الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université de 8 Mai 1945 – Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière
Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Sciences et Technologie de l'Information et de la Communication (STIC)

Thème :

Détection des vulnérabilités des applications Web aux attaques XSS

Encadré par :

Dr. HANNOUSSE ABDELHAKIM

Présenté par :

GUELMI ZAID

Juin 2022

Remerciement

Avant tout, il apparait opportun de rendre grâce à **DIEU** de nous avoir donné le courage et la volonté et la patience et le savoir-faire afin d'accomplir ce travail.

Je tiens tout d'abord à remercier mon encadreur **Dr. HANNOUSSE ABDELHAKIM**, enseignante à l'Université de GUELMA, pour tout le soutien, l'aide, l'orientation, la guidance, ainsi que pour ses précieux conseils et ses encouragements lors de la réalisation de mon mémoire. J'ai beaucoup appris à son contact et ce fut un grand plaisir de travailler avec elle.

Je voudrai lui adresser mes vifs remerciements et de lui témoigner mon sincère reconnaissance.

Je remercierai également toutes les personnes qui, de près ou de loin, m'aident à l'élaboration de ce mémoire Surtout **Ayman**, et le meilleur

Je remercierai également toutes les personnes qui, de près ou de loin, m'aident à l'élaboration de ce mémoire

Dédicace

Je dédie ce modeste travail aux parents décédés, en plus de la généreuse épouse qui m'a soutenu tout au long de la saison scolaire, en plus de mes chères filles **Hanine** et **Afnane**,
À mes très chers frères **Noreddine** et **Ahcen youcef, Aymane** pour leurs encouragements permanents, et leur soutien moral.
A toute ma famille pour leur soutien tout au long de mon parcours universitaire.
Et je n'oublierai pas tous ceux qui nous ont soutenu De près et de loin, surtout mes amis et **Adlane, Malik,** et tous les élèves de promotion, tout comme nous

Résumé

Le cross-site scripting (XSS) est l'une des attaques les plus dangereuses menaçant la navigation sur le Web depuis sa révélation fin 1999. Depuis lors, plusieurs techniques ont été développées dans le but de sécuriser les applications Web contre divers types d'attaques XSS. Dans ce projet, nous contribuons avec la conception d'une approche hybride pour la détection des vulnérabilités des applications web aux attaques XSS. De cette façon, les applications vulnérables peuvent être détectées et donc modifiées pour se défendre contre les attaques XSS. L'approche hybride combine deux types d'analyses : *statique* et *dynamique*. Alors que l'analyse statique est utilisée pour détecter tous les points d'injection inclus dans des pages individuelles en analysant leur contenu, l'analyse dynamique est utilisée pour confirmer la vulnérabilité de ces points à l'injection du vecteur d'attaque XSS. Un prototype nommé *XSS Checker* est développé en Node.js, implémentant l'approche proposée. Les expériences menées, avec le prototype développé, ont montré la capacité de l'approche proposée à détecter les vulnérabilités dans des applications du monde réel.

Mots-clés : Sécurité Web, Cross-site scripting; Détection des vulnérabilités XSS.

Abstract

Cross-site scripting (XSS) is one of the most dangerous attacks menacing the navigation in the Web since its reveal in late 1999. Since then, several techniques have been developed in the aim to secure web applications against diverse types of XSS attacks. In this project, we contribute by designing a hybrid approach for the detection of web application vulnerabilities to XSS attacks. This way, vulnerable applications can be detected and hence updated to defend against XSS attacks. The hybrid approach combines static and dynamic analysis. While static analysis is used to detect of all the injection points included in individual pages through analyzing their contents, dynamic analysis is used to confirm the vulnerability of such points to XSS payload injection. A prototype named *XSS Checker* is developed in Node.js implementing the proposed approach. Conducted experiments, with the developed prototype, showed the ability of the proposed approach to detect vulnerabilities in real world applications.

Keywords : Web security, Cross-site scripting; XSS vulnerabilities detection.

ملخص

تعد البرمجة النصية عبر المواقع (XSS) واحدة من أخطر الهجمات التي تهدد التنقل في الويب منذ الكشف عنها في أواخر عام 1999. ومنذ ذلك الحين ، تم تطوير العديد من التقنيات بهدف تأمين تطبيقات الويب ضد أنواع مختلفة من هجمات XSS. في هذا المشروع ، نساهم من خلال تصميم نهج مختلط لاكتشاف نقاط الضعف في تطبيقات الويب لهجمات XSS. بهذه الطريقة ، يمكن اكتشاف التطبيقات الضعيفة وبالتالي تحديثها للدفاع ضد هجمات XSS. يجمع النهج الهجين بين التحليل الساكن والديناميكي. بينما يتم استخدام التحليل الثابت لاكتشاف جميع نقاط الحقن المضمنة في الصفحات الفردية من خلال تحليل محتوياتها ، يتم استخدام التحليل الديناميكي لتأكيد ضعف هذه النقاط في حقن حمولة XSS. تم تطوير نموذج أولي يسمى XSS Checker ب Node.js لتنفيذ النهج المقترح. أظهرت التجارب التي تم إجراؤها مع النموذج الأولي المطور قدرة النهج المقترح على اكتشاف نقاط الضعف في تطبيقات العالم الحقيقي.

