris précédemment, cependant elles ont étaient probablement corrigées, et on aura besoin d'autres moyen pour attaquer le serveur web.

Un bon point de commencement sera l'utilisation des scanners de vulnérabilités automatiques telles que Nessus.

En plus de scanners de vulnérabilités, nous devons toujours faire des recherches nous-mêmes sur les vulnérabilités de logiciel cible, il existe plusieurs ressources en ligne telles que Security Focus, NVD et la mailing liste de bugtraq pour trouver plus de détails sur les vulnérabilités existantes sur notre cible qui ne sont pas encore fixées, ensuite il faut toujours consulter la base de données de metasploit pour voir si quelqu'un a déjà fait le travail pour nous en créant l'exploit d'une vulnérabilité existante sur notre cible.

Voici quelques URL qui peuvent aider :

- www.cvedetails.com
- nvd.nist.gov/home.cfm
- www.metasploit.com
- www.securityfocus.com

3.5. Sécuriser un serveur apache

Sécuriser la configuration de serveur web n'est pas une tâche difficile, il faut juste bien comprendre la documentation du produit et les tutoriaux de durcissements, parmi les problèmes à aborder :

- Changer tout les identificateurs par défaut, et supprimer tout compte par défaut qu'on n'a pas besoin
- Bloqu0er l'accès public aux interfaces d'administrateurs en utilisant des ACL ou des pare-feu
- Retirer tout contenu et fonctionnalité par défaut qu'on n'a pas besoin.
- Si une fonctionnalité par défaut est retenue, il faut l'endurcir autant que possible
- Vérifier les listes de répertoire dans tous les répertoires web, et désactiver les listes de répertoire
- Désactiver toutes les méthodes non nécessaires et garder seulement ce qui est nécessaire (typiquement POST et GET)
- Assurer que le serveur web n'est pas configuré en tant que proxy.

Une organisation soucieuse de la sécurité peut faire beaucoup pour se protéger contre les vulnérabilités logicielles :

- Appliquer les correctifs du vendeur d'apache
- Appliquer le principe de séparation des privilèges
- Surveiller les nouvelles vulnérabilités
- Utiliser des IDS ou IPS
- Nous pouvons imposer des filtres stricts au niveau de réseau sur le trafic vers et depuis le serveur web

4. MySQL

MySQL est l'un des systèmes de gestion des bases de données les plus utilisés dans le monde. Selon le classement du mois de novembre de DB-Engines, cette base de données relationnelle est le second au monde en termes de popularité juste derrière Oracle.

Les failles Mysql sont les plus dangereuses de toutes les autres failles des autres couches de la pile technologique qui supporte l'application web, car si la base de données sera compromise y'a un grand risque de compromettre le serveur, exécution d'un code arbitraire et enfin une élévation de privilèges.

4.1. Vulnérabilité MySQL

Parmi les vulnérabilités de MySQL :

4.1.1. Injection SQL

Une exploitation réussie [de la vulnérabilité CVE-2016-6662] permettrait à un attaquant d'exécuter du code arbitraire avec les privilèges root, ce qui lui permettrait de compromettre entièrement le serveur » détaille l'expert. Elle peut être exploitée localement ou à distance, autant via un accès avec authentification ou une injection SQL, même avec les modules SELinux et AppArmor installés. Toutes les versions de MySQL « en configuration par défaut » sont touchées (5.7, 5.6 et 5.5).

4.1.2. Elévation de privilège

Dans ce cas de figure, l'exécution de la déclaration REPAIR TABLE SQL par un utilisateur doté de privilèges réduits comme la sélection, la création et l'insertion est effectuée de manière non sécurisée. En conséquence, cela permet à un utilisateur local doté de droits d'accès limités dans une base de données d'élever ses privilèges afin d'exécuter du code arbitraire comme un utilisateur ayant des droits sur toutes les bases de données sur le serveur de base de données.

De manière détaillée, une fois que le tiers malveillant a eu accès à l'ensemble de ces bases de données sur le serveur, il peut se servir de cette vulnérabilité découverte et qui a pour référence CVE-2016-6664. La vulnérabilité CVE-2016-6664 permet à l'attaquant d'avoir accès aux répertoires /var/log ou /var/lib/mysql. Une fois à ce niveau, il suffit de supprimer le fichier error.log et de le remplacer avec un lien symbolique afin d'élever les privilèges.

5. Windows

Windows présente un grand danger pour notre application web s'il n'est pas sécurisé correctement et pro activement vu qu'il est facile à compromettre. L'absence de chiffrement des disques et le principe de séparation de privilèges sont les causes de plusieurs failles de sécurité sur Windows.

Toutefois si nous durcissons l'installation et la configuration de Windows nous pouvons diminuer le risque de ces vulnérabilités.

5.1. Vulnérabilités Windows

Parmi les vulnérabilités Windows :

5.1.1. Élévation de privilèges

L'élévation de privilèges est le résultat de l'octroi à un intrus d'autorisations supérieures à celles initialement accordées. Par exemple un intrus avec autorisations de lecture seulement, élèvent d'une façon ou d'une autre ses autorisations à lecture et écriture.

Un exemple de cette vulnérabilité sous Windows est comme suit :

Un attaquant arrivera à compromettre une machine d'une façon ou une autre, ensuite il essayera d'élever ses autorisation au niveau d'administrateur de cette machine, il pourra insérer une image de Windows et redémarrer la machine par la suite, il commencera l'installation, juste pour avoir la ligne de commande (en appuyant sur shift dans la sélection de la langue d'installation), puis il remplacera le fichier sethc.exe par cmd.exe et il annulera l'installation par la suite. Ensuite qu'on il aura le guide d'identification, il suffit juste d'appuyer sur shift cinq fois et on aura la ligne de commande en mode administrateur au lieu de processus de touches rémanentes. [10]

5.1.2. Abus de privilèges des administrateurs

Les permissions dénient ou permettent un certain type d'accès à une ressource, les privilèges donnent le droit de faire quelque chose dans le système d'exploitation.

Les administrateurs ont six superpuissances [10] :

- La possession qui permet l'accès à n'importe quelle ressource
- Debug qui permet l'ajustement du jeton d'accès au processus et la manipulation de la mémoire
- Sauvegarde et restauration qui permet de contourner tous les contrôles d'accès

- Imitation qui permet à un administrateur d'imiter n'importe qu'elle utilisateur connecté
- Les administrateurs décident qui aura des privilèges
- Ils peuvent être administrés avec des politiques mais ils peuvent toujours les contourner

5.1.3. Contournement des stratégies de groupes

Stratégies de groupe est un nom trompeur, car nous ne pouvons pas gérer les groupes avec les stratégies de groupes sous Windows. Stratégies de groupe est un ensemble de paramètres fournit pour les ordinateurs.

Un administrateur peut facilement contourner ces stratégies de groupe simplement par la suppression de ces paramètres de registre. En outre un administrateur peut nier le système de lire les paramètres d'éditeur de registre et contourne les stratégies de groupe par la suite, car sous Windows si nous ne pouvons pas lire les règles, elles ne s'appliquent pas sur nous. [10]

5.1.4. Contournement de logicielle de chiffrement

Le chiffrement de disque dur est mandataire, mais le chiffrement tout seul ne peut pas nous protéger, car les attaquants peuvent le contourner de plusieurs façon, par exemple en trompant l'utilisateur à désactiver le chiffrement en injectant une tâche planifiée qui change le mot de passe de restauration, ou par des attaques matérielle comme attaque-DMA, l'attaque de Princeton, ou l'enlèvement de la mémoire qui est la méthode la plus délicate à exécuter.[10]

5.2. Sécuriser Windows

Pour une installation sécurisée de Windows il faut :

- Avoir un logiciel de chiffrement (BitLocker par exemple)
- Appliquer le principe de séparation de privilèges
- Limiter l'utilisation des privilèges d'administrateur
- Utiliser UAC
- Sensibilisation des utilisateurs finaux

6. Linux

Tout comme Windows, Linux est facile à pirater s'il est mal configuré et mal installé, il existe plusieurs problèmes sous linux tels que les vulnérabilités de noyau, l'élévation de privilèges,

l'abus de privilèges et les services vulnérables. Mais avec un bon durcissement de configuration d'installation nous pourrons diminuer le risque de ces vulnérabilités.

6.1. Vulnérabilités Linux

Il existe plusieurs vulnérabilités de linux parmi ces vulnérabilités :

6.1.1. Les vulnérabilités de noyau de Linux

Il existe plusieurs types de vulnérabilités qui affectent le noyau de Linux. Les vulnérabilités les plus répandues en 2017 selon cvedetails.com sont les vulnérabilités DOS (18 vulnérabilités), l'exécution de code arbitraire (80 vulnérabilités), débordement de tampon (8 vulnérabilités), mémoire corrompue (4 vulnérabilités), gain de l'information (27 vulnérabilités) et l'élévation de privilèges (12 vulnérabilités). [11]

6.1.2. Elévation de privilèges

Les exploits les plus récents comme Dirty Cow montre que malgré les améliorations continues de la sécurité de Linux, l'élévation de privilèges reste un problème pour la communauté Linux. L'élévation de privilèges est un pas important dans la méthodologie d'un attaquant, l'élévation de privilège consiste à exploiter les vulnérabilités de système pour augmenter les privilèges dans le but d'avoir plus d'accès aux différentes ressources de systèmes.[11]

6.1.3. Les vulnérabilités DOS

Les vulnérabilités de déni de service affectent la disponibilité de système, le traitement des attaques de déni de service est une tâche difficile, car il ne nécessite pas seulement la détection et la prévention d'un adversaire d'effondrer le noyau ou le système, mais aussi d'assurer l'exécution correcte de noyau et de système pour effectuer correctement les opérations des utilisateurs.

Il existe plusieurs vulnérabilités de déni de service sous linux, plus de 70 % des vulnérabilités de linux en 2016 sont de type DOS, ce qui montre que les vulnérabilités de déni de service présente un grand danger pour les environnements basés sur linux. [11]

6.2. Comment Sécuriser Linux

Le processus de sécurisation d'un système Linux/Unix dépend de système déployer mais généralement il faut faire [12] :

- Sécuriser la location physique avec des serrures

- Appliquer tous les correctifs et les mises à jour de système installé
- Diminuer au maximum l'intervention des administrateurs (Automatisation)
- Chercher les autorisations inappropriés des fichiers et répertoires et les corriger si trouvées
- Désactiver tous les services inutiles
- Si possible exécuter les processus de server en tant qu'un utilisateur spécial et non pas en tant que root
- Spécifier les contrôles d'accès et de connexion pour tous les services
- Choisir un bon mot de passe root et changer le régulièrement
- Empêcher les connexions root, sauf sur la console système
- Utiliser SSH pour tout accès à distance
- Activer la comptabilisation des processus.
- Effectuer les sauvegardes et vérifier les données

7. Conclusion

Dans ce deuxième chapitre nous avons énuméré toutes les vulnérabilités qui existent dans la pile Php, MySQL, Apache, Windows et linux, ensuite nous avons proposé des parades pour s'y protéger. Nous concluons que pour bien sécuriser un site web, Il est essentiel que les programmeurs et les administrateurs comprennent parfaitement ces vulnérabilités pour y faire face et pour que l'application web tienne toutes ses promesses à l'entreprise coté sécurité.

Chapitre III. Approche pour tester l'application Web

1. Introduction

Ce chapitre contient une approche étape par étape détaillée que nous allons suivre pour tester les failles d'une application Web. Il couvre toutes les catégories de vulnérabilités et Techniques d'attaque décrites dans les chapitres précédents.

Nous allons utiliser les étapes de cette approche pour guider notre travail en tant que liste de contrôle qu'il faut respecter pour une sécurité soit optimale.

2. Choix de l'approche

Il existe trois type de tests de pénétration, Black Box, White Box, un Grey Box, dans un test Black Box le client nous ne fournit aucune informations sur son système avant le début du test, par contre dans un test White Box le client nous fournit des détails complets du réseau et des applications, et enfin dans un test Grey Box le client peut fournir des détails partiels sur les systèmes cibles.

Il existe une différence entre un test de pénétration et un scan de vulnérabilité, le but d'un scan de vulnérabilité est d'identifier, classer et signaler les vulnérabilités qui peuvent compromettre le système une fois exploitées, ensuite un scan de vulnérabilité est fait au moins une fois par trimestre ou après des changements importants en utilisant des outils d'automation et des vérifications manuelle, par contre dans un test de pénétration notre but est d'identifier une manière pour contourner les mesures de sécurité utilisées par le système ce test est conduit annuellement ou après des changements importants en suivant un processus manuelle qui inclut l'utilisation des scanners de vulnérabilités et d'autres outils d'automation et à un rapport détaillé sur les résultats trouvés.

Nous avons proposé une approche inspiré des autres méthodes et approches, mais qui traite exclusivement les applications web afin de conduire des tests de pénétration contre l'applicatif de gestion de base de données de Sonatrach, et proposer des parades par la suite.

Cette approche est construite et validée avec la collaboration de l'équipe de sécurité de la Sonatrach.

Voici un tableau comparatif entre l'approche proposé et les autres méthodes/approche qui existent sur le marché :

App e/M de	oma	Nc e D'é	Rapport	Toolkit	Teste d'authentification web	Teste de Session	Recommandation d'atténuation
Approche Suivi	Application Web	44	Oui	Suggère l'utilisation de certains outils	Oui	Oui	Oui
Penetration Testing Execution Standard (PTES) [13]	Application Web Réseaux Sécurité physique ingénierie social Réseaux sans fil Effectif Humain	329	Oui	Propose un toolkit	Oui	Oui	Non
PCI Penetration testing guide[14]	Couche Application Couche réseau Ingénierie sociale Segmentation	31	Oui	Non	Oui	Oui	Non
Penetration Testing Framework [15]	Application Sécurité Réseaux Sécurité Réseaux sans fil Sécurité physique	16 grandes étapes et plus de 300 sous étapes	Oui	Propose un toolkit	Non	Non	Non
Technical Guide to Information Security Testing and Assessment (NIST800-115) [16]	Réseaux et Systèmes	55	Oui	Non	Non	Non	Oui
Information Systems Security Assessment Framework (ISSAF) [17]	Système d'information	plus de 300	Oui	Suggère l'utilisation de certains outils	Oui	Oui	Non

Open Source Security Testing Methodology Manual (OSSTMM) [18]	Effectif Humain Sécurité physique Sécurité sans fil Sécurité Réseaux Sécurité Télécommunication Compliance	58	Oui	Non	Non	Non	Non

Tableau 1 : Récapitulatif des approches et méthodes

3. Les étapes de l'approche

1. Mapper le contenu de l'application

Les étapes de cette phase sont montrées dans le schéma suivant :

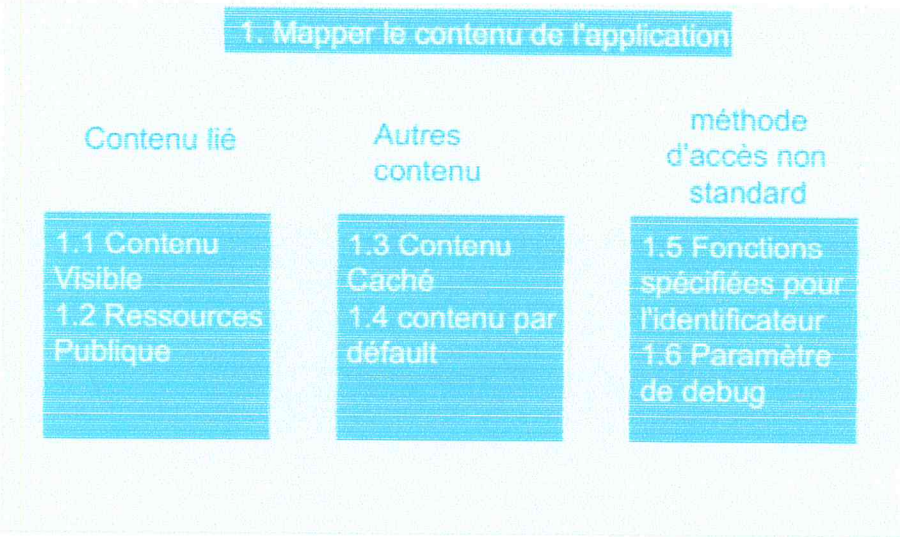


Figure 4 :Etapes mappage le contenu de l'application

1.1. Explorer le contenu visible

Dans cette étape nous devons configurer notre navigateur web pour utiliser des outils comme BurpSuite ou WebScarab pour l'indexation passive du site, de plus nous devons naviguer avec et sans JavaScript sur le site cible et visiter tout les liens, soumettre tout les formulaires, et essayer tout les fonctionnalités proposé par le site, en outre si l'application utilise l'authentification nous devons créer un compte afin d'accéder aux fonctionnalités protégées par le processus d'authentification.

Nous devons surveiller tout le trafic qui passe sur notre proxy afin de comprendre quel type de données est en cours de soumission et s'il y a des contrôles de côté client sur ces données. A la fin de cette étape nous examinons la carte de site générée par les outils d'indexation et identifier tout le contenu que nous n'ayons pas vu sur notre navigateur web.