الكلمات الدالة :

أمن الويب، البرمجة النصية عبر المواقع؛ الكشف عن ثغرات XSS.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Le cross-site scripting (XSS) a été découvert en 1999, et est classé de nos jours, comme étant l'attaque la plus virulente vis-à-vis des applications Web. Les attaquants par XSS injectent des scripts malveillants dans des endroits spécifiques au sein des applications, des serveurs ou des plug-ins ; ces emplacement cibles sont appelés Vulnérabilités ; on peut dire que ces vulnérabilités-nommées parfois aussi XSS-représentent les maillons faibles des applications Web. A travers ces vulnérabilités, et en explorant les pages Web des utilisateurs cibles, les attaquants peuvent accéder de manière transparente aux informations personnelles et sensibles, sauvegardées par les navigateurs des victimes, telles que leurs noms d'utilisateurs et mots de passe.

La cause de premier abord des attaques XSS concerne les zones de saisie : ces zones souffrent d'un manque remarquable de contrôles ; ce manque a entre autres pour cause, l'inexpérience des développeur Web dans le domaine de sécurité ; la majorité des développeurs sont emportés par le design et l'ergonomie de leurs applications, et sous-estiment les répercussions d'un contrôle limité ou parfois inexistant. Le changement rapide dans les exigences de leur client, peut aussi affecter le contrôle de sécurité des applications ; enfin l'attaque XSS est si banale, que beaucoup de développeurs n'imaginent pas qu'une simple saisie d'un nom ou d'un prénom peut entrainer une injection d'un script qui peut se révéler dévastateur [2].

Le rapport de l'Open Web Application Security Project (OWASP) publié en 2021 [W1], indique que les attaques XSS occupent les premiers rangs dans la liste des dix principales vulnérabilités mondialement connues. Le fait que l'on peut lancer une attaque XSS depuis un client ou un serveur, rend la tâche de détection de ces attaques encore plus difficile.

Pour lutter contre ce type de cyberattaques, plusieurs efforts ont été réalisés pour le développement des outils qui permettront de détecter les vulnérabilités des applications Web aux attaques XSS ; la majorité de ces tentatives focalisent sur l'utilisation unique des approches statique ou dynamique. Notre projet de fin d'étude s'inscrit dans ce cadre, il s'intéresse à examiner les applications Web et déterminer si le code de ces applications est vulnérable aux attaques XSS. Pour cela, nous visons le développement d'une approche hybride qui permet d'examiner le code des pages Web des applications en cours d'exécution et déterminer si des vecteurs d'attaques XSS spécifiques peuvent passer les codes de contrôles associés à chaque champ de saisie. Afin de réaliser ce but, le présent mémoire est structuré en trois chapitres :

Chapitre 1 : dans le premier chapitre, nous définissons l'attaque et la vulnérabilité XSS et nous expliquons ses différents types et le principe de base de chaque type.

Chapitre 2 : dans ce chapitre, nous présentons les différentes solutions récentes proposées pour la détection des vulnérabilités XSS. Cette présentation nous permettra d'avoir un aperçu sur les derniers développements réalisés pour la détection des vulnérabilités XSS par l'utilisation des approches statiques, dynamiques et hybrides.

Chapitre 3 : dans ce chapitre, nous détaillons notre solution hybride proposée pour la détection des vulnérabilités XSS. L'approche proposée consiste à examiner les pages des applications Web. Une phase d'analyse statique légère permet l'identification des différents champs de saisie existants dans la page ; une phase d'analyse dynamique permet d'injecter des vecteurs d'attaques XSS dans chacun de ces champs et observer le comportement du serveur pour décider du succès ou de l'échec des attaques.

CHAPITRE I.

INTRODUCTION AUX ATTAQUES ET VULNERABILITES XSS

CHAPITRE I.

INTRODUCTION AUX ATTAQUES ET VULNERABILITES XSS

L'utilisation massive de l'internet, avec la technologie mobile et la prolifération des réseaux sociaux, incite les créateurs des applications Web à améliorer de façon constante et accélérée leur contenu graphique et fonctionnel, au détriment des contrôles de sécurité. De ce fait, les cyberattaques profitent énormément des failles de sécurité dans les applications Web et multiplient leurs ruses afin d'exploiter le plus grand nombre de victimes. Il est donc primordial de comprendre comment fonctionnent ces attaques afin d'établir les meilleures solutions et mesures d'atténuation de leurs impacts nocifs.

Nous présentons dans le présent chapitre une cyber-attaque des plus anciennes et des plus virulentes : le cross-site scripting ou XSS. Nous commençons par sa définition et ses conséquences sur le Web, nous détaillons ensuite ses différents types à travers des exemples pratiques, enfin nous expliquons ses principales causes et les mesures à prendre pour la contrer.

1. Introduction aux risques de navigation Web

La popularisation d'Internet a connu son élan vers la fin des années 80s, avec l'apparition du Web. Le nombre des internautes n'a cessé d'augmenter depuis, et a connu une augmentation phénoménale avec l'apparition des réseaux sociaux vers la fin des années 90s. Avec l'arrivée des dispositifs mobiles sur le marché, le nombre a connu son premier boom avec environ 2 milliards d'utilisateurs en 2012. La

technologie Web 2.0 a facilité énormément cette métamorphose en popularisant la création du contenu Web ; un utilisateur novice peut créer en quelques clics son blog ou son forum [W1].

Plus de la moitié de la population mondiale est rivé sur Internet ; on compte environ 5 milliards d'utilisateurs de ce réseau mondial au début de l'année 2022. Avec un tel engouement, le commerce et la gouvernance électroniques sont en changement constant afin d'adapter leurs services aux besoins du plus grand nombre d'utilisateurs, en qualité et en quantité. Toutefois, les internautes deviennent de plus en plus exigeants et conscients des risques encourus par l'utilisation du Web, en ce qui concerne la divulgation de leurs données personnelles, avec l'augmentation des cyber-attaques [2].

Les cyber-attaques sont dues principalement à l'ignorance des utilisateurs du Web, mais pas seulement ; s'ajoute à ce problème les faiblaisses ou failles des applications Web, qu'on appelle *vulnérabilités*. Ces vulnérabilités peuvent être associées à l'application elle-même, son plug-in ou bien au serveur qui l'héberge. Ces failles sont dues généralement à une mauvaise gestion de la partie sécurité de l'application. Beaucoup d'attaques, comme l'attaque XSS, exploitent judicieusement ces failles et injectent des scripts malicieux pour dérober les données sensibles de leurs victimes, telles que les noms utilisateurs, mots de passes et numéros de cartes bancaires [2].

2. Le cross-site scripting (XSS)

Les applications Web contenant des zones de saisies sont les cibles privilégiées des attaques XSS. Le vecteur d'attaque XSS est injecté dans ces zones pour se propager par la suite sur toutes les pages Web. Ce qui indique qu'une telle attaque n'est rendu possible que parce que les développeurs d'application ne font pas attention à bien mettre les contrôles de sécurités et les filtres appropriés sur les formulaires et zones de saisie. Les code XSS malicieux se présentent sous forme de scripts écrit en JavaScript, ActiveX ou VBScript. Ils seront alors interprétés directement par le navigateur chaque fois que les pages cibles sont visitées. Généralement un code XSS est injecté de deux façons distinctes [5] :

1. Dans les paramètres des URLs, ensuite la victime est amenée à cliquer sur les liens menant vers ces URLs, soit en envoyant ce lien vers le mail de la victime, soit en l'insérant dans un site tiers très visité.
2. Par injection directe dans la base de données des serveurs, à travers des messages ou des fichiers infectés, par la suite, tous les visiteurs voulant lire ces messages verrons leurs pages infectées.

Les navigateurs Web ne sont pas armés contre les attaques XSS ; les données sensibles qui y sont caché peuvent être découvertes ; la structure des pages HTML qu'ils interprètent peut être modifiée, et un script malicieux peut contourner les contrôles d'accès et se substituer aux victimes.

Dès sa découverte en 1999 par des ingénieurs de Microsoft, l'attaque XSS ne cesse de se développer et de causer des pertes signifiantes, profitant de la nouvelle technologie Web et des exigences de rapidité ainsi que de la qualité de services des applications [5].

3. Effets des attaques XSS

Les conséquences d'une attaque XSS vont de l'affichage banale d'une fenêtre pop-up au vol des informations sensibles de connexion et de finance. Nous présentons dans ce qui suit trois de ces effets les plus marquants.