1.2. Les ressources publique

Dans cette étape nous utilisons plusieurs outils et site web pour se renseigner sur notre application cible, nous pouvons utiliser par exemple The Wayback Machine qui archive les sites web sur internet pour voir quel contenu a été indexé et sauvegardé de notre application cible, nous utilisons aussi Google et ses paramètres de recherche avancés (par exemple site :

pour récupérer tout le contenu sur notre site), il faut aussi faire des recherches sur tous les noms et les adresses mail trouvés sur l'application.

1.3. Le contenu caché

Dans cette étape nous confirmons comment l'application gère les requêtes pour des items qui n'existent pas, nous établissons des requêtes pour des items valides et non valides, et nous observons les réponses pour établir une manière de savoir quand l'item existe et quand il n'existe pas, ensuite nous essayons de comprendre la convention suivi pour nommer les fichiers et les répertoires sur l'application cible, puis nous utilisons des outils d'automation pour envoyer un grand nombre de requêtes basées sur une liste de répertoire, noms des fichiers et les extensions et nous surveillons les réponses de serveur pour voir quelle items sont présents sur l'application.

1.4. Le contenu par défaut

Nous exécutons Nikto ou un autre outil similaire contre le serveur web pour détecter tout contenu par défaut, et nous vérifions par la suite tous les résultats manuellement pour éviter les faux positifs, ensuite nous essayons de demander le répertoire racine en spécifiant l'adresse IP dans l'entête HTTP Host et nous déterminons si l'application répond avec un contenu différent et si c'est le cas nous exécutons Nikto contre l'adresse IP de l'application

1.5. Fonctions spécifier pour l'identificateur

Nous identifions tous les fonctions que nous accédons en passant des identificateurs de fonctions dans les paramètres de requête URL par exemple **/main.php?func=A21.**

Nous appliquons la technique décrite dans le point 1.3 pour accéder à des fonctions individuelles, si par exemple l'application utilise des paramètres qui identifient les noms des fonctions, tout d'abord nous déterminons de l'application quand nous spécifions une fonction qui n'existe pas.

1.6. Les paramètres de debug

Nous choisissons une ou plusieurs pages où les paramètres de debug peuvent être cachés (des noms comme debug, source ou test et des valeurs comme True, 1 ou On), nous itérons à travers toutes les combinaisons possibles des paires nom/valeur et nous soumettrons ces combinaisons pour tous les pages et fonctions cibles, nous surveillons les réponses de l'application par la suite pour toute anomalie qui peut nous indiquer que les paramètres ont un effet sur l'application cible.

2. Analyser l'application

Les étapes de cette phase sont montrées dans le schéma suivant :

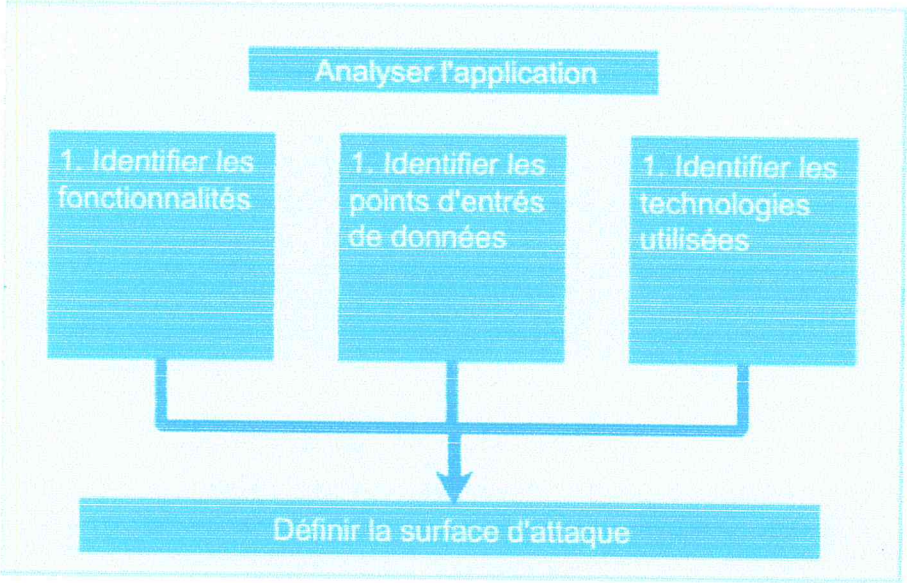


Figure 5 : Résumé des étapes pour analyser l'application

2.1. Identifier la fonctionnalité

Nous identifions le but de l'application (pourquoi elle a été créée) et les actions effectuées par les différentes fonctions de l'application, ensuite nous devons identifier le mécanisme de sécurité déployé par l'application et comprendre son fonctionnement, et comprendre en particulier les différents mécanismes qui gèrent l'authentification, les sessions et le contrôle d'accès, ensuite nous essayons d'identifier les fonctions périphériques comme l'utilisation de redirection, les messages d'erreur et les fonctions administratives et des journaux d'accès.

2.2. Identifier les points d'entrée des données

Tout d'abord nous devons identifier tous les points d'entrée des données où l'utilisateur peut introduire des données à l'application y compris les URL, les données dans les requêtes POST, cookies et tous les entêtes HTTP qui sont traité par l'application, ensuite nous examinons tout transmission personnalisée ou codage de données, enfin nous identifions s'il existe d'autres canaux où les données d'utilisateurs peuvent être passées à l'application comme par exemple une application web de mail qui traite et rend les emails reçu via SMTP.

2.3. Identifier les technologies utilisées

Premièrement nous identifions toutes les technologies utilisées dans le côté client tel que les formulaires, JavaScript, ActiveX et les cookies, puis nous devons identifier les technologies utilisées dans le côté serveur tels que le langage de script utilisé, la plateforme d'application, les bases de données et système qui gère les emails, ensuite nous examinons l'entête HTTP

server retourner dans les réponses de serveur, et nous pouvons utiliser des outils comme Httprint pour avoir une empreinte digitale de serveur, enfin nous devons réviser les résultats obtenues dans l'étape de mappage de contenu de l'application pour identifier tous les fichiers, les répertoires qui peuvent donner des indices sur les technologies utilisées dans le côté serveur.

2.4. Définir la surface d'attaque

Nous essayons de déterminer la structure interne et les fonctionnalités de côté serveur et le mécanisme utilisé afin de livrer le contenu pour le côté client.

Pour chaque item de fonctionnalité identifié nous devons chercher les vulnérabilités qui sont commun pour ces fonctionnalités, et nous formulons un plan d'attaque en donnant priorité aux fonctionnalités les plus intéressantes et les vulnérabilités les plus sérieuses qui sont associées avec ces fonctionnalités.

3. Tester les contrôles de côté client

Les étapes de cette phase sont montrées dans le schéma suivant :

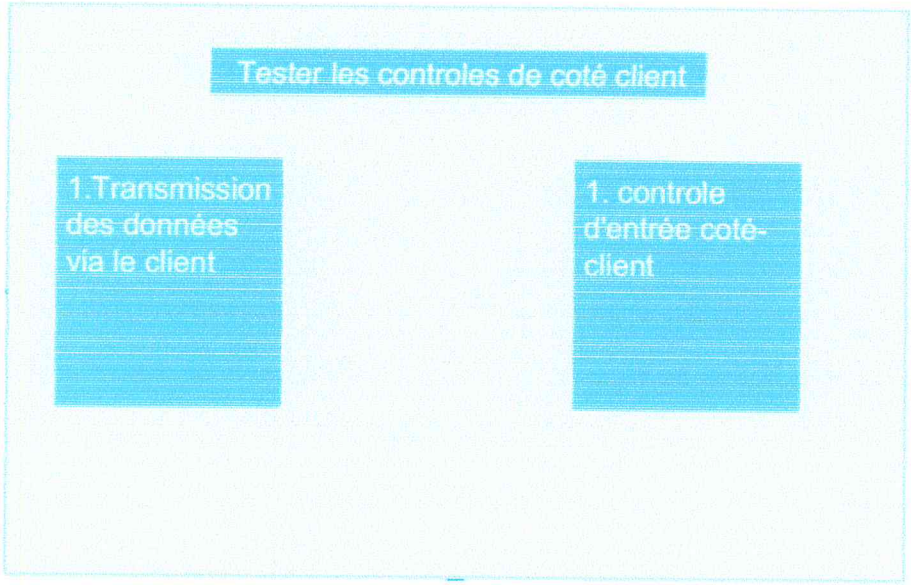


Figure 6: Résumé des étapes pour tester les contrôles de côté client

3.1. Transmission de données via le client

D'abord nous localisons toutes les instances dans l'application qui contiennent des formulaires cachées, des cookies et tout contenu caché, ensuite nous essayons de déterminer le rôle joué par chaque item dans la logique de l'application en se basant sur le contexte de son apparence dans l'application et nous déterminons par la suite si l'application traite des valeurs arbitraires soumis dans les champs des items et si nous pouvons exploiter ses items,

puis si l'application transmit des données opaques via le client, nous pouvons par exemple attaquer l'algorithme de chiffrement utilisé par l'application.

3.2. Contrôle d'entrée côté client

Dans cette étape nous devons identifier tous les cas où il y a des contrôles sur les entrées de client tel que les contrôles de validation JavaScript et de limite de longueur, ces contrôles peuvent être facilement contournés, une fois que nous arrivons à contourner ces contrôles nous vérifions si les mêmes contrôles existent sur le côté serveur.

Nous devons réviser tous les formulaires HTML pour identifier les éléments désactivés par exemple : **<input disabled="true" name="product">**, et nous soumettrons ces formulaires avec d'autres paramètres et nous observons si ces paramètres ont un effet sur le traitement de l'application, nous pouvons utiliser un outil d'automation pour activer tous les éléments désactivés comme les règles de modification HTML de proxy Burp.

4. Tester le mécanisme d'authentification

Les étapes de cette phase sont montrées dans le schéma suivant :

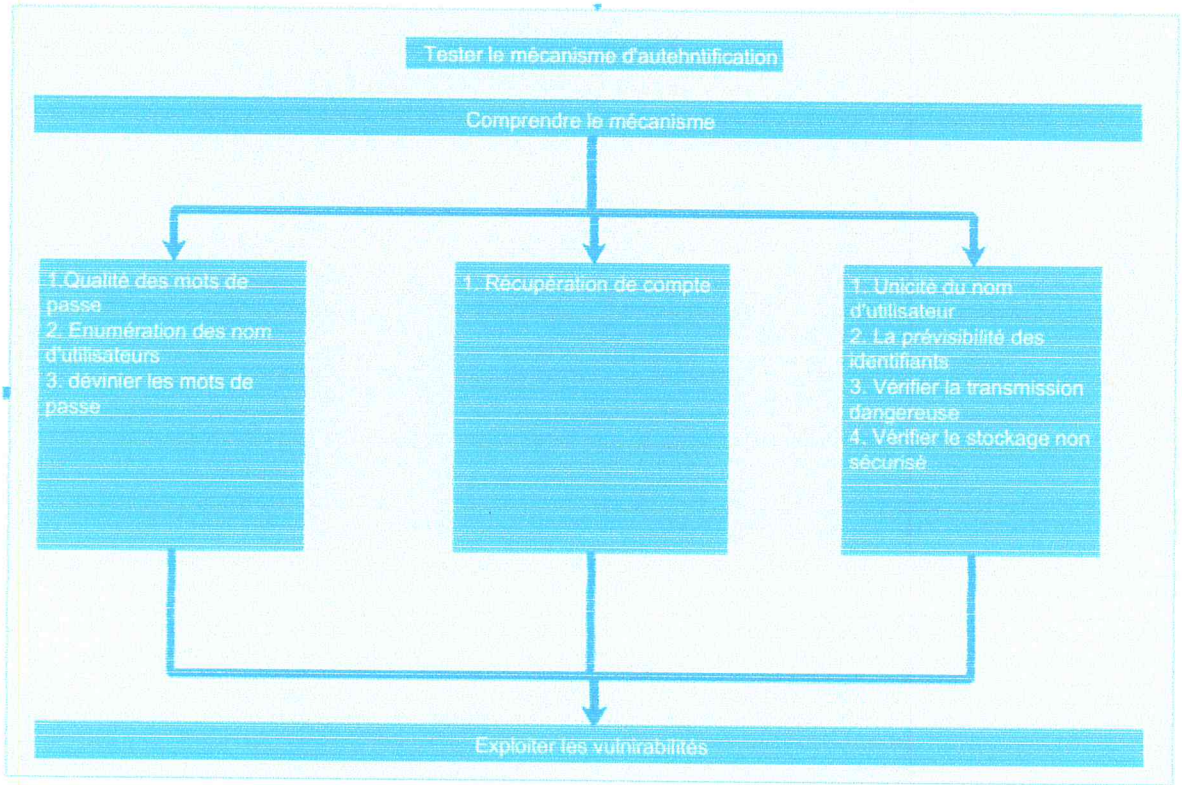


Figure 7: Résumé des étapes Tester le mécanisme d'authentification

4.1. Comprendre le mécanisme

Nous déterminons la technologie d'authentification déployer dans l'application cible tel que les formulaires, les certificats d'authentification et l'authentification multi-facteurs, puis nous devons localiser tous les fonctionnalités d'authentification qui existent sur l'application.

4.2. Tester la qualité des mots de passe

Tout d'abord nous vérifions s'il existe des descriptions de qualité de mots de passe imposé par l'application, ensuite nous essayons d'entrer plusieurs type de mot de passe, nous essayons les mots de passe court, que des caractères alphabétique, caractères unique seulement, mot de dictionnaire ou le nom d'utilisateur comme mot de passe, nous passons par la suite à tester la validation incomplète des mots de passe, nous entrons un mot de passe assez complexe et nous essayons de connecter en utilisons des variations de ce mot de passe en supprimant le dernier caractère, en changeant le caractère de minuscule à majuscule ou le contraire, en supprimant les caractères spéciaux et si l'un de ces essais se connecte nous continuons à essayer systématiquement afin d'apprendre quel validation est effectuée.

Enfin nous essayons de lancer des attaques où nous essayons de deviner les mots de passe.

4.3. Tester l'énumération des noms d'utilisateurs

Nous devons identifier tous les fonctions d'authentification où le nom d'utilisateur est soumis, et pour chaque fonction nous soumettrons deux requêtes une avec un nom d'utilisateur valide et l'autre avec un nom d'utilisateur qui n'existe pas, puis nous examinons les réponses de serveur de chaque requête, nous examinons le code de statuts HTTP, les redirections, les informations affichées sur l'écran, différence dans l'HTML des deux réponses et le temps pris par le serveur pour répondre à chaque requête, nous devons répéter ce travail avec plusieurs noms et examiner toutes les réponses afin de savoir si nous pouvons faire ce travail d'une façon automatique, nous devons suivre toute fuite d'information sur l'application qui nous permet d'énumérer les utilisateurs de cette application.

4.4. Tester la résilience contre l'intuition des mots de passe

Nous commençons par identifier tous les locations dans l'application où les identifiants sont soumis, ensuite dans chaque location nous testons si une politique de blocage de compte après plusieurs tentatives de connexion échouées est implémentée sur l'application ou pas.

4.5. Récupération de compte

Nous identifions tout fonctionnalité qui facilite l'utilisateur à récupérer son compte, nous devons comprendre comment la récupération de compte fonctionne, si la fonctionnalité utilise des défis

comme question secrète, nous essayons d'utiliser des listes de noms en commun ou de deviner les réponses pour contourner la fonctionnalité, si elle inclut l'envoi d'un email pour compléter le processus de récupération de compte nous déterminons s'il y a des faiblesses au niveau de processus d'envoi et si c'est possible de contrôler l'adresse à laquelle l'email est envoyé, si le message continent une URL unique pour la récupération de compte nous obtenons plusieurs messages de récupération et nous essayons d'identifier si il y a un modèle d'URL qui nous permet de prédire les URL émises aux autres utilisateurs.

4.6. Tester l'unicité du nom d'utilisateur

Si l'application a une fonction qui essaye de nous inscrire avec un nom d'utilisateur que nous choisissions, nous essayons d'inscrire avec le même nom d'utilisateur deux fois mais avec des mots de passe différents et si l'application bloque notre essai nous pouvons exploiter cette fonctionnalité pour énumérer les utilisateurs de l'application.

Dans le cas où l'application nous laisse s'inscrire avec le même nom d'utilisateur, nous déterminons le comportement de l'application en cas de collision de nom d'utilisateur et de mot passe, nous essayons de changer le mot de passe d'un des deux compte pour correspondre au mot de passe de l'autre compte, ou nous créerons deux comptes avec les mêmes identifiants, si l'application tolère la collision de nom d'utilisateur et de mot de passe sans générer une erreur, nous connectons avec les identificateurs qui génèrent cette collision et nous déterminons si nous pouvons avoir un accès non autorisé aux compte d'autres utilisateurs.

4.7. Tester la prévisibilité des identifiants auto générés

Si l'application génère des identifiants automatiquement nous essayerons de générer plusieurs identifiants en succession et nous identifions s'il y a des séquences détectables ou si l'application suit un modèle pour générer les identifiants, si nous trouvons que l'application génère des identifiants d'une façon prévisible nous exploitons cette faille en utilisons des identifiants déjà générés par l'application pour gagner accès à des comptes qui nous ne appartiennent pas.