3.1. Vol des cookies

Les utilisateurs qui ont tendance à faire souvent leurs achats en ligne, aiment rester connectés à leurs sites préférés, et ne pas avoir besoin d'introduire leurs noms utilisateurs, mots de passes, numéros de carte de crédit, code CCV, etc ., à chaque connexion. Ces données sensibles doivent donc être enregistrés dans le navigateur client et stockées sous forme textuelle. On appelle ces fichiers stockés les *cookies*. En plus de sauvegarder les informations sensibles des utilisateurs [6], ils peuvent aussi enregistrer leurs préférences et expériences de navigation pour tel ou tel site, pour s'en souvenir lors de leurs prochaines visites. Les informations sensibles dissimulées dans les cookies sont forcément la cible préférée d'une attaque XSS qui peut à travers

un script malicieux se faire passer pour l'utilisateur légitime et transférer ces informations au serveur de l'attaquant [5].

3.2. Altération de l'apparence des pages Web

En exploitant les vulnérabilités des sites Web, un attaquant XSS peut injecter facilement des contenu varié et nocifs dans le site de la victime, cela peut aller de l'image obscène, aux discours xénophobes et extrémistes affectant ainsi leur notoriété auprès de leurs fidèles utilisateurs [5].

3.3. Déni de service

Le déni de service est lui-même une attaque très nuisible qui peut provoquer la paralysie des serveurs Web. Traditionnellement il est lancé par plusieurs attaquants en même temps. L'attaque XSS vient faciliter cette attaque sournoise en générant via un script malicieux plusieurs requêtes vers un même serveur [2].

4. Prévalence des vulnérabilités aux attaques XSS

Les vulnérabilités des applications Web sont les principales causes des attaques XSS. Dans le rapport 2014-2015 du *State of Open Source Security Report 2020 – Snyk,* la vulnérabilité XSS occupe la première place en total et en détail sur les cinq années, largement devant l'exécution de code et le déni de service comme le montre la figure 1.1.

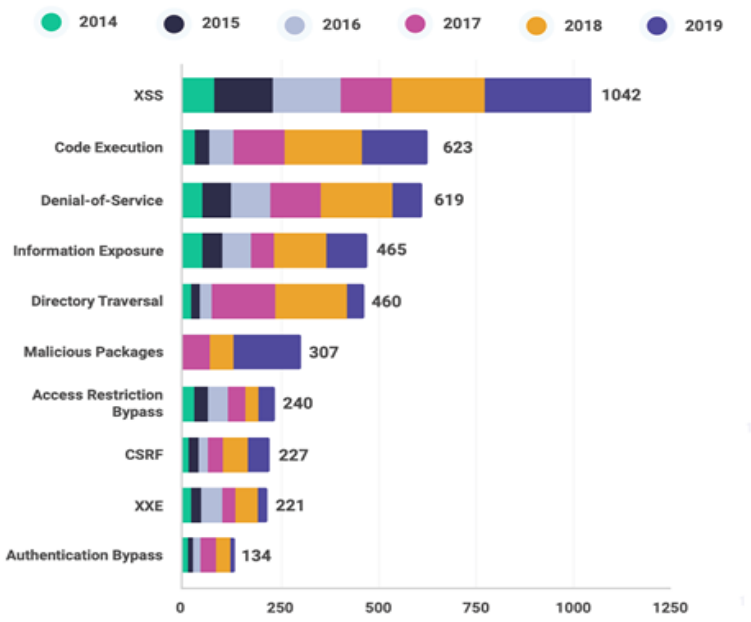

Figure 1.1. Prévalence des vulnérabilités XSS [W6]

5. Catégories des attaques XSS

Il existe trois catégories d'attaques XSS que L'Open Web Application Security Project (OWASP) [W1] a signalé, dans ce qui suit, nous en discutons les détails.

5.1. Attaques XSS reflétées (Reflected XSS)

Appelés « reflected XSS » en anglais, ces attaques se produisent lorsqu'un utilisateur clique sur un lien hypertexte qui lui est envoyé par mail, ou qui est inclus de manière attirante dans un site tiers. Ce lien hypertexte est en réalité construit par l'attaquant à partir de la réponse d'un serveur, via une requête http ou URL, en y intégrant des scripts malicieux. Donc lorsque le lien est exécuté sur le navigateur client, le script malveillant peut s'infiltrer [W1], comme le montre la figure 1.2.

Les attaques reflétées sont aussi appelées attaques de type I [W1] ou attaques non-persistantes, car les scripts malicieux ne vont pas être stockés sur le serveur de manière permanente. Les applications Web victimes de ce type d'attaques sont celles qui contiennent des champs de recherche vulnérables [2,5], c'est-à-dire sans contrôle de saisie, ce qui permet à l'attaquant d'exécuter le script sur le navigateur de la victime sans être intercepté. En découvrant cette vulnérabilité, le pirate construit un lien malicieux et l'envoie au mails de ces victimes, ou bien insère un lien hypertexte dans un site Web tiers, incitant la victime à cliquer dessus, ce qui provoquera son exécution dans le navigateur de la victime et la redirection des informations personnelles de celle-ci vers le serveur de l'attaquant. Il est à noter que la popularisation du Web 2.0 a rendu très facile la publication de liens hypertextes malicieux ou non [4].

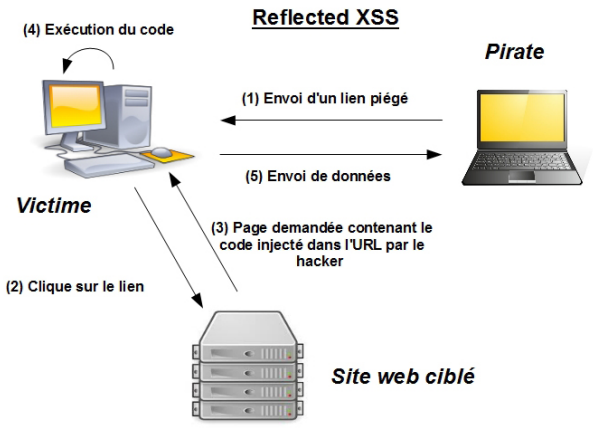

Figure 1.2. Attaque XSS reflétée [W6]

Exemple : Soit une application Web légitime, on prendra par exemple celle de l'université de Guelma (https://univ-guelma.dz/index.php), un attaquant pourra construire un vecteur d'attaque en utilisant l'URL de cette application comme suit :

```
http://univ-guelma.dz/index.php?user=<script>alert(document.cookie)</script>
```

En effet, ce vecteur est construit sur la zone de saisie 'user', contenu dans l'application Web ; si cette injection est réussite, cela veut dire que le champ de saisie 'user' est vulnérable et n'a pas été protégé contre une attaque de ce type. L'étape suivante sera que l'attaquant va envoyer ce code via un lien imbriqué dans une page tierce ou vers le mail de sa victime. Lorsque cette dernière clique sur le lien, le code

va s'exécuter sur son navigateur et, pour notre exemple, une boite de dialogue va afficher les cookies de la victime. Pour une attaque XSS reflétée réelle, ces informations vont être transférées automatiquement et de manière transparente vers le serveur de l'attaquant [W1].

5.2. Attaques XSS stockées (Stored XSS)

Dans ce type d'attaque, le pirate va stocker son code malicieux directement dans le serveur de l'application Web (i.e., dans sa base de données), en l'introduisant dans une zone de saisie (d'un formulaire, de commentaires, etc.,), et donc cela confirmé que ces zones de saisie ne sont pas bien protégées (pas de filtres ni de contrôles). Lorsqu'un utilisateur demande une page de cette application infectée, le script XSS sera exécuté, et le navigateur de la victime infecté [2,5]. Les attaques XSS stockées sont plus virulentes que celles reflétées, car le vecteur d'attaque est stocké en permanence sur le serveur, et peut donc infecter tous les utilisateurs qui demanderont des données depuis l'application Web infectée [4]. De ce fait, ces attaques sont aussi appelées *persistantes* ou de type II [W1], le contenu persistant nuisible continuera de l'être sauf si détecté et filtré par les outils appropriés. Les applications cibles préférées des attaques XSS stockées sont celles qui sauvegardent les commentaires et information introduits par les utilisateurs dans leurs bases de données, telles que les forums de discussion et les réseaux sociaux. La figure 1.3 explique les étapes d'une attaque XSS stockée.

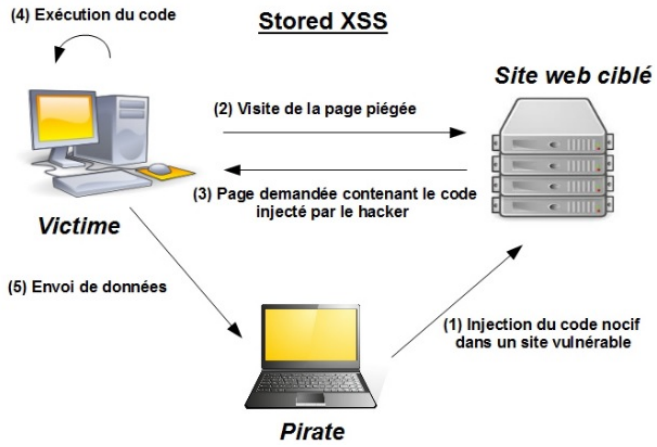

Figure 1.3. Attaque XSS stockée [W6]

Exemple : On va prendre comme exemple les commentaires introduits dans les postes d'un réseau social ; l'attaquant va introduire un commentaire sur un post utilisateur sur cette forme : (e.g., `<script>alert(document.cookie)</script>`). Dans le cas où la zone du commentaire n'est pas filtrée, le vecteur d'attaque ainsi introduit va-t-être stocké dans la base de données du réseau social, et exécuté à chaque fois que le commentaire est visualisé. Dans l'exemple cité ci-dessus, l'exécution va afficher les cookies de la victime sur une boite de dialogue, dans une attaque réelle XSS stockée, les cookies vont être transférées au serveur de l'attaquant à l'insu de la victime [W3].