4.8. Vérifier la transmission dangereuse

Nous parcourrons toutes les fonctions qui gèrent l'authentification et qui transmettent les identifiants, nous devons surveiller le trafic entre le client et le serveur, si les identifiants sont transmis dans l'URL l'application est potentiellement vulnérable à la divulgation des identifiants dans l'historique de navigateur ou dans les journaux de serveur, et si les identifiants sont enregistrés dans les cookies l'application est potentiellement vulnérable à la

divulgation des identifiants si une attaque XSS est réussite contre l'application, dans le cas où le serveur transmit les identifiants au clients, l'application est peut être compromis via des vulnérabilités dans la gestion des sessions ou dans le contrôle d'accès ou par une attaque XSS, nous devons vérifier aussi si l'application utilise le chiffrement dans les canaux de transmission des identifiants.

4.9. Tester le stockage non sécuriser

Si nous gagnons accès à des mots de passe hachés ou à des données chiffrées, nous essayerons de craquer ces mots de passes, et de craquer tous les données chiffrées que nous avons accès.

5. Tester le mécanisme de gestion des sessions

Comme toutes les variables super-globales, les variables de session sont également un danger potentiel. Une session mal protégée peut être volée ou empoisonnée (respectivement "sessions hijacking" et "session poisoning").

Les étapes de cette phase sont montrées dans le schéma suivant :

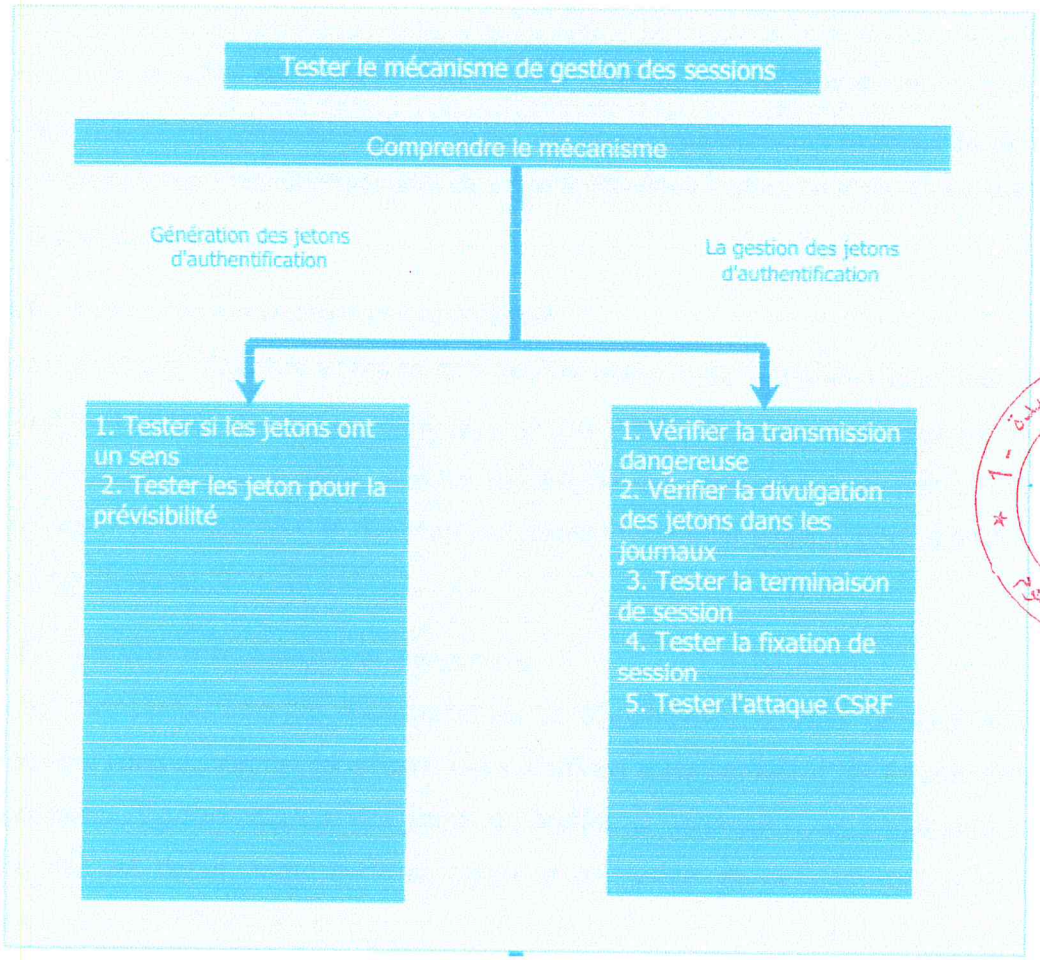


Figure 8: Résumé des étapes pour tester le mécanisme de gestion des sessions

5.5. Tester la divulgation des jetons dans les journaux

Si notre mappage de l'application a identifié des journaux ou de la surveillance nous devons réviser ces résultats et vérifier s'il y a une divulgation des jetons de session, ensuite nous identifions toutes instances où les jetons de session sont transmis dans l'URL.

Si nous trouverons des jetons de session qui appartient à d'autres utilisateurs nous devons déterminer si le type de l'utilisateur (Administrateur ou utilisateur normale).

5.6. Tester la terminaison de session

Nous vérifions si l'expiration des jetons de session est configurée sur le serveur, tout d'abord nous connectons à l'application pour obtenir un jeton valide, ensuite nous attendons une période sans utiliser ce jeton, puis nous soumettrons une requête pour la page protégée en utilisant ce jeton, si la page s'affichera donc le jeton est toujours valide, nous recommençons avec une période plus longue pour déterminer le temps d'expiration de jeton.

Nous vérifiions s'il y a une fonction de déconnexion, si elle existe nous déconnectons de l'application avec cette fonction et nous essayons de nous reconnecter en utilisant l'ancien jeton de session pour tester si la fonction nous déconnecte effectivement.

5.7. Tester la fixation de session

Si l'application génère des jetons de session pour les utilisateurs non authentifiés, nous obtenons un jeton et nous effectuons une connexion, si l'application ne génère pas un nouveau jeton pour nous, alors elle est vulnérable à la fixation de session.

Même si l'application ne génère pas des jetons pour les utilisateurs non authentifiés, nous obtenons un jeton et si l'application nous permet de revenir à la page d'identification, nous soumettrons des nouveaux identificateurs, et si elle ne génère pas un nouveau jeton pour nous elle est vulnérable à la fixation de session.

5.8. Tester l'attaque CSRF

Si l'application repose uniquement sur les cookies HTTP comme méthode de transmission de jetons de session, elle peut être vulnérable aux attaques de falsification de requêtes entre sites CSRF.

Nous créerons une page HTML qui va envoyer une requête souhaitée sans l'interaction de l'utilisateur, par exemple pour une requête GET nous pouvons utiliser la balise HTML **<img>** avec le paramètre **src** défini à l'URL vulnérable, nous utilisons JavaScript pour soumettre la requête dés que la page se charge sur le navigateur et pendant que nous sommes connectés sur

l'application cible, nous vérifions par la suite si l'action souhaitée est effectuée sur l'application cible.

6. Tester le contrôle d'accès

Les étapes de cette phase sont montrées dans le schéma suivant :

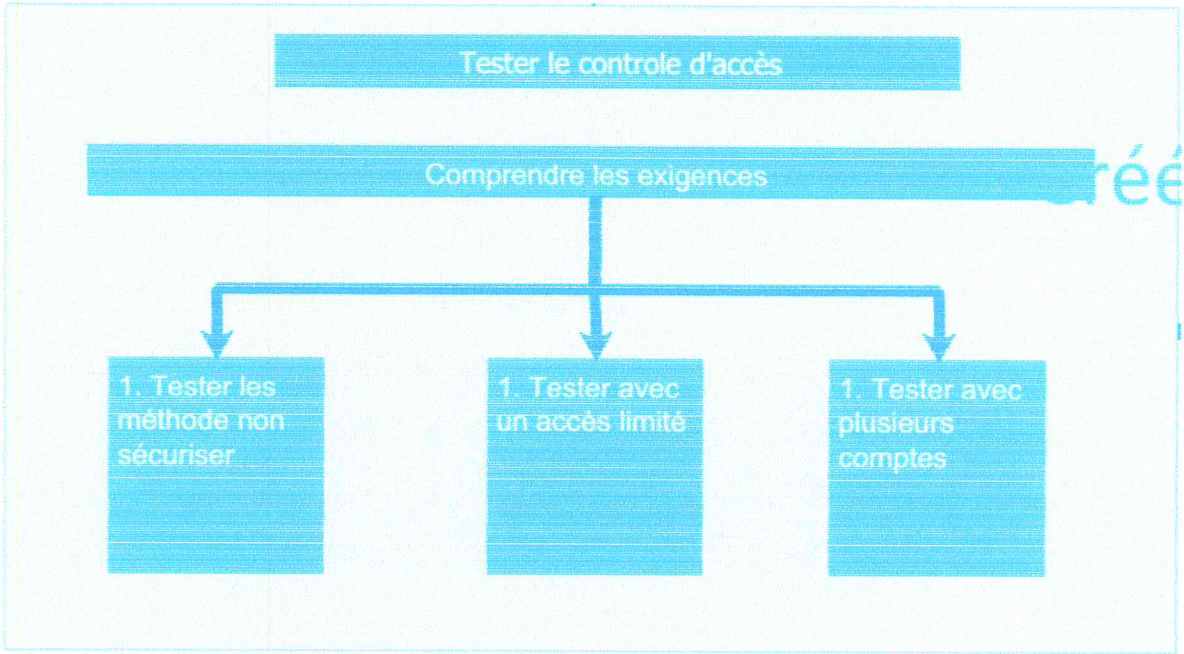


Figure 9: Résumé des étapes pour Tester le contrôle d'accès

6.1. Comprendre les exigences

Nous devons comprendre les exigences de contrôle d'accès de l'application, c'est-à-dire comprendre les différents niveaux d'utilisateur qui ont des fonctionnalités différentes, c'est ce qu'on appelle la ségrégation verticale, nous devons comprendre aussi la ségrégation horizontale; c'est-à-dire les utilisateurs qui ont un même niveau de privilège ont accès au différent sous ensemble de données, ensuite nous révisons les résultats de notre mappage de l'application pour essayer d'identifier les fonctionnalités et ressources qui sont cibles à des attaques d'élévation de privilèges.

6.2. Tester avec plusieurs comptes

Si l'application impose la ségrégation verticale des privilèges, nous utilisons un compte avec des privilèges élevés et localiser tous les fonctionnalités qu'on a accès à, ensuite nous utilisons un compte peu puisant ou normale et nous essayons d'accéder aux même fonctionnalités.

Si l'application impose la ségrégation horizontale nous effectuons un test similaire en essayant d'accéder aux données d'un compte différent de notre compte en remplaçant l'identifiant de la donnée par exemple.

6.3. Tester avec un accès limité

Si nous n'aurons pas accès à des comptes de différents niveaux, le test de contrôle d'accès sera beaucoup plus difficile car nous ne connaissons pas les URL, les identificateurs et les paramètres dont nous aurons besoin pour exploiter les faiblesses.

Généralement les données qui sont sujet au contrôle d'accès horizontal sont accédées via des identificateurs, pour tester si les contrôles d'accès sont effectifs en utilisant un seul compte, nous devons essayer de découvrir les identificateurs des données des autres utilisateurs, si possible nous générons une séquence d'identificateurs en succession, et nous essayons d'identifier si ces identificateurs suivent un modèle précis qui nous permet de prédire les identificateurs des autres utilisateurs.

6.4. Tester les méthodes non sécurisées

Quelques applications implémentent un contrôle d'accès basé sur les paramètres des requêtes, nous devons chercher ces paramètres comme par exemple modifier=faux ou accès=écriture et les modifier pour interférer avec la logique de contrôle d'accès de l'application.

D'autres applications utilisent l'entête **refer** du protocole HTTP pour prendre des décisions de contrôle d'accès, par exemple une application implémente un contrôle d'accès sur la page **/admin.php** et accepte toutes les requêtes qui ont cette URL dans l'entête **refer**, pour tester ce comportement nous essayons d'effectuer des actions privilégiées en modifiant ou en supprimant l'entête **refer**, si l'application bloque notre requête donc l'application utilise probablement cet entête pour le contrôle d'accès, nous essayons d'effectuer les mêmes actions avec un utilisateur non autorisé à cette ressource mais en soumettant l'entête **refer** original qui nous permet d'accéder à cette ressource, et si nous aurons accès, le contrôle d'accès implémenté pour cette application est vulnérable.

7. Tester les vulnérabilités basées sur les entrées

Les étapes de cette phase sont montrées dans le schéma suivant :

7.6. Tester l'inclusion des fichiers

Nous injectons des URL à l'application par exemple **http://adressedenotreserveur**, si nous recevons des connexions HTTP de la part de l'application, l'application est presque certainement vulnérable à l'inclusion des fichiers. Si nous trouverons une vulnérabilité d'inclusion de fichier nous déployons un serveur web qui contient un ou des scripts malicieux et nous utilisons des commandes pour vérifier si notre script est exécuté.

8. Tester les vulnérabilités de serveur d'application

Les étapes de cette phase sont montrées dans le schéma suivant :

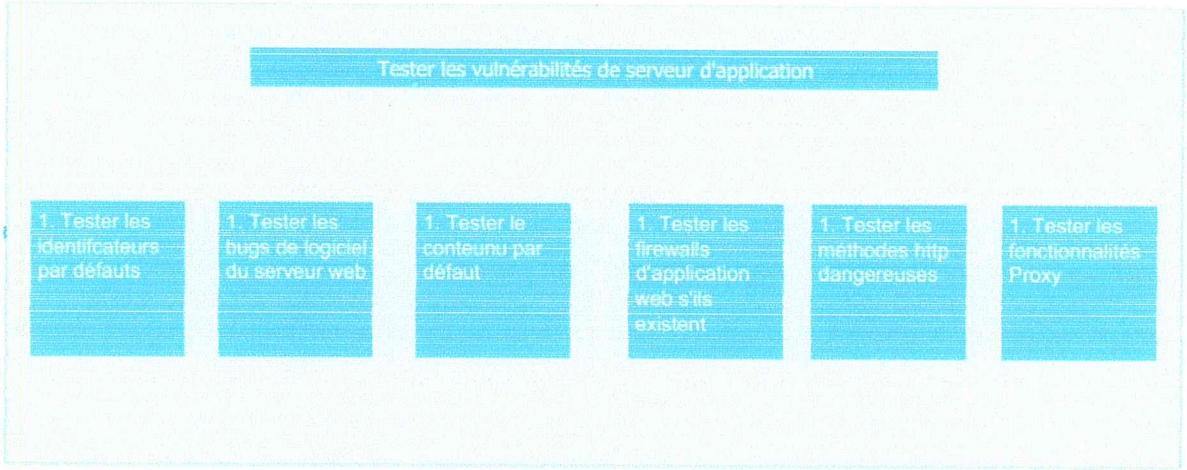


Figure 11 : Résumé des étapes pour tester les vulnérabilités de serveur d'application

8.1. Tester les identificateurs par défaut

Nous examinons les résultats de mappages de l'application, et nous identifions le serveur web utilisé pour héberger l'application et les autres technologies utilisées par l'application qui peuvent avoir une interface administrative, puis nous effectuons un scan de port pour identifier les interfaces administratives qui s'exécutent sur des ports différents que les ports de l'application principale, pour chaque interface identifier nous consultons la documentation de producteur pour obtenir les identifiants par défaut.

Si nous gagnons accès à une interface administrative, nous déterminons si nous pouvons utiliser cette interface pour compromettre le serveur web et/ou l'application.

8.2. Tester les bugs de logiciel du serveur web

Nous utilisons des scanners de vulnérabilités pour identifier les vulnérabilités le logiciel du serveur web, nous devons consulter des ressources comme Security Focus, Bugtraq ou Full Disclosure pour trouver les détails des vulnérabilités qui existe sur le logiciel du serveur web.

8.3. Tester le contenu par défaut

Nous examinons les résultats de scan de Nikto et nous identifions tout le contenu par défaut qui peut être présent sur le serveur, nous utilisons des moteurs de recherche comme exploit-db ou osvdb pour identifier le contenu par défaut ou les fonctionnalités qui sont inclus avec les technologies utilisées pour héberger l'application, nous examinons ce contenu pour déterminer si nous pouvons lancer une attaque contre le serveur via des vulnérabilités qui existent dans le contenu par défaut.

8.4. Tester les méthodes dangereuses du protocole http

Nous utilisons la méthode **OPTION** pour lister les méthodes HTTP disponibles sur le serveur web, sans oublier que différentes méthodes sont activées dans plusieurs répertoires, nous vérifions aussi si les méthodes WebDAV sont activées sur le serveur web.

8.5. Tester les fonctionnalités de proxy

En utilisant les requêtes **GET** et **CONNECT** nous essayons d'utiliser le serveur web comme proxy pour connecter à d'autres serveur sur internet et récupérer de contenu de ces serveurs, ensuite en utilisant les mêmes requêtes nous essayons de connecter à une adresse IP de la même infrastructure de serveur web.

4. Conclusion

Suivre toutes les étapes de cette approche ne nous garantira pas que nous allons découvrir toutes les vulnérabilités au sein d'une Application. Cependant, il nous fournira un bon niveau d'assurance puisque nous avons étudié toutes les régions nécessaires de la surface d'attaque de l'application et on a trouvé autant de problèmes que possible compte tenu des ressources dont nous disposons.

Chapitre IV. Début des tests

1. Introduction

Nous allons commencer à faire les étapes déjà décrites précédemment, étapes par étapes, ensuite tirer des conclusions et expliquer tout les points fait et non fait.

1. Phase de Mappage du contenu de l'application

Nous allons appliquer les étapes de la phase :

1.1. L'exploration du contenu visible

Nous commençons les tests par utiliser BurpSuite qui est composé de différents outils comme robot d'indexation qui permet d'initier des connexions avec l'application web, d'examiner les cookies et d'en parcourir les pages afin d'identifier sa structure interne. Ci-dessous l'image illustre l'url des pages obtenues.

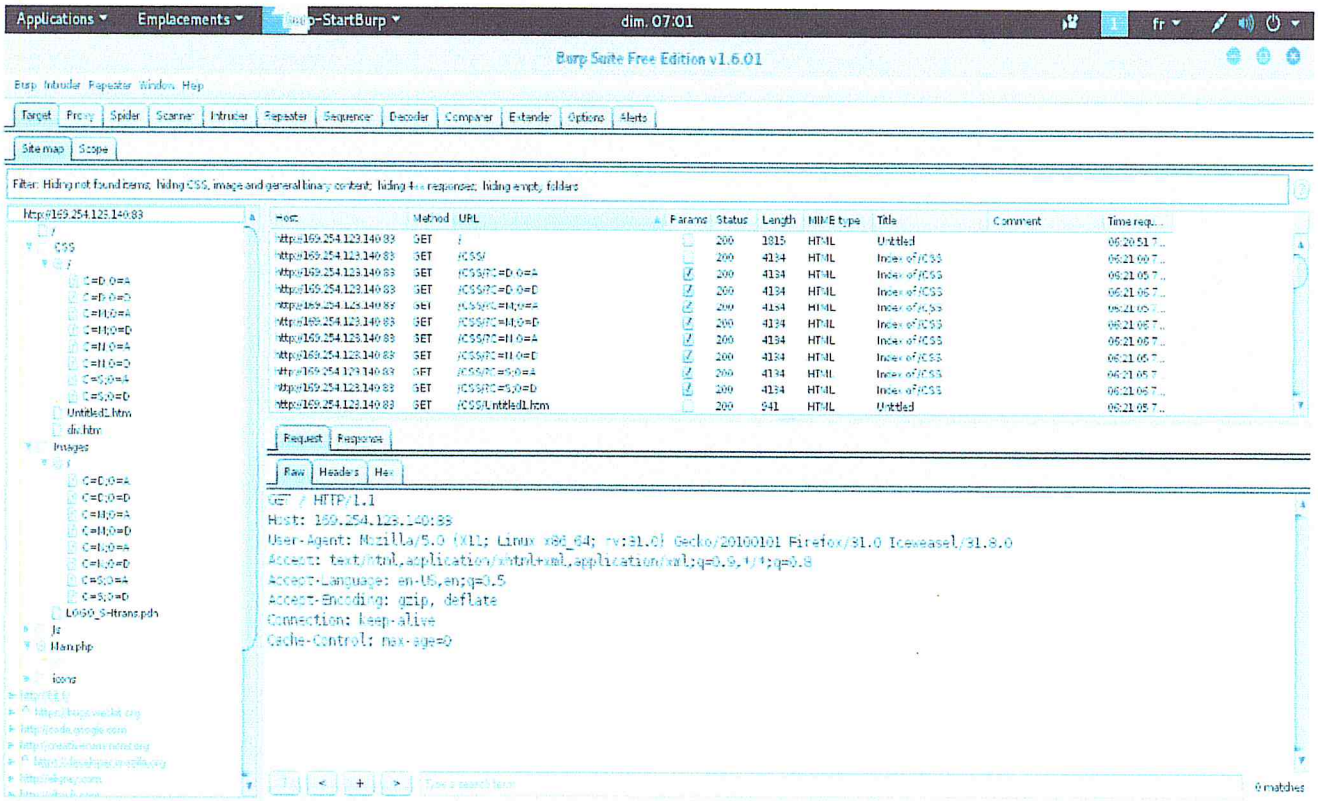


Figure 12: Résultats BurpSuite

La suite des tests se fait en désactivant JavaScript, là nous remarquons que toutes les fonctionnalités du site seront indisponibles, par contre en navigant normalement nous pouvons constater en examinant le code nous pouvons énumérer toutes les pages qui sont disponible pour les utilisateurs normal, et d'autres répertoires qui sont utiliser par des fonctions JavaScript.

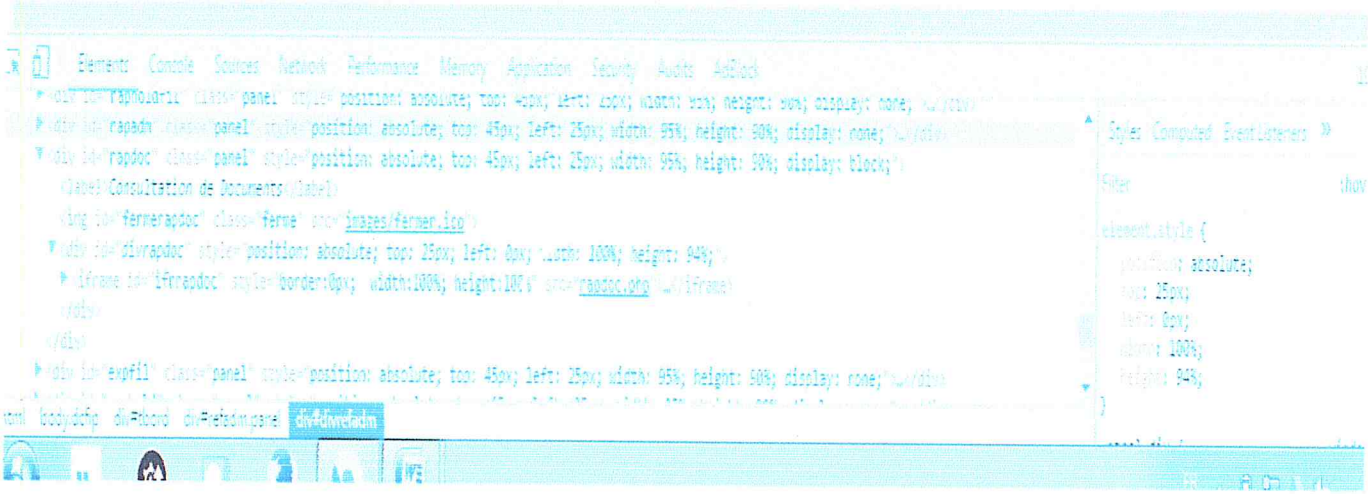


Figure 13 : Examination du code 1

Figure 14 : Examination du code 2

1.2. Exploration du contenu caché

En exécutant un Script qui utilise un dictionnaire des noms des répertoires souvent utilisés par les développeurs web, En lançant des requêtes http pour tous ces répertoires et en examinant le code http retourné par le serveur nous déterminons les répertoires cachés qui existent sur le serveur et la manière dont le serveur réponde à des items qui existent et à des items qui n'existent pas.

Ci-dessous le résultat du script :

```
[200] => http://192.168.1.2:83/main.php
!!! 403 => http://192.168.1.2:83/.htaccess
!!! 403 => http://192.168.1.2:83/.htaccess.php
!!! 403 => http://192.168.1.2:83/.htaccess.bak
!!! 403 => http://192.168.1.2:83/.htaccess.orig
!!! 403 => http://192.168.1.2:83/.htaccess.inc
[200] => http://192.168.1.2:83/index.php
[200] => http://192.168.1.2:83/themes/
[200] => http://192.168.1.2:83/mysql/
[200] => http://192.168.1.2:83/index.php
[200] => http://192.168.1.2:83/ui/
[200] => http://192.168.1.2:83/test.txt
!!! 403 => http://192.168.1.2:83/%3F%3F%3F%3F.txt
!!! 403 => http://192.168.1.2:83/%3F%3F%3F%3F.txt.php
!!! 403 => http://192.168.1.2:83/%3F%3F%3F%3F.txt.bak
!!! 403 => http://192.168.1.2:83/%3F%3F%3F%3F.txt.orig
!!! 403 => http://192.168.1.2:83/%3F%3F%3F%3F.txt.inc
!!! 403 => http://192.168.1.2:83/%3F%3F.txt
!!! 403 => http://192.168.1.2:83/%3F%3F/
```

Figure 15 : Résultat du script

- Pour éviter ce genre de vulnérabilités et contrer ce type de scan, nous proposons d'utiliser des noms de répertoires qui ne sont pas répandus et utiliser htaccess pour refuser l'accès à partir de l'url.

1.3. Le contenu par défaut

Apres une recherche avec l'outil de scanner de vulnérabilités Open Source Nikto avec la commande : **nikto -host 169.254.123.140:83** on obtient un rapport détaillé sur la version du serveur utilisé et les différentes vulnérabilités qui existent

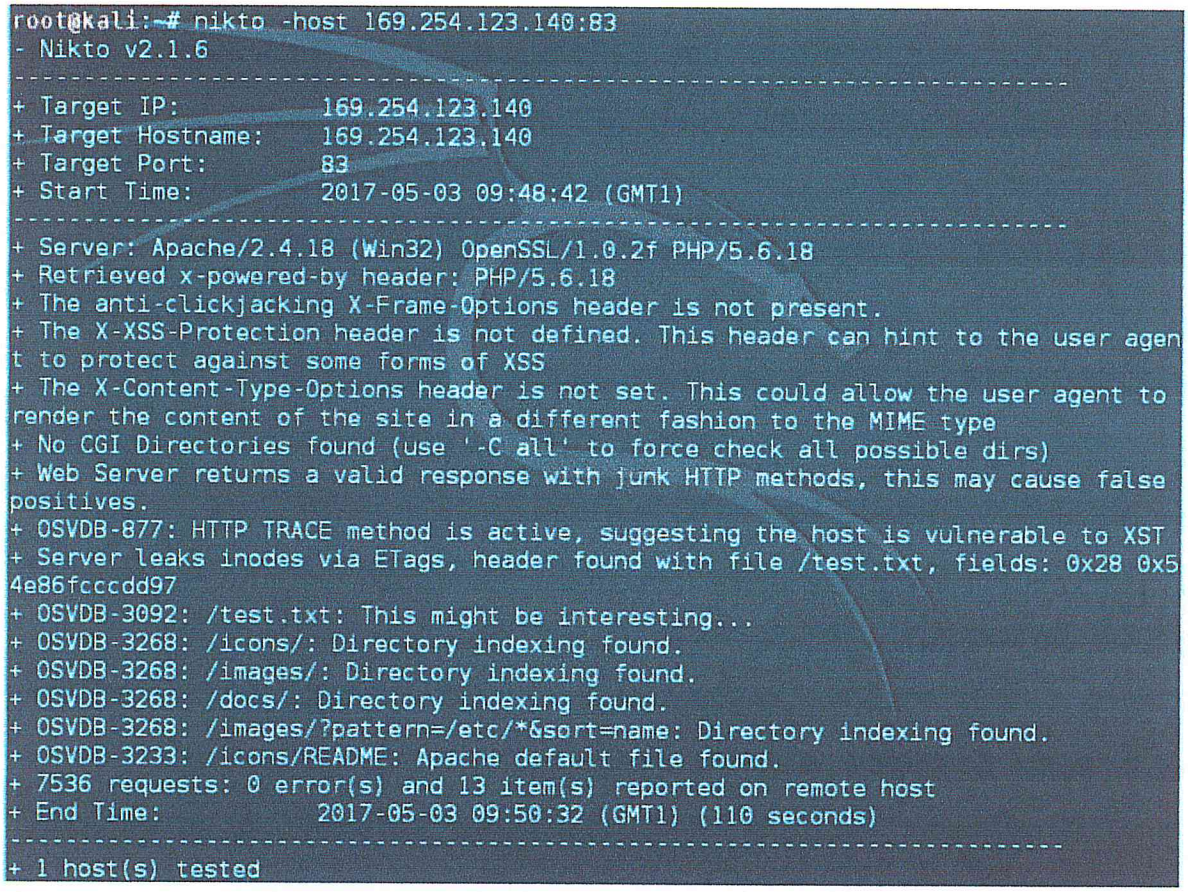


Figure 16 : Résultat de l'outil Nikto

Nous remarquons un code d'identité de la vulnérabilité qui est connu dans la base de données des vulnérabilités suivi d'un url qui nous affiche qu'on a une liste des différents dossiers du site et ça nous permet de connaitre l'architecture du site et potentiellement où sont stockés les différents fichiers confidentiels qui peut être un véritable risque.

Exemple en utilisant l'Url http://169.254.123.140:83/docs/ on pourrait voir tout les fichiers du site qui pourraient être sensible :

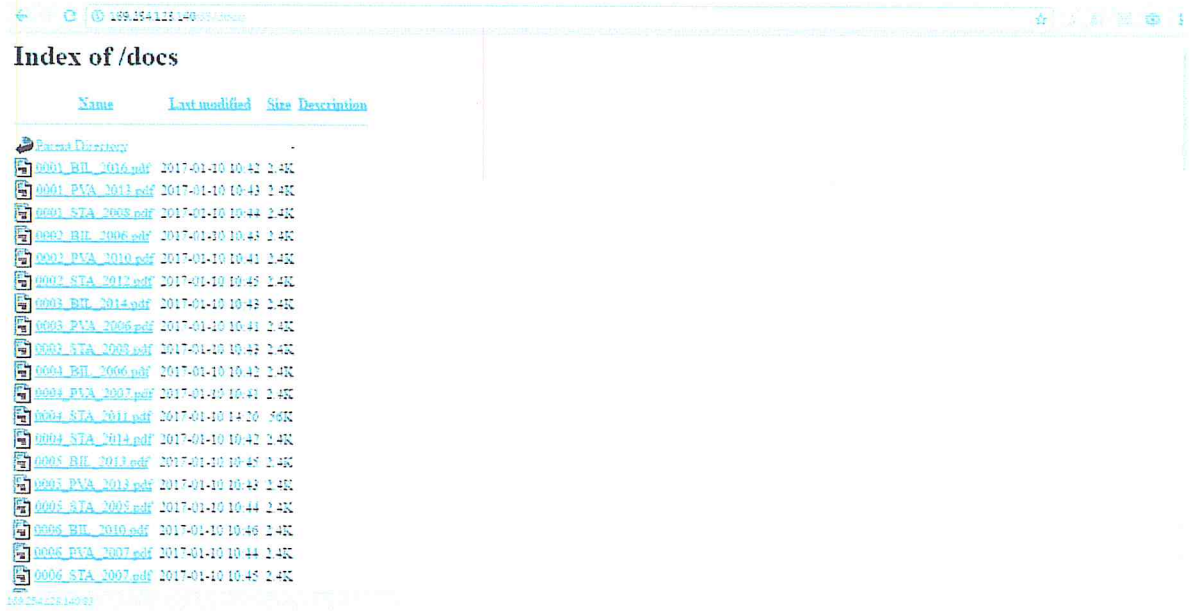


Figure 17 : Résultat recherche par l'Url

- Pour y remédier on a crée un fichier htaccess qui aura la commande **Options - Indexes** qui va bloquer l'utilisation de l'url pour chercher les répertoires du site.

Résultats de la phase de mappage :

Le tableau résume les Url obtenus en parcourant le site fait en première phase des tests

Nom	**URL**
CSS	/CSS/
Images	/Images/
	/images/?pattern=/etc/*&sort=name
Js	/Js/
Js/images	/images/LOGO_SHtrans.pdn
Js/graph	/Js/graph/
Docs	/docs/
icons	/icons/
	/icons/readme

small	/icons/small/
test.txt	/Test.txt
phpMyAdmin	/mysql
phpserver	/phpserver
	/phphserver/rapdocrep.php
Main.php	/Main.php
	Reffil.php
	Refadm.php
	Refdoc.php
	Majhold.php
	Majfil.php
	Majadm.php
	Majmand.php
	Majfin.php
	Raphold.php
	Rapfil.php
	Rapadm.php
	Rapdoc.php
	Expfil.php
	Exphold.php

Tableau 2: Récapitulatif des Url trouvés phase 1

Pour les autres tests de la première phase (Les ressources publique, Fonctions spécifié pour l'identificateur et Les paramètres de debug) nous avons jugé que notre site peut être dispensé d'eux puisque que c'est un site privé qui n'est pas ouvert au grand public réservé juste pour les administrateurs.