5.3. Attaques XSS basées sur le DOM (DOM-based XSS)

Ces attaques ont la particularité d'être exécutées au moment de l'interprétation des pages HTML par le navigateur. Les codes malveillants vont modifier l'arbre DOM : Document Object Model, généré lors de l'interprétation des pages résultats des requêtes [7]. La page Web telle qu'elle existe sur le serveur ne va pas changer, mais le code coté client va s'exécuter différemment à cause des modifications malicieuses effectuées sur le DOM [2,5] ; les données malveillantes pouvant être intégrées dans des URLS en tant que valeurs d'objets ou d'éléments HTML. Les attaques XSS basées DOM sont aussi appelées de *Type 0*. Comme illustré dans la figure 1.4, l'attaquant commence par fabriquer une URL contenant le code malicieux et l'envoie à la victime ; cette dernière va demander l'URL malicieux depuis le serveur de l'application Web, Ce dernier reçoit la requête mais n'inclut pas le code malicieux dans la réponse. C'est en exécutant le script de la page légitime que le navigateur de la victime va insérer automatiquement le code malicieux dans le DOM HTML ; le navigateur exécute alors le script malicieux via le code coté client, après cette exécution des informations sensibles concernant la victime vont -t- être transférées vers le pirate.

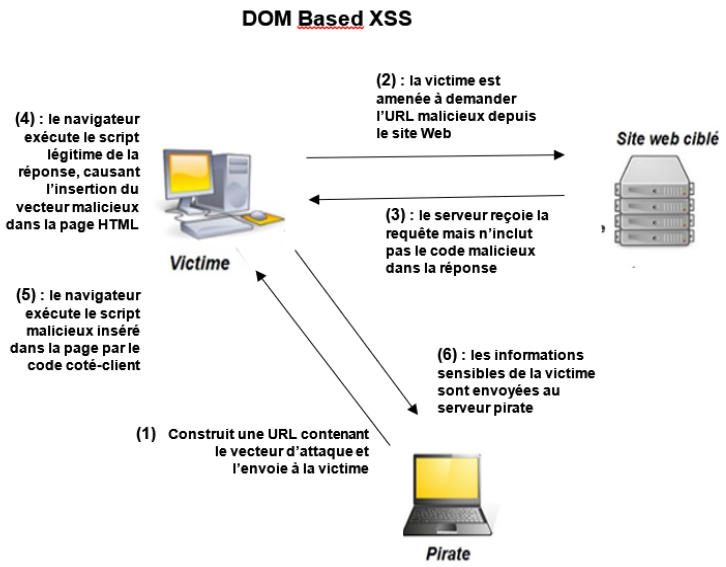

Figure 1.4. Attaque XSS basée sur le DOM [W6]

Exemple : Soit une page Web en HTML contenant le code suivant :

```
<script>
      document.write('Lien disponible sur :' + document.location.href + '.');
</script>
```

L'attaquant peut ajouter le code malveillant sous la forme suivante :

```
http://sitelegitime.com/index.php#<script>alert(document.cookie)</script>
```

En interprétant cette page, le code malveillant va remplacer l'attribut (`document.location.href`) dans la structure DOM de la page ; une boîte de dialogue s'affichera alors contenant les cookies de la victime. Si l'attaque DOM XSS était réelle, ces cookies vont être transférés vers le serveur de l'attaquant de manière transparente [W2, W3].

6. Causes de prévalence des attaques XSS

Le plus souvent, les attaques XSS ont pour cause des faiblesses au niveau des applications Web. Attiré par le souci de fonctionnalité, les développeurs oublient la plupart du temps d'ajouter les contrôles adéquats aux données introduites par les utilisateurs de leur application. Plusieurs causes sont à l'origine de cette omission :

6.1. Manque d'expérience des développeurs Web

Un très grand nombre de créateurs de contenu Web, même professionnels, a recourt aux plateformes de génération automatique des applications Web. Ces dernières attirent par leur utilisation intuitive et facile, et leur multitude fonctionnalités, concernant spécialement leur aspect graphique. Emportés par la vitesse de création et l'ergonomie fascinante, les développeurs oublient dans la mêlée les aspects sécuritaires de ces applications, souvent omis par les plateformes qui les ont générées, et qui demandent l'implication d'une équipe spécialisée dans la sécurité [2].

6.2. Exigences socio-économiques

Dans un monde aussi connecté et numérisé que le nôtre, et où le commerce électronique s'est introduit aux foyers les plus lointains, l'adaptations aux exigences des clients en matière de temps de réponse, de disponibilité et de qualité de service, dépasse parfois tout autre aspect. Cela a comme conséquence la négligence des contrôles de sécurité qui exigent une décélération dans la vitesse de création de l'application pour multiplier les tests de sécurité [2,5].

6.3. La simplicité de l'attaque XSS

Un attaquant XSS peut réaliser son infiltration en injectant un script malicieux dans une zone de saisie banale. Aussi simple d'apparence, car introduite à travers un canal légitime, cette attaque va générer de multiples dégâts. En plus, les attaquant peuvent utiliser une multitude de langages de programmation pour écrire leurs scripts, peuvent les encoder, les chiffrer, etc., rendant leur détection encore plus difficile. C'est pour cela qu'il est impératif de contrôler les zones d'entrée des données et les considérer comme étant les points d'attaques les plus sensibles à surveiller [2].

7. Quelques mesures de sécurités contre les attaques XSS

Les mesures de sécurité devant être prises pour contrer les attaques XSS, doivent impérativement impliquer deux parties : d'une part, le client, qui doit prendre ses précautions au niveau de son navigateur, et d'autre part, le développeur de

l'application Web qui doit sécuriser son application. Ces mesures représentent le premier front amortisseur de l'attaque XSS. Le programmeur doit donc vérifier de manière approfondie et continue les contrôles de saisie de son application pour éviter les infiltrations XSS et le client doit se comporter de manière plus alerte lors de sa navigation. En particulier, pendant et après les phases de conception, maintenance et déploiement des applications Web, ces mesures de sécurité doivent être prises très tôt.

7.1. Mesures de sécurité côté développeurs ou propriétaires

Les développeurs des applications Web doivent respecter certaines mesures avant et après le déploiement de leurs applications pour diminuer le risque de l'injection XSS :

1. Les applications doivent être mises à jour régulièrement afin d'éviter les nouveaux types d'attaques [W2].

2. Les développeurs doivent disposer de guides explicatifs concernant les routines de filtrages de données, ces guides doivent expliquer comment ces routines vont être intégrés aux applications [2].

3. Les plateformes qui génèrent automatiquement des applications Web doivent être sécurisés. Entre autres, ces plateformes doivent contenir des routines de filtrage qui servent à contrôler les données saisies par l'utilisateur. Avec des plateformes sécurisées, les développeurs verront leur travail allégé, et vont concentrer leur énergie à rendre leurs applications plus performantes [2].

4. Il existe des outils commercialisés qui peuvent aider dans la détection des vulnérabilités XSS de manière automatique. Les développeurs d'applications Web peuvent en profiter pour corriger leurs applications [W2].

5. Les navigateurs Web utilisent souvent des API qui y sont intégrés, dans le but de permettre au navigateur de réaliser des actions complexes, telles que par exemple manipuler les objets de la structure DOM des pages HTML. Ces APIs

ont leurs propres failles qui peuvent être exploitées par des attaquants XSS. Il faut donc sécuriser ces APIs pour une utilisation plus sereine du navigateur. L'une des fonctionnalités les plus importante à implémenter par des APIs sécurisés est le typage des éléments des balises HTML. Cela aura pour effet la réduction du risque d'injection XSS liés aux balises HTML mal typées [2].

6. Toutes les zones de saisie, qui sont considérées comme des points d'injection potentiels pour les attaques XSS, doivent être contrôlées dans les applications web, les entrées utilisateurs doivent être décodées et filtrées [W2].

7.2. Mesures de sécurité côté clients ou utilisateurs

L'utilisateur novice doit aussi prendre sa responsabilité dans le contrôle et la sécurité de son navigateur, pour ne pas être une proie facile aux attaquants XSS ; parmi les mesures de sécurité qu'il doit entreprendre, on peut citer :

1. La mise à jour régulière des navigateurs pour renforcer leurs sécurités [W4].

2. Ils ne doivent installer que les plug-ins nécessaires et certifiés [W4].

3. Prendre l'habitude de saisir l'URL recherché directement dans la barre d'adresse du navigateur, et ne pas utiliser une source ou un lien suspect [W4].

4. Si un script n'est pas nécessaire, il faut le désactiver [W4].

5. Ne pas cliquer sur des liens provenant d'un tiers suspect, surtout ceux qui surviennent par email (cas du phishing) [4].

6. Il faut réduire au maximum la sauvegarde et l'activation des cookies sur les navigateurs [W4].

Concrètement, un utilisateur novice trouvera des difficultés à appliquer régulièrement ces règles ; le besoin d'automatisation de ces tâches s'avère alors indispensable. En

outre, quelques attaques XSS peuvent surpasser ses mesures et s'infiltrer malgré tout dans le navigateur de la victime. On peut citer parmi ces attaques, celle via laquelle un attaquant XSS se sert des scripts déjà utilisés par les applications Web pour déclencher son attaque, au lieu d'utiliser un script malicieux ; ceci démontre l'incapacité des méthodes de détection classique devant de telles menaces [2].