2. Analyser l'application

Nous allons appliquer les étapes de la phase :

2.1. Identifier la fonctionnalité du site

Le site est en intranet consacré strictement aux administrateurs qui a pour but de consulter les données de l'entreprise. Son rôle est avant tout de permettre le partage d'information et la communication au sein de l'entreprise. Il offre aux administrateurs la possibilité de produire et de diffuser facilement l'information.

Alors on peut voir une table d'administrateur avec leur code et qualité (Fonction dans le site) ainsi y'aura un contrôle d'accès et des restrictions qui dépend du code administrateur (Le site tel qu'il nous a été donné pour des raisons de confidentialités ne comporte pas de contrôle d'accès ni d'authentification et session).

On trouve une page de «document » où on peut ajouter des fichiers les mettre à jour. Et aussi une page références Filiales et une autre sur les holdings.

2.2 Identifier les points d'entrée des données

Afin d'identifier tous les points d'entrer où l'utilisateur peut introduire des données nous avons fait un tableau qui résume tout ça

Nom de la page	Liste des entrées de chaque page
Références Filiales	Ajouter, Modifier et Supprimer
Références Administrateurs	Ajouter, Modifier et Supprimer
Références Documents	Ajouter, Modifier et Supprimer
Mise à jour des données Holding	Ajouter, Modifier et Supprimer
Mise à jour des données Administrateurs	Ajouter, Modifier et Supprimer
Mise à jour des données Mandats	Ajouter, Modifier et Supprimer
Mise à jour des données financière	Ajouter, Modifier et Supprimer
Consultation de Documents	Ouvrir un fichier, Ajouter un fichier et Supprimer un fichier

Tableau 3: Tableau qui résume les points d'entrée des données

2.3 Identifier les technologies utilisées

Apres une révision de la première phase celle du mappage et après avoir utilisé l'outil httprint, on peut conclure que le site utilise les technologies suivantes :

Apache/2.4.18 (Win32) ; OpenSSL/1.0.2f ; PHP/5.6.18 ; JavaScript ; Json; Ajax ; MySQL 5.7.11; Windows.

Pour contre certain scan d'empreinte de serveur nous devons changer les valeurs suivantes dans le fichier httpd.conf :

ServerTokens Full à ServerTokens Prod

ServerSignature On à ServerSignature Off

Cela limite la réponse de serveur sur les technologies utilisées dans l'entête server de réponse http à Apache au lieu d'Apache/2.4.18 (Win32) OpenSSL/1.0.2f PHP/5.6.18, de plus nous devons refuser toute connexion aux interfaces administratives qui ne viennent pas de localhost comme suit :

Figure 18 : information pour la connexion a la base de données MySQL

De plus nous devons changer les noms par défaut à d'autres personnalisé pour éviter les scans qui utilisent des dictionnaires pour répertorier un site web.

2.4. Définir la surface d'attaque

D'après les résultats des deux phases précédentes nous concluons que l'application est développée sur une pile WAMP (Windows Apache MySQL PHP).

En examinant les fonctionnalités de l'application, nous remarquons qu'il y a la possibilité d'injecter des chaines de caractères malicieuse aux différents point d'entrées de l'application, télécharger des fichiers malveillants sur l'application, et contourner tout type de contrôle côté client.

Donc notre surface d'attaque ce sera tout les formulaires de l'application, toutes les interfaces administratives, le serveur web utilisées et la base de données

3. Tester les contrôles de côté client

Nous allons appliquer les étapes de la phase :

3.4. Transmission de données via le client

L'application ne contient pas des formulaires cachées, ni des cookies, les formulaires de l'application nous permettent d'entrer des données de différent type à la base de données

Nous remarquons qu'il n'existe pas de contrôle ou de validation de données, l'application accepte tous les caractères soumit par l'utilisateur, de plus il n'existe pas un mécanisme de chiffrement de données, l'application transmit les données en claire.

3.5. Contrôle d'entrée côté client

Après avoir identifié toutes les entrées de données possible de l'application dans la phase précédente, nous remarquons que l'application n'applique aucun contrôle de côté client sur les données soumit par l'utilisateur, donc nous pouvons soumettre des chaines de caractères malicieuse à l'application dont le but est de compromettre la sécurité de l'application.

4. Tester le mécanisme d'authentification

Nous allons appliquer les étapes de la phase :

4.1 Comprendre le mécanisme

L'authentification n'existait pas dans la version telle qu'elle nous a été donné, nous avons développé notre propre système d'authentification, on a implémenté deux fonctionnalités d'authentification : la connexion et le blocage, ainsi la connexion se fait en comparant les identifiants (Utilisateur et mot de passe) dans notre base de données.

4.2 Tester la qualité des mots de passe

Nous avons implémenté une politique de mot de passe avec des fonctions de validation qui les acceptent où refusent selon cette politique en exigeant:

-Un mot de longueur de plus de 8 caractères et moins de 20.

-Des lettres minuscules et majuscules.

-Des chiffres et certains caractères spéciaux.

Et après avoir fait tous les tests et combinaisons de la phase (validation incomplète des mots de passe, nous entrons un mot de passe assez complexe …) nous avons conclu qu'aucun mot de passe ne sera accepté par l'application.

Voila ci-dessous le code qui valide les mots de passe :

```
function validatePass($pwd){
    if (preg_match(                                          , $pwd)){
        return true;
    } else {
        echo                                ;
        return false;
    }
```

Figure 19 : Code validation des mdp

4.3 Tester l'énumération des noms d'utilisateurs

Nous avons soumis plusieurs requêtes d'authentification avec des noms d'utilisateurs qui existent sur l'application et d'autres avec des noms qui n'existent pas, la réponse de serveur lorsque nous soumettrons un nom d'utilisateur qui existe est la suivante :

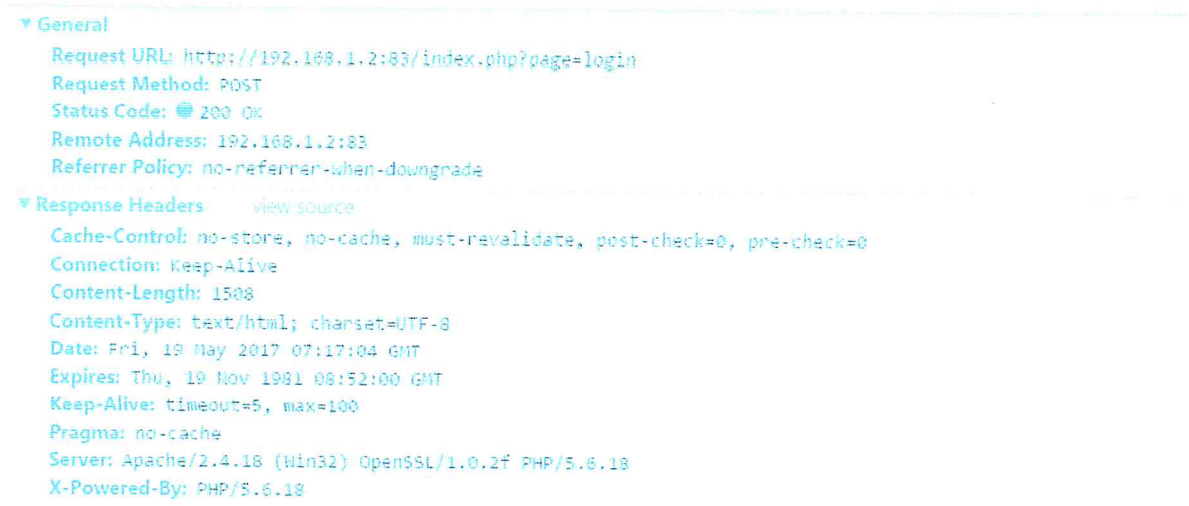


Figure 20 Réponse de serveur quand nous soumettrons un nom d'utilisateurs qui existe

Et la figure suivant nous montre la réponse de serveur lorsque nous soumettrons un nom d'utilisateur qui n'existe pas :

Figure 21 Réponse de serveur quand nous soumettrons un nom d'utilisateur qui n'existe pas

Après avoir examiné les réponses de serveur pour les différents noms d'utilisateurs, nous concluons qu'il n'existe pas de différence entre ces réponses donc il n'existe aucun indice

pour énumérer les utilisateurs de cette application, de plus les deux types de réponses ont le même code HTML.

4.4 Tester la résilience contre l'intuition des mots de passe

Nous avons implémenté une politique de blocage qui n'existait pas au début où un utilisateur qui se trompent ou par malveillance plus de deux fois lors de l'identification il y sera bloqué pendant 15 secondes ceci permet d'éviter toute déduction du nom d'utilisateur ou bien du mot de passe, voilà ci-dessus un aperçu du code :

```
if ((isset($_SESSION[          ])) && ($_SESSION[          ] <  )){
    header(                    );
    echo                                                                    ;
}
elseif ((isset($_SESSION[          ])) && ($_SESSION[          ] >  )){
    echo                   ;
    $_SESSION[          ] =  ; //  initialisation du compteur à 0
    sleep(  );
    header(                    );
    echo                                                                    ;
}
else{
    echo                   ;
    $_SESSION[          ] =  ; //  initialisation du compteur à 0
    sleep(  );
    header(                    );
    echo                                                                    ;
}
}
```

Figure 22 : code blocage

4.5 Récupération de compte

Pour récupérer un compte faut contacter le super administrateur ainsi aucun autre mécanisme ne peut permettre une récupération de compte.

4.6 Tester l'unicité du nom d'utilisateur

L'inscription sur l'application se fait en contactant un administrateur pour vous créer un compte, le formulaire proposé par nous vérifie l'unicité des noms d'utilisateurs avant de créer le compte, lorsque nous essayons d'inscrire avec un nom d'utilisateurs qui existe déjà sur l'application, elle nous répond avec le message suivant : **Le compte existe déjà vous pouvez vous connectez**, donc l'application nous ne laisse pas créer deux comptes avec les même identifiants et évite toute sorte de collision ou d'usurpation d'identité. Voici un aperçu du code qui teste l'unicité du nom d'utilisateur :

```
$bdd = new PDO(                $hostname                ,$username ,$pass);

$query = $bdd->prepare( SELECT * FROM tabadm WHERE codeadm= ? or nom= ?  );
$query->bindValue( ,$cadm,PDO::PARAM_STR);
$query->bindValue( ,$nom,PDO::PARAM_STR);
$query->execute();
$rowCount = $query->rowCount();
```

Figure 23: code qui teste l'unicité d'utilisateur

4.7 Tester la prévisibilité des identifiants auto générés

Selon les résultats de la première phase de mappage nous avons conclus que l'application ne génère pas automatiquement d'identifiants.

4.8 Vérifier la transmission dangereuse

L'application n'utilise pas la version chiffrée du protocole http par conséquent toutes les données sont transmis en claire donc vulnérables aux attaques de l'homme au milieu, nous proposons d'utiliser HTTPS pour chiffrer toutes les données et contrer cette attaque.

5. Tester le mécanisme de gestion des sessions

Nous allons appliquer les étapes de la phase :

5.1. Comprendre le mécanisme

Les sessions est un moyen très sur de communication entre le client et le serveur. Nous avons mis en place un système de "tickets". Le serveur génère un ticket qu'il garde en mémoire. Il le stock ensuite dans les cookies de l'utilisateur. A chaque fois que l'utilisateur demande une nouvelle page, le serveur vérifie qu'ils ont bien le même ticket avant d'en générer un nouveau pour la page suivante. Exemple :

1. L'utilisateur se connecte : début de la session
2. Il se rend sur la page "index.php".
3. Le serveur génère un ticket qui a pour valeur "ticket001"
4. Il l'enregistre simultanément dans une variable de session et dans les cookies de l'utilisateur
5. L'utilisateur change de page
6. Le serveur vérifie qu'ils ont tous les deux le même ticket
7. Il génère un nouveau ticket qui a pour valeur "ticket002" (par exemple)

Nous réalisons dans ce cas que si le hacker vole la session, il se retrouvera avec un ticket aléatoire et donc forcément différent de celui de l'utilisateur. Lorsqu'il essaye de se rendre sur une page, le serveur réalise que les tickets sont différents et la session est détruite.

Ci-dessous le code mécanisme de gestion des sessions que nous avons implémenté :

```
if($num> ){
    if($role==           ){
        $cookie_name =        ;
         // On génère quelque chose d'aléatoire
        $ticket = session_id().microtime().rand( ,          );
        $token =  session_id().microtime().rand( ,          );

        // on hash pour avoir quelque chose de propre qui aura toujours la même forme
        $ticket = hash(        , $ticket);
        $token =  hash(        , $token);

        // On enregistre des deux cotés
        setcookie($cookie_name, $ticket, time() + (   *   ));
        setcookie(        , $token, time() + (   *   ));          // Expire au bout de 20 min
        $_SESSION[        ]= $token;
        $_SESSION[        ] = $ticket;

        $_SESSION[      ]=$user;

        header(                                  );
      exit();
    }
        else {
            $cookie_name =        ;
              // On génère quelque chose d'aléatoire
            $ticket = session_id().microtime().rand( ,          );

            // on hash pour avoir quelque chose de propre qui aura toujours la même forme
            $ticket = hash(        , $ticket);

            // On enregistre des deux cotés
            setcookie($cookie_name, $ticket, time() + (   *   )); // Expire au bout de 20 min

            $_SESSION[        ] = $ticket;
          //$_SESSION['user']=$user;

            header(                                  );
          exit();
```

Figure 24:code mécanisme de gestion des sessions

5.2. Tester si les jetons ont un sens

Après avoir connecté avec plusieurs comptes dans des temps différents nous avons collecté les jetons suivants pour voir s'il y a une corrélation entre eux :

1ff118fe7fc3b6db88480363545b7736af45c5e27c686f3f1ef59d1bd734b8dc91d89f4fe5cf951ea97f2fa122d35c0faf3894693ebbee74237aa5db96afe03a

29c7e8e86d96560f711cc83ff8069c19e2204aec88ded82feed3e44b365c953c397a8fca9ca010e178eae498ce836e4ccf4e83a5ad7c59f47aa356f241c630e9

78bff7af73f9ed080a85c11933e646576d977379908f536507c25681a72b4370e57a3406e40d45e8009038b814ed502622e5d5208d286212ed6988b481c45580

1da1830cfc52fabfb195e56d5fe58196c9446151a7b72d9dff4877a7c3a40dd50cf7e72e0c9fac6533005f7882bae3bc9562a1868f0adcb5aff865f9263a471c

Nous remarquons que les jetons ne suivent pas un modèle précis, mais ils ressemblent à des haches, nous utilisons l'outil Hash-Identifier pour essaye d'identifier quelle type de hachage est utilisé, nous obtenons le résultat suivant pour les quatre jetons :

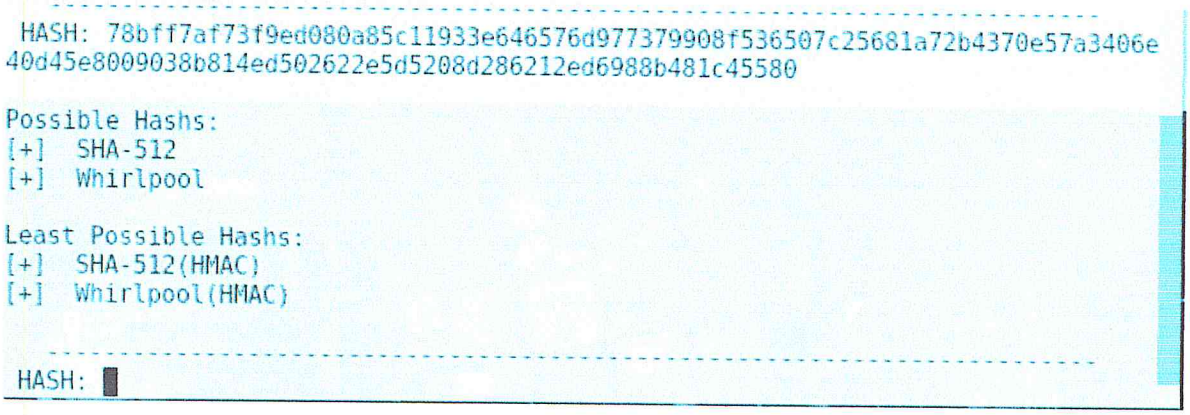


Figure 25 : Résultat de l'outil Hash-identifier

Nous constatons que l'algorithme de hachage utilisé est SHA-512 (ce qui est vrai), qui est un algorithme de hachage puissant, ainsi un attaquant peut prendre plusieurs jours pour le cracker, nous remarquons aussi qu'il n'existe pas de relation avec le nom d'utilisateur et la longueur de hache.

5.3. Tester les jetons pour la prévisibilité

Comme nous avons dit dans la section précédente l'application ne suit pas un modèle pour la génération des jetons mais elle hache ces jetons en utilisant SHA-512, afin d'essayer de prévenir les jetons il faut d'abord les crackers, comprendre leur structure et essayer de générer des jetons similaire qui peuvent contourner la gestion de session implémenté par l'application.

5.4. Vérifier la transmission dangereuse

Les cookies sont en clair alors les jetons risques d'être interceptés par des outils de sniffing donc nous proposons d'utiliser un mécanisme de chiffrement (https) pour éviter tous les risques.