8. Conclusion

XSS est l'une des attaques les plus redoutables du Web. Dans le présent chapitre, nous avons présenté quelques concepts de base concernant cette attaque ; nous l'avons défini, situé parmi les attaques les plus prépondérantes et décrit ses différents types. Nous avons enfin présenté ses principales causes et discutés de quelques mesures de sécurité à considérer pour la contrer. Dans le chapitre suivant, nous allons présenter les travaux de recherche les plus récents et les techniques proposées dans la littérature pour la détection des vulnérabilités XSS.

CHAPITRE II.

DETECTION DES VULNERABILITES XSS

CHAPITRE II.

DETECTION DES VULNERABILITES XSS

La vulnérabilité XSS est un type de faille qui peut mettre en danger les applications Web. La vulnérabilité est exprimée par la possibilité d'injecter des scripts malveillants sous forme de valeurs, aux variables ou aux éléments des balises HTML dans le code source des applications [5]. Ces points sont aussi appelés *points d'injection* ou *points sensibles*. Sans filtrage approprié, les scripts injectés sont exécutés du côté client et peuvent causer le vol des données personnelles des utilisateurs.

Plusieurs techniques ont été proposées et adoptées pour détecter les vulnérabilités XSS. Ces techniques peuvent être classées en fonction de leurs intentions et de leurs natures. La figure 2.1 montre la classification proposée par Hannousse et al. [2]. Dans ce qui suit, nous exposons chacune de ces méthodes et nous présentons quelques travaux liés à chacune d'elles.

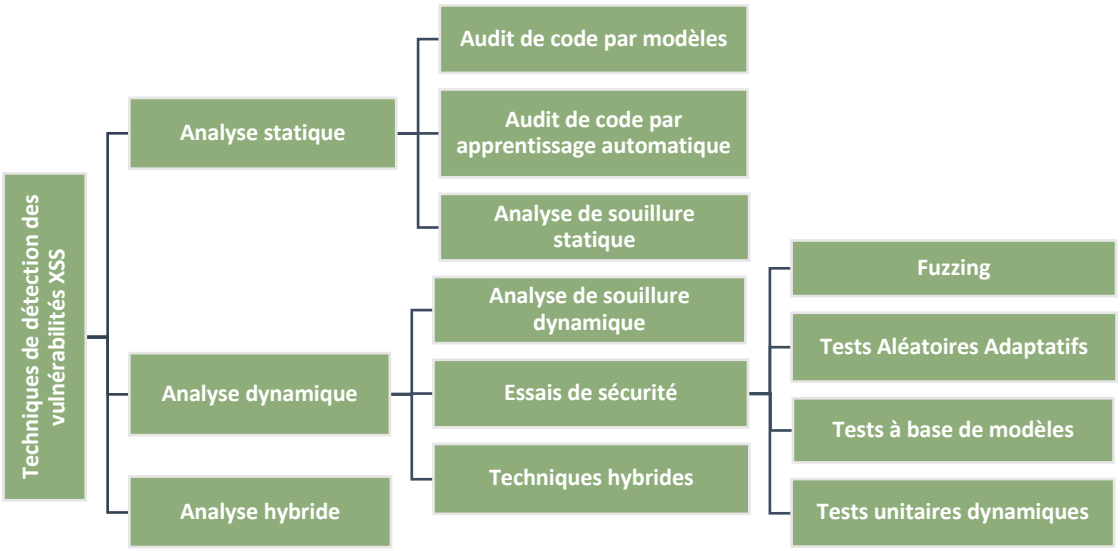

Figure 2.1. Classification des techniques de détection des vulnérabilités XSS [2]

1. Approches basées sur l'analyse statique

La méthode d'analyse statique a l'avantage de découvrir efficacement tous les chemins dans le code source, ce qui réduit efficacement le taux de faux négatifs. Cependant, les méthodes d'analyse statique ont également de nombreuses limites [2]. Dans de nombreux cas, les applications n'exposent pas leur code source pour des raisons de sécurité. Certaines applications appliquent même une technique de confusion de code pour empêcher la décompilation, ce qui rend le code plus difficile pour l'analyse statique. En raison de la nécessité de vérifier le code source, certains outils sont déployés côté serveur et ne peuvent détecter les attaques XSS à base de DOM, car il s'agit d'une vulnérabilité côté client et le code malveillant n'a pas besoin de passer par le serveur [5]. En plus, certaines applications Web comportent des codes dynamiques qui sont chargés uniquement au moment de l'exécution. Les méthodes statiques n'ont pas la capacité d'analyser ces types de code. Par exemple, la fonction *eval