5.5. Tester la divulgation des jetons dans les journaux

Selon la première phase de mappage et l'analyse de l'application nous n'avons pas trouvé de fichiers de journalisations d'accès et de jetons, donc récupérer les jetons à partir des fichiers logs est une tâche impossible.

5.6. Tester la terminaison de session

Nous avons proposé une gestion de session qui utilise des jetons qui expire dans une durée de vingt minutes après leur génération.

Nous avons testé notre fonctionnalité de déconnexion afin de savoir si les jetons générés peuvent être utilisés à nouveau après avoir déconnecté, nous avons utilisé Burp Suite pour intercepter les requêtes et de capturer les jetons de sessions comme suit :

GET /index.php?page=Main HTTP/1.1

Host: 192.168.1.2:83

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Referer: http://192.168.1.2:83/index.php?page=login

Cookie:PHPSESSID=truokvs66pf5eta67pdp7j5qm6;

ticket=b439e1e8e039d9405c68977bf6a7504b828ef98e0402e510804c17a9b316a2155f0cf41c372e5b465f7d24acd658b7c92fe3ce26efa774ec31de4beced64b76c;

token=0e4407d759e1f559862b53f4f8d20f0f4b714026f949c3fc0dde0cbc4b9a33aee8d36130ce21e0861d2f66f945b41a8f772d2cb5e5f3636524a1f8611167dd1f

Connection: close

Ensuite nous avons déconnecté de l'application et essayé d'accéder à la page Main en utilisant les mêmes jetons comme suit :

GET /index.php?page=Main HTTP/1.1

Host: 192.168.1.2:83

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Cookie: PHPSESSID=3idn62sk5jotq5m325ig10h1b0;

ticket=b439e1e8e039d9405c68977bf6a7504b828ef98e0402e510804c17a9b316a2155f0cf41c372e5b465f7d24acd658b7c92fe3ce26efa774ec31de4beced64b76c;

token=0e4407d759e1f559862b53f4f8d20f0f4b714026f949c3fc0dde0cbc4b9a33aee8d36130ce21e0861d2f66f945b41a8f772d2cb5e5f3636524a1f8611167dd1f

Connection: close

Nous serons redirigés vers la page index sans avoir accès à la page main.

Voici un aperçu du code qui termine la session correctement au bout de vingt minutes :

```
<?php
session_start();
if (ini_get(                                    )) {

    $params = session_get_cookie_params();

    setcookie(session_name(),     , time() - 42000,
        $params[      ], $params[       ],
        $params[      ], $params[       ]
    );
    setcookie(      ,     , time() - 42000);
    setcookie(      ,     , time() - 42000);

}

session_destroy();
header(                                         );
exit();
?>
```

Figure 26: code de terminaison de session

5.7. Tester la fixation de session

Nous avons proposé un mécanisme de gestion qui ne génère pas de jetons avant d'être connecté, donc nous avons connecté avec un compte et vérifié si l'application nous génère des nouveaux jetons une fois nous connectons avec un autre compte, l'application nous a donné les jetons suivants avec le premier compte :

Name	Value	[	Path	Expir...	Size
Request Cookies					**36**
PHPSESSID	lqtdoalhnc2rh34f7f6jps2qs2	M	N/A	N/A	36
Response Cooki...					**376**
ticket	8b93730fedecc2eddb1d166ac869cc81f8d6f1d737cc2509ab92a56a3d273084f7963bcbeed...			20.0 ...	189
token	b6c3f7870de58c1253c30d9f61eb6a28e206cf82a25d81b933c03a7a299d8c61f4abc4bc4ca5...			20.0 ...	187

Figure 27: test de génération de jetons1

Ensuite nous connectons avec un autre compte sans avoir déconnecté avec le premier compte et nous aurons les résultats suivants :

Name	Value	Do...	P...	Expir...	Size
Request Cookies					**309**
PHPSESSID	lqtdoalhnc2rh34f7f6jps2qs2	N/A	N...	N/A	38
ticket	8b99730fedecc2eddb1d166ac869cc81f8d6f1d737cc2509ab92a56a3d273084f7963bcbeed...	N/A	N...	N/A	137
token	b6c3f7870de58c1253c30d9f61eb6a28e206cf82a25d81b933c03a7a299d8c61f4abc4bc4ca5...	N/A	N...	N/A	134
Response Cooki...					**376**
ticket	ad8d197b805b8555dd4232edd202394ba4e2f29585c771fa1aee0c73aabcbec9be7b17e055...			20.0 ...	189
token	0fd83ac89c54e4b696be813d7869693f3d6089540c42a7679fe3c260cb48207b48a83543c618...			20.0 ...	187

Figure 28:test de génération de jetons2

Donc elle génère de nouveaux jetons différents pour chaque nouvelle connexion.

5.8. Tester l'attaque CSRF

Grace à l'Authentification par les jetons (token). L'utilisateur aura un jeton unique qui sera vérifié à chaque modification, ainsi même si un user « admin » qui n'a pas certaines fonctionnalités envoie un url au « superadmin » qui fera un travail non autorisé, la session sera détruite puisque les jetons ne sont pas identiques.

6. Tester le contrôle d'accès

Nous allons appliquer les étapes de la phase :

6.1. Comprendre les exigences

Dans l'application web on a deux niveaux d'utilisateurs un supérieur qu'on l'a appelé « superadmin » qui aura le plein pouvoir sur le site qui peut créer des utilisateurs de son niveau ou bien inférieur, ajouter des fichiers, naviguer dans tout le site...etc. et un autre inferieur « admin » qui aura quelques fonctionnalités en moins et des pages qui ne pourra pas y accéder. Voici comment marche le contrôle d'accès pour le « superadmin » :

```
<?php


if (isset($_COOKIE[        ]) == isset($_SESSION[        ]) && isset($_COOKIE[        ]) == isset($_SESSION[        ]) && $_SESSION[        ]!=
    && $_SESSION[        ]!=  )
{
```

Figure 29 : Code contrôle d'accès

Ainsi il faut que l'utilisateur « superadmin » ait les deux jetons (ticket et le token) pour avoir accès à la page. Et « admin » aura besoin d'un seul jeton.

6.2. Tester avec plusieurs comptes

En utilisant un compte avec des privilèges élevés par rapport aux autres comptes qui ont moins de privilèges nous trouvons des différences sur le fait qu'avec des « superadmin » nous pourrons ajouter des comptes utilisateurs les supprimer accéder aux pages de paramétrages (Références Filiales, Références Administrateurs, Références Documents) que l' « admin » ne pourra pas faire.

6.3. Tester avec un accès limité

En utilisant un compte d'administrateur normal nous avons essayé d'accéder aux fonctionnalités des super administrateurs, nous avons essayé d'utiliser le même jeton pour le champ ticket et token afin d'essayer de contourner le contrôle d'accès utilisé par l'application, tout d'abord nous connectons avec un compte administrateur, ensuite nous avons soumis une requête pour une page réservé aux super administrateurs en ajoutant un jeton de nom token à notre cookie comme suit :

Cookie:PHPSESSID=2b8hhvkftg46q6gsp4cnc1r6h2;
ticket=d41f40461189f97a1d9921388b08626903044fbcc6866e9c9b057286d6519169187a09
03b7d1d7c0475f054b451c722e1a8f7f4092780700f27d985b586d298f;
token=d41f40461189f97a1d9921388b08626903044fbcc6866e9c9b057286d6519169187a09
03b7d1d7c0475f054b451c722e1a8f7f4092780700f27d985b586d298f

L'application nous redirige vers la page indexe, donc le mécanisme de contrôle d'accès implémenté par l'application n'est pas vulnérable à ce type d'attaque.

6.4. Tester les méthodes non sécurisées

Nous avons proposé un contrôle d'accès simple qui se repose sur les jetons d'authentification, sans avoir besoin d'implémenter des méthode qui donne l'accès aux ressources, de plus lorsque nous utilisons un compte de type super administrateur et nous demandons une ressource qui exige des privilège, nous pouvons accéder à cette requête avec ou sans l'entête refer, donc nous concluons que l'application n'utilise pas ce type méthode de contrôle d'accès.

7. Tester les vulnérabilités basées sur les entrées

Nous allons appliquer les étapes de la phase :

7.1. Tester Injection SQL

Nous avons essayé d'injecter tous les points d'entrées de données précédemment identifiées dans la phase de mappage de l'application, nous observons que l'application utilise des requêtes POST pour envoyer les données, donc nous avons intercepté ces requêtes en utilisant

BurpSuite, ensuite enregistrer la requête dans un fichier texte et nous avons utilisé ce fichier pour l'injection automatique avec l'outil sqlmap, la commande suivante nous permet d'automatiser le processus : **sqlmap -r sqlmap.txt -p coderfil** on a obtenu les mêmes résultats pour les tous les champs.

[21:57:22] [INFO] parsing HTTP request from 'sqlmap.txt'

[21:57:22] [INFO] testing connection to the target URL

[21:57:22] [INFO] testing if the target URL is stable

[21:57:23] [INFO] target URL is stable

[21:57:23] [WARNING] heuristic (basic) test shows that POST parameter 'coderfil' might not be injectable

[21:57:23] [INFO] testing for SQL injection on POST parameter 'coderfil'

[21:57:23] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'

[21:57:23] [WARNING] reflective value(s) found and filtering out

[21:57:23] [INFO] testing 'MySQL >= 5.0 boolean-based blind - Parameter replace'

[21:57:23] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'

[21:57:23] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'

[21:57:23] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'

[21:57:23] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'

[21:57:23] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace (FLOOR)'

[21:57:23] [INFO] testing 'MySQL inline queries'

[21:57:24] [INFO] testing 'PostgreSQL inline queries'

[21:57:24] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'

[21:57:24] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'

[21:57:24] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'

[21:57:24] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'

[21:57:24] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'

[21:57:24] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'

[21:57:24] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'

[21:57:24] [INFO] testing 'Oracle AND time-based blind'

[21:57:24] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'

[21:57:24] [WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS. You can try to explicitly set it with option '--dbms'

[21:57:25] [WARNING] POST parameter 'coderfil' does not seem to be injectable

[21:57:25] [CRITICAL] all tested parameters appear to be not injectable. Try to increase '--level'/'--risk' values to perform more tests. Also, you can try to rerun by providing either a valid value for option '--string' (or '--regexp'). If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could retry with an option '--tamper' (e.g. '--tamper=space2comment')

Nous concluons que l'application utilise les requêtes préparées ou d'autres mécanismes de protection contre l'injection SQL.

7.2. Tester les attaques XSS

Nous avons testé les deux types d'attaques XSS

7.2.1. Tester les attaques XSS reflétées

Nous soumettrons tout d'abord des chaines de caractères aux entrées des données identifiées dans la phase de mappage de l'application et nous examinons si elles sont retournées dans la page HTML.

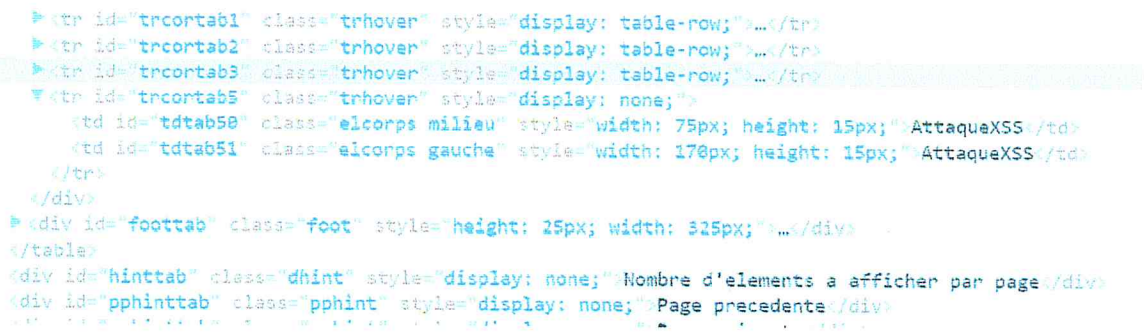


Figure 30 :Examination code XSS

Nous avons injecté ensuite un simple JavaScript **<script>alert('attaqueXSS') ;</script>** nous avons eu le résultat suivant :

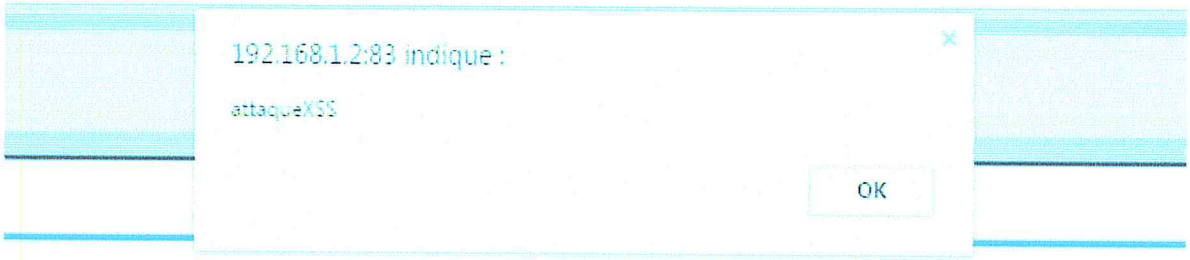


Figure 31 : Résultat test XSS reflétée

Notre attaque a été couronnée de succès, nous avons conclu que l'application n'utilise aucun mécanisme de défense contre les attaques XSS reflétées.

7.2.2. Tester les attaques XSS enregistrées

Après avoir identifié les fonctionnalités de l'application dans les étapes précédentes nous savons que l'application enregistre les données entrées par les utilisateurs et les affiche pour ces derniers, donc il y une forte possibilité de vulnérabilités XSS enregistrés.

Nous commençons par injecter un script **<script>alert('attaqueXSS') ;</script>**, se déconnecté après et se reconnecter avec un autre utilisateur, nous avons obtenu le résultat suivant :

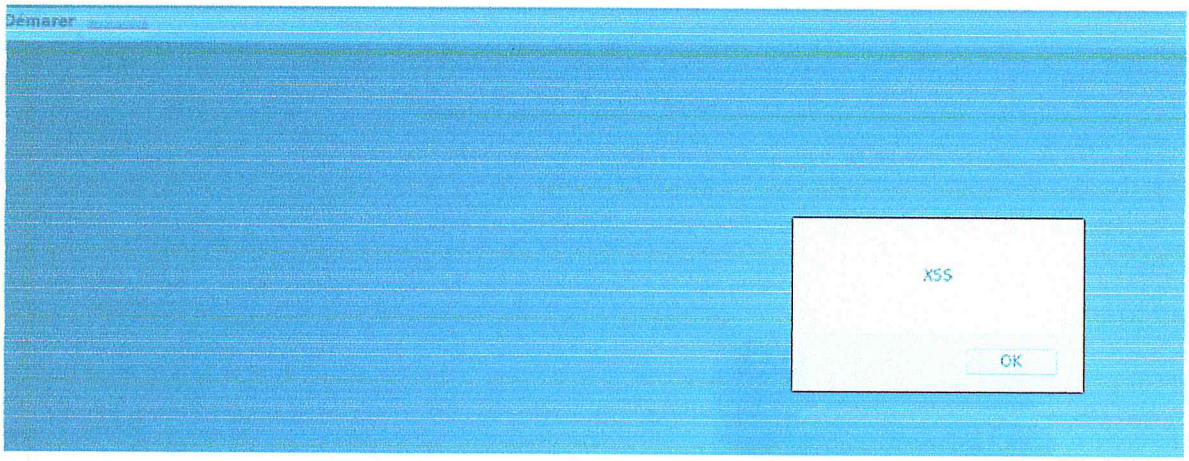


Figure 32 :Résultat test XSS enregistrée

Nous concluons que l'application est vulnérable aux attaques XSS enregistrées, et nous pouvons utiliser des scripts pour par exemple voler les cookies des utilisateurs

7.3. Tester l'injection des commandes de système d'exploitation

Nous avons injecté les commandes suivantes dans tous les points d'entrées de l'application :

|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &

| ping –i 30 127.0.0.1 |

| ping –n 30 127.0.0.1 |

& ping –i 30 127.0.0.1 &

& ping –n 30 127.0.0.1 &

; ping 127.0.0.1 ;

%0a ping –i 30 127.0.0.1 %0a

` ping 127.0.0.1 `

Et nous remarquons que l'application enregistre ces commandes dans sa base de données sans les exécutées, donc nous concluons que l'application n'exécute pas les commandes et elle n'utilise pas les fonctions d'exécution de commandes tel que **exec** par exemple.

7.4. Tester l'attaque de parcours de répertoire

Nous avons injecté attaque de parcours de répertoire contre l'application, nous avons essayé les attaques suivantes :

C:/boot.ini

C:/WINDOWS/php.ini

C:/WINDOWS/Repair/SAM

C:/Windows/repair/system

C:/Windows/repair/software

C:/Windows/repair/security

C:/WINDOWS/System32/drivers/etc/hosts

C:/Windows/win.ini

C:/WINNT/php.ini

C:/WINNT/win.ini

C:/xampp/apache/bin/php.ini

C:/xampp/apache/logs/access.log

C:/xampp/apache/logs/error.log

C:/Windows/system32/config/AppEvent.Evt

C:/Windows/system32/config/SecEvent.Evt

C:/Windows/system32/config/default.sav

C:/Windows/system32/config/security.sav

C:/Windows/system32/config/software.sav

C:/Windows/system32/config/system.sav

C:/Program Files/MySQL/MySQL Server 5.1/my.ini

A chaque soumission d'une attaque l'application nous redirige vers la page index, et c'est pour cela, nous avons conclu que l'application n'est pas vulnérable aux attaques de parcours de répertoire.

7.5. Tester l'injection des scripts

Nous avons injecté plusieurs scripts dans les différentes entrées identifiées dans la phase de reconnaissance, nous avons remarqué que l'application n'exécute pas les scripts mais elle les enregistre dans sa base de données. Sauf pour une seule entrée, celle ou on peut télécharger des fichiers sur l'application, l'entrée est la suivante :

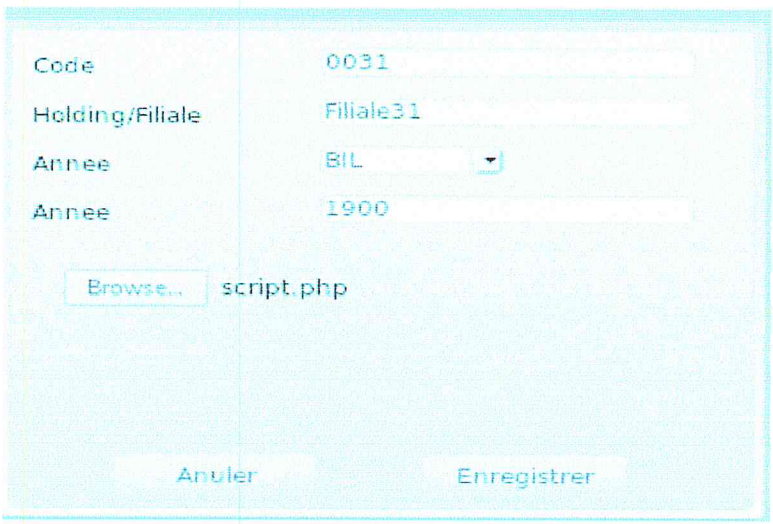


Figure 33 : injection Script

Nous avons crée et télécharger un fichier script.php (simple echo 'injection de Script' ;) sur l'application, nous remarquons par la suite qu'il y a un nouveau fichier dans le dossier de Filiale 31 avec le nom de 031_BIL_1900, nous avons intercépté la requête d'ouverture de ce fichier, la requête est la suivante :

GET /docs/0031_BIL_1900.pdf HTTP/1.1

Host: 192.168.1.2:83

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Referer: http://192.168.1.2:83/rapdoc.php

Cookie: PHPSESSID=7619kvlahmfofojlt898o40gq2; ticket=ebd210f8026b58fc4fa8737c16ba13506e980b07912136c83a3a2b8841b68f77605381f6cb5ebcfb5417dd6a1a2c98d21d1b4a122b249c82f570679cdfa34657; token=2b6dbef7e8007c73e9deb16e9f282ebf6d2be3e5a06818dd5aa7e6f31fd1e412fedee80ca97e48e3698d86d65e9ee43e252e2683f2ed958d8955474b48b7aafe

Connection: close

En modifiant la requête GET par la suivante : **GET /docs/0031_BIL_1900.php HTTP/1.1**

Nous exécutons notre script sur le serveur :

Figure 34: injection script 2

Nous concluons que l'application est vulnérable à l'injection des scripts arbitraires.

7.6. Tester l'inclusion des fichiers

Pour tester cette faille si elle existe il suffit juste de passer en variable dans l'url de notre site un autre site n'importe le quel exemple :

http://169.254.123.140:83/index.php?page=http://google/index.html

La page nous redirigera directement vers la page d'accueil (identification) ce qui nous prouve que la faille n'existe pas. Tout cela grâce à un code qui comparait et filtrent dans un tableau les pages qui existent et celles qui n'existent pas de l'extérieur du site et même ceux caché éventuellement dans le site. Voici un aperçu du code qui est considéré comme la meilleure solution pour contrer la faille include :

```
$page = htmlentities($_GET[ ]);
if !in_array($page.      ,$dir)){
    header(                                    );
    exit();
}
include(          .$page.          );
?>
```

Figure 35 : code contre l'inclusion des fichiers

8. Tester les vulnérabilités de serveur d'application

Nous allons appliquer les étapes de la phase :

8.1. Tester les identificateurs par défaut

Lors de la phase de mappage de l'application nous avons découvert l'interface administrative phpmyadmin sur le lien **192.168.1.2 :83/MySQL**, après avoir cherché les identifiants par défaut de ce produit sur la page de son producteur, nous avons eu accès à cette interface en utilisant ces identifiants (Utilisateur : root, mot de passe : root), donc le développeur de cette application n'a pas changé les identificateurs par défaut, et il n'a pas rejeté les connexions à cette interface à partir de toutes les adresses IP.

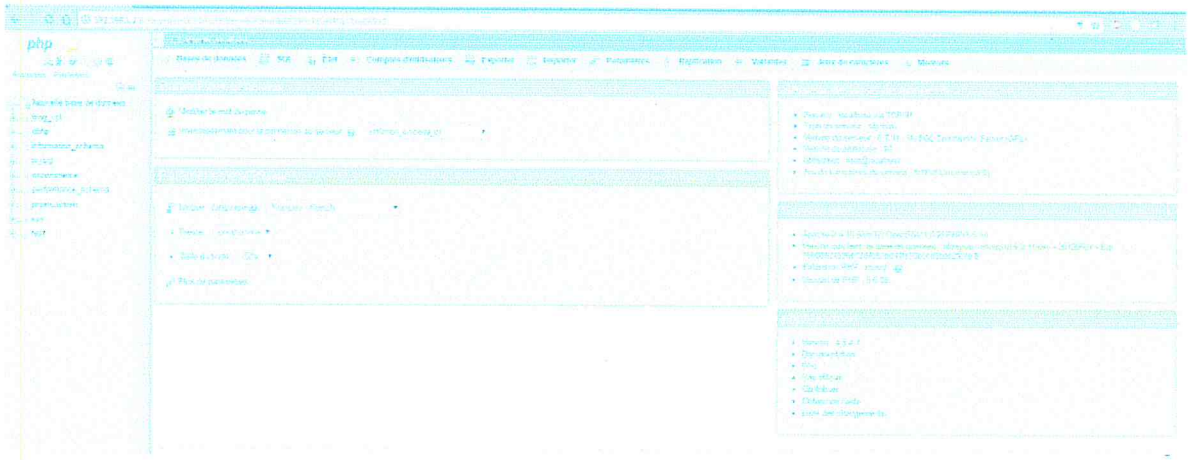


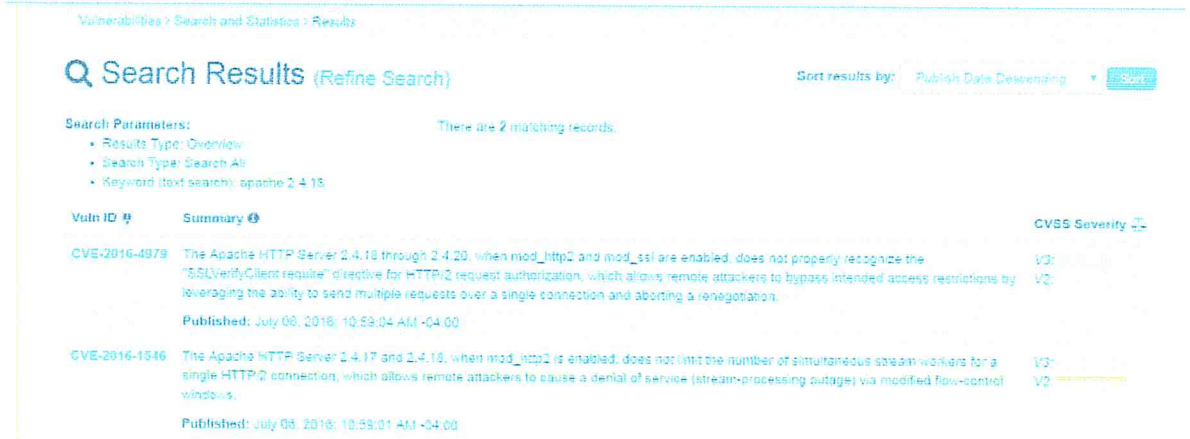
Figure 36 : interface phpMyAdmin

Après avoir accès à cette interface administrative nous pouvons effacer les bases de données de l'application cible, et les bases de données des autres applications qui utilisent la même interface, nous pouvons créer des utilisateurs avec tous les privilèges, nous pouvons voler les informations ou les modifier, donc ça affecte l'intégrité et la confidentialité des données et leur disponibilité à les utilisateurs de cette application.

8.2 Tester les bugs de logiciel du serveur web

Nous avons identifié dans la phase de reconnaissance que le logiciel utilisé est Apache 2.4.18 après avoir consulté les bases de données de vulnérabilités (NIST NVD), nous avons trouvé qu'il existe des vulnérabilités dans ce produit.

Voilà les vulnérabilités listées par NIST



Vulnerabilities > Search and Statistics > Results

Search Results (Refine Search)

Sort results by: Publish Date Descending Sort

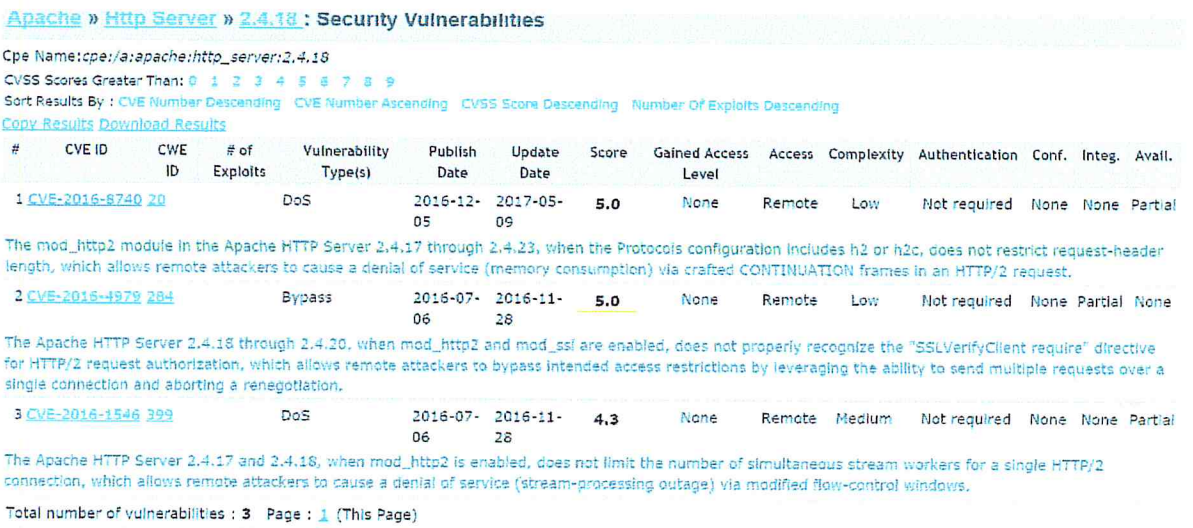
Search Parameters:
- Results Type: Overview
- Search Type: Search All
- Keyword (text search): apache 2.4.18

There are 2 matching records.

Vuln ID	Summary	CVSS Severity
CVE-2016-4979	The Apache HTTP Server 2.4.18 through 2.4.20, when mod_http2 and mod_ssl are enabled, does not properly recognize the "SSLVerifyClient require" directive for HTTP/2 request authorization, which allows remote attackers to bypass intended access restrictions by leveraging the ability to send multiple requests over a single connection and aborting a renegotiation. Published: July 06, 2016; 10:59:04 AM -04:00	V3: V2:
CVE-2016-1546	The Apache HTTP Server 2.4.17 and 2.4.18, when mod_http2 is enabled, does not limit the number of simultaneous stream workers for a single HTTP/2 connection, which allows remote attackers to cause a denial of service (stream-processing outage) via modified flow-control windows. Published: July 06, 2016; 10:59:01 AM -04:00	V3: V2:

Figure 37 : Recherche de vulnérabilités dans la base de données NIST

Aussi les vulnérabilités listées par CVE DETAILS



Apache » Http Server » 2.4.18 : Security Vulnerabilities

Cpe Name:*cpe:/a:apache:http_server:2.4.18*

CVSS Scores Greater Than: 0 1 2 3 4 5 6 7 8 9

Sort Results By : CVE Number Descending CVE Number Ascending CVSS Score Descending Number Of Exploits Descending

Copy Results Download Results

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2016-8740	20		DoS	2016-12-05	2017-05-09	5.0	None	Remote	Low	Not required	None	None	Partial
The mod_http2 module in the Apache HTTP Server 2.4.17 through 2.4.23, when the Protocols configuration includes h2 or h2c, does not restrict request-header length, which allows remote attackers to cause a denial of service (memory consumption) via crafted CONTINUATION frames in an HTTP/2 request.														
2	CVE-2016-4979	284		Bypass	2016-07-06	2016-11-28	5.0	None	Remote	Low	Not required	None	Partial	None
The Apache HTTP Server 2.4.18 through 2.4.20, when mod_http2 and mod_ssl are enabled, does not properly recognize the "SSLVerifyClient require" directive for HTTP/2 request authorization, which allows remote attackers to bypass intended access restrictions by leveraging the ability to send multiple requests over a single connection and aborting a renegotiation.														
3	CVE-2016-1546	399		DoS	2016-07-06	2016-11-28	4.3	None	Remote	Medium	Not required	None	None	Partial
The Apache HTTP Server 2.4.17 and 2.4.18, when mod_http2 is enabled, does not limit the number of simultaneous stream workers for a single HTTP/2 connection, which allows remote attackers to cause a denial of service (stream-processing outage) via modified flow-control windows.														

Total number of vulnerabilities : 3 Page : 1 (This Page)

Figure 38: Recherche de vulnérabilités dans la base de données *CVE DETAILS*

Donc la version du serveur web utilisé par l'application est vulnérable.

8.3 Tester le contenu par défaut

Apres avoir lancé Nikto nous avons déduis quelques vulnérabilités et selon la base de données de vulnérabilités OSVDB voila ci-dessous leur explication :

+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST

Probablement une vulnérabilité **Cross-Site Tracing (XST)** qui implique l'utilisation de Cross-site Scripting (XSS) et les méthodes HTTP TRACE.

+ OSVDB-3268: /icons/: Directory indexing found.

Ceci est considéré comme une vulnérabilité mineure de divulgation d'informations.

+ OSVDB-3233: /icons/README: Apache default file found.

Peut être un problème de mise à jour du serveur apache ou bien de configuration.

8.4 Tester les méthodes dangereuses du protocole http

Nous avons utilisé la méthode OPTIONS pour essayer de savoir quelle sont les méthodes WebDAV autorisées par le serveur dans les différents répertoires trouvés précédemment dans la phase de mappage de l'application, nous avons obtenu le même résultat pour tous les répertoires identifiés :

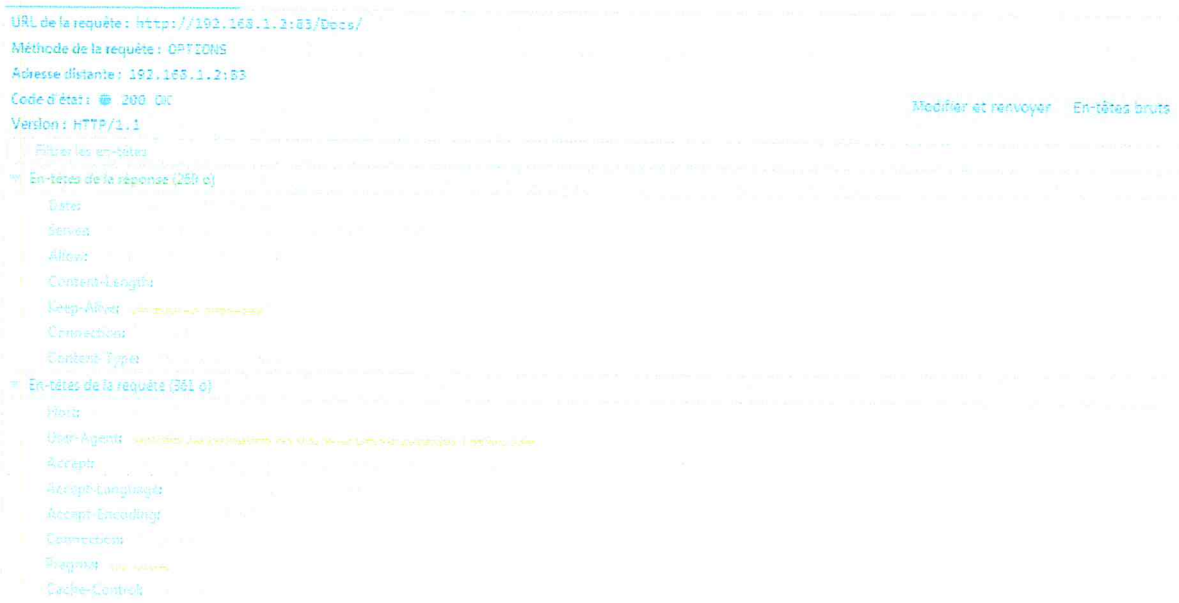


Figure 39 : réponse de requête http en utilisant la méthode OPTION

Donc nous remarquons que les méthodes autorisées par le serveur pour les répertoire sont GET, POST, HEAD, OPTIONS et TRACE, donc aucune méthode dangereuse est utilisées par le serveur dans les répertoires.

Lorsque nous essayons de supprimer une page avec une requête DELETE, le serveur nous répond avec un statut http 301 et nous redirige vers la page indexe.s

8.5 Tester les fonctionnalités de proxy

Nous avons essayé d'utiliser le serveur de l'application en tant que proxy, en envoyant des requêtes GET et CONNECT à partir de l'application à un autre site web, nous avons utilisé Netcat pour exécuter ces requêtes, voici le résultat pour la requête GET :

```
          : # nc -nvv 192.168.1.2 83
(UNKNOWN) [192.168.1.2] 83 (?) open
GET http://192.168.145.131 HTTP/1.1
HTTP/1.1 408 Request Timeout
Date: Sun, 28 May 2017 12:01:42 GMT
Server: Apache/2.4.18 (Win32) OpenSSL/1.0.2f PHP/5.6.18
Content-Length: 327
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>408 Request Timeout</title>
</head><body>
<h1>Request Timeout</h1>
<p>Server timeout waiting for the HTTP request from the client.</p>
<hr>
<address>Apache/2.4.18 (Win32) OpenSSL/1.0.2f PHP/5.6.18 Server at 192.168.145.131 Port 80</address>
</body></html>
 sent 36, rcvd 538
          : #
```

Figure 40: résultat pour la requête GET

Et le résultat pour la requête CONNECT :

```
          : # nc -nvv 192.168.1.2 83
(UNKNOWN) [192.168.1.2] 83 (?) open
CONNECT 192.168.145.131:80 HTTP/1.1
HTTP/1.1 408 Request Timeout
Date: Sun, 28 May 2017 12:12:10 GMT
Server: Apache/2.4.18 (Win32) OpenSSL/1.0.2f PHP/5.6.18
Content-Length: 327
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>408 Request Timeout</title>
</head><body>
<h1>Request Timeout</h1>
<p>Server timeout waiting for the HTTP request from the client.</p>
<hr>
<address>Apache/2.4.18 (Win32) OpenSSL/1.0.2f PHP/5.6.18 Server at 192.168.145.131 Port 80</address>
</body></html>
 sent 36, rcvd 538
          : #
```

Figure 41: résultat pour la requête CONNECT

Nous remarquons que le serveur répond avec un code http 408, ce qui veut dire que le serveur ne transmet pas les requêtes aux autres sites. Donc le serveur n'est pas configuré en tant que proxy.

2. Conclusion

Afin d'illustrer l'efficacité de notre approche, nous avons réalisé un ensemble de tests, nous avons fait toutes les étapes tracées, nous concluons qu'au début l'application était vulnérable qu'il n'existait pas un mécanisme d'authentification, de sessions et contrôle d'accès. Et que le site était vulnérable face aux attaques d'injection de scripts, qu'il n'utilise aucun mécanisme de défense contre les attaques XSS reflétées et enregistrées, et Que nous avons corrigé et proposer des parades pour rectifier cela.

Conclusion générale

De nos jours, les applications Web sont à la fois beaucoup plus répandues et beaucoup plus complexes que celles des années 2000. Le développement du Web dynamique et la richesse fonctionnelle qu'offrent les nouvelles technologies du Web permettent de répondre à un grand nombre de besoins. Néanmoins, cette richesse fonctionnelle s'accompagne d'une complexité grandissante, et cette progression va de pair avec une multiplication du nombre de vulnérabilités informatiques publiées, offrant une surface d'attaque conséquente. De ce fait, aucun serveur Web n'est sûr à 100% et les applications Web sont devenues de plus en plus vulnérables et exposées à des attaques malveillantes pouvant porter atteinte à des propriétés essentielles telles que la confidentialité, l'intégrité ou la disponibilité des informations. Pour faire face à ces menaces, il est primordial de développer des parades efficaces permettant de les contrer et à identifier les vulnérabilités dans ces applications.

Dans cette perspective la « Sonatrach » nous a demandé de travailler sur le développement d'un applicatif web sécurisé consacré à leurs bases de données donc très sensibles.

Les principales contributions de ce mémoire peuvent se résumer comme suit :

En premier lieu nous avons étudié l'anatomie du web ensuite nous avons examiné les failles les plus répandus sur Php, Apache et Mysql, Windows et Linux qui va permettre d'injecter ou exécuter un code, interrompre le service , élévation de privilège …etc. Puis le troisième chapitre était réservé pour le développement d'une approche permettant de tester l'application web et pour cela nous avons tracé 8 étapes bien définies pour tester les failles, les exploiter et d'implémenter des solutions pour cela, et enfin nous avons appliqué ces étapes à l'aide de différents outils et de scripts.

Les résultats que nous avons obtenus sont encourageants. Ils montrent en particulier que l'approche proposée peut effectivement contribuer à l'amélioration de la sécurisation des sites web.

Nous pouvons envisager différentes perspectives à ces travaux et transformer cette approche en méthodologie et développer un framework de pentest spécial pour les applications web.

Ce projet a été pour nous une chance et une formidable opportunité pour découvrir un environnement de travail informatique nouveau, complexe et vaste, ce qui nous a permis d'acquérir de l'expérience en sécurité informatique et d'approfondir nos connaissances dans le domaine du développement web sécurisé

Annexe :

La présente annexe montre une liste non exhaustive de vulnérabilités des technologies php, MySQL, Apache et Windows et des parades pour contrer ces vulnérabilités.

Numéro	Vulnérabilité / Menaces	impact	parade
1	$GLOBALS	1. Peut être utilisé pour accéder a d'autre variable par nom	1. Assurez-vous que toutes les variables sont correctement initialisées.
2	register_globals	1. L'application ne peut pas indiquer efficacement d'où proviennent les données	2. Examiner le code de script
			1. Assurez-vous que toutes les variables sont correctement initialisées.
			3. Désactiver register_globals
3	Les paramètres d'entrée dont les noms contiennent des indices entre crochets sont automatiquement converties en tableaux	1. Peut entraîner un comportement inattendu dans l'application si un tableau est passé à une fonction qui attend une valeur scalaire	4. Echapper les crochets.
4	accès aux fichiers fopen fpassthru unlink	1.Peut être utilisé pour accéder à des fichiers distants	5. Validation des données entrée par l'utilisateur
			6. Contrôle d'accès
			7. Désactiver allow_url_fopen si nous avons pas besoin
5	accès aux fichiers readfile, gzopen, gzfile, gzpassthru readgzfile	1.Peut être utilisé pour accéder à des fichiers distants	5. Validation des données entrée par l'utilisateur
			6. Contrôle d'accès
			8. Désactiver use_include_path si nous n'avons pas besoin

6	accès aux fichiers file, file_put_contents	1.Peut être utilisé pour accéder à des fichiers distants	5. Validation des données entrée par l'utilisateur
			6. Contrôle d'accès
			9. Désactiver FILE_USE_INCLUDE_PATH si nous n'avons pas besoin
7	accès aux fichiers copy, rename, file_get_contents	1.Peut être utilisé pour accéder à des fichiers distants	5. Validation des données entrée par l'utilisateur
			6. Contrôle d'accès
			10. Ne pas accepter les URL externe.
8	accès aux fichiers rmdir, mkdir	1.Peut être utilisé pour accéder à des fichiers distants	5. Validation des données entrée par l'utilisateur
			6. Contrôle d'accès
			11. Utiliser Safe-mode de php
9	accès aux fichiers parse_ini_file	1.Peut être utilisé pour accéder à des fichiers distants	5. Validation des données entrée par l'utilisateur
			6. Contrôle d'accès
10	Fonction include, include_once, require, require_once	1. Possibilité d'exécution des commandes arbitraires sur le serveur	12. Définir spécifiquement le chemin
			4. Echapper les caractères dangereux
			13. Ne pas autoriser l'utilisateur à déterminer le chemin de fichier si possible, et ne pas utiliser allow_url_include si possible
11	Protocole HTTP/HTTPS	1. Peut être utilisé pour récupérer un fichier distant	14. Désactiver file_uploads si possible
12	Protocole FTP, SSH	1. Peut être utilisé pour récupérer un fichier distant	6. Authentification et contrôle d'accès

13	Si les données contrôlées par l'utilisateur sont passées à l'application par les fonctions suivantes (eval, call_user_func, call_user_func_array, call_user_method, call_user_method_array, create_function).	1. L'application est probablement vulnérable à l'injection des scripts	15. Ne pas accepter les chaines de caractères si possibles et éviter de laisser le contrôle des données pour l'utilisateur
14	La capacité d'invoquer des fonctions dynamiquement via une variable qui contient le nom de la fonction.	1. Un utilisateur peut provoquer l'application pour appeler une fonction arbitraire sans paramètres	16. Filtrage, Validation et Echappement des entrées d'utilisateurs
15	Les fonctions utilisées pour exécuter des commande de système (exec, passthru, popen, proc_open, shell_exec, system)	1. Exécution de commande arbitraire	16. Filtrage, Validation et Echappement des entrées d'utilisateurs 11. Utiliser le safe-mode
16	Redirection d'URL http_redirect	1. vulnérabilité au vecteur d'attaques de phishing	15. Ne pas laisser au utilisateur le contrôle sur la valeur de chaine de caractères entrée en paramètres de la fonction http_redirect
17	Les API utilisées pour créer et utiliser le réseau (socket_create, socket_connect, socket_write, socket_send, socket_recv, fsockopen, pfsockopen)	1. L'application peut être exploitable pour provoquer des connexions réseau à des hôtes arbitraires	15. Ne pas laisser à l'utilisateur le contrôle des informations de l'hôte

18	Identificateur par défaut de serveur apache	1. affecte la disponibilité, l'intégrité et la confidentialité de l'application	Changer les identificateurs par défauts et utiliser des mot de passse complexe
19	Fonctionnalité de debug	1. divulgation des informations sur la configuration de serveur	
		2. divulgation des informations sur l'état de serveur	
		3. divulgation des informations sur l'application qui est en cours d'exécution	
20	Exemples de fonctionnalités	1. risque d'existence de vulnérabilités dans ces exemples de fonctionnalités	
21	Liste de répertoire	1. Risque de divulgation des fichiers sensibles	
22	Méthodes WebDav (PUT, DELETE, COPY, MOVE, SEARCH, PROPFIND)	1. Risque de téléchargement des scripts malicieux	
23	Serveur en tant que proxy	1. Un pirate peut utiliser le serveur pour attaquer des	

		systèmes tiers sur internet	
		2. Un pirate peut utiliser le serveur pour connecter à des hôtes arbitraires sur le réseau interne de l'organisation	
		3. Un pirate peut utiliser le serveur pour connecter à d'autres services qui sont exécutés sur le serveur ainsi contourné les restrictions de pare-feu	
25	Déni de Service	1. affecte la disponibilité de l'application	
26	Exécution de code arbitraire	1. affecte la disponibilité, l'intégrité et la confidentialité de l'application	
27	Exposition de serveur utilisé	1. La configuration par défaut exposera la version d'apache et le type de système	Modifier le fichier httpd.conf et ajouter les lignes suivantes: ServerTokens Prod, ServerSignature Off.

		d'exploitation	
28	ETAG	1. Il permet aux attaquants distants d'obtenir des informations sensibles comme le nœud d'index, Et processus fils via l'en-tête Etag	Ajouter la ligne FileETag None au httpd.conf
29	l'utilisation d'un compte apache privilégié pour l'exécuter	1. Risque d'exposer les autres services aux attaquants si le serveur apaches est compromis	Exécuter apache en utilisant un compte non-priviligié
30	Permission par défaut pour les répertoires binary et configuration.	1. Divulgation de configuration de serveur	Changer la permission de répertoires bin et conf (sous linux: #chmod –R 750 bin conf)
31	Modifier la configuration d'apache en utilisant .htaccess	1. Perte de configuration de serveur Apache	modifier le fichier httpd.conf comme suit : <Directory /> Options -Indexes AllowOverride None </Directory>
32	la méthode HTTP Trace	1. Peut permettre l'attaque Cross Site Tracing et potentiellement donner une option à un pirate pour voler des informations cookie.	Ajouter la directive suivante au fichier httpd.conf: TraceEnable off

33	XSS		Assurer que le mod_headers.so est activé dans le fichier httpd.conf
			Ajouter la directive suivante au fichier httpd.conf: Header edit Set-Cookie ^(.*)$ $1;HttpOnly;Secure et Header set X-XSS-Protection "1; mode=block"
34	Attaque Clickjacking		Assurer que le mod_headers.so est activé dans le fichier httpd.conf
			Ajouter la directive suivante au fichier httpd.conf: Header always append X-Frame-Options SAMEORIGIN
35	Server Side Include (SSI)	1. L'injection des Scripts dans des pages HTML et les exécuter à distance	Dans le fichier httpd.conf, chercher le repertoire et ajouter 'includes' dans la directive Options comme suit <Directory /opt/apache/htdocs> Options –Indexes **-Includes** Order allow,deny Allow from all </Directory>
		2.Risque d'augmenter la charge sur le serveur	
36	HTTP 1.0	1. HTTP 1.0 a une faiblesse de sécurité liée au détournement de session.	Assurer de charger le module mod_rewrite dans le fichier httpd.conf et Activer la directive RewriteEngine comme suit : RewriteEngine On RewriteCond %{THE_REQUEST}

			!HTTP/1.1$ RewriteRule .* - [F]
37	Configuration de valeur de timeout	1. peut exposer l'application à des attaques de déni de service (Attaque Slow Loris)	Modifier le fichier httpd.conf en ajoutant la directive suivante : Timeout 60
38	SSL v2/v3	1. Expose l'application à des attaque de l'homme au milieu	Désactiver SSL V2/V3 en modifiant le fichier httpd-ssl.conf comme suit : SSLProtocol –ALL +TLSv1 +TLSv1.1 +TLSv1.2
39	Cache d'identifcateurs de Windows	1. Cette fonctionnalité peut être abusée par un adversaire qui peut récupérer ces informations d'identification mises en cache sur la base de données SAM	Pour réduire ce risque, les informations d'identification mises en cache ne doivent pas être stockées pour les postes de travail
40	Authentification Wdigest	1. les identificateurs d'un utilisateur dans une session actif peuvent être volés par un adversaire	Désactiver Wdigest
41	Utilisation dangeureuse de mémoire	1. Exécution de code depuis des	Activer prévention de l'exécution des données (DEP)

		blocs de mémoire censés contenir des données	
42	Élévation de privilèges	1. L'octroi à un intrus d'autorisations supérieures à celles initialement accordées	Appliquer le principe de séparation de privilèges
			Limiter l'utilisation des privilèges d'administrateur
			Utiliser UAC
43	Système d'authentification faible	1. Accès facile à des ressources critiques	Utiliser authentification multi-facteur
44	Politique de mot de passe	1. Exposer à des attaques de force brute	Exiger des mots de passe complexe, et le changement de ces mots de passe régulièrement
45	Abus de privilèges	1. Possession des ressouces non autorisées	Avoir un logicielle de chiffrement
		2. Ajustement du jeton d'accès au processus	Appliquer le principe de séparation de privilèges
		3. et la manipulation de la mémoire	Limiter l'utilisation des privilèges d'administrateur
		4. Sauvegarde et restauration	Utiliser UAC
		5. Imitation d'autres utilisateurs	
46	Contournement des stratégies de groupes	1. Accès non autorisé	Avoir un logicielle de chiffrement
			Appliquer le principe de séparation de privilèges
			Limiter l'utilisation des privilèges

			d'administrateur
47	Contournement de logicielle de chiffrement	1. Affecte la confidentialité, l'intégrité et la disponibilité des données	Appliquer le principe de séparation de privilèges
			Limiter l'utilisation des privilèges d'administrateur
			Utiliser UAC
48	Faille XSS	Vol d'identifiant de session, redirection du client, altération visuelle du site, etc.	strip_tags() Permet de supprimer tous les tags HTML non souhaités (tous tags sauf ceux passés en paramètre de la fonction) htmlspecialchars() Permet de convertir les caractères &, ‘,",<,> sous forme d'entités HTML htmlentities() Permet de convertir tous les caractères sous forme d'entités HTML y compris les caractères ‘à’, ‘é’, ‘è’, etc… WAF – WEB APPLICATION FIREWALL HEADER HTTP
49	CSRF	exécuter des actions à l'insu des utilisateurs	Un captcha authentification par les jetons
50	CRLF	Mdp	supprimer les retours à la ligne
51	Injection	modifier une requête SQL	Utiliser la fonction mysql_real_escape_string()

[19] Sécurité PHP 5 et MySQL par Damien Seguy Philippe Gamache 2007

[20] Protégez-vous efficacement contre les failles web par Charles Senges sur openclassrooms.com 2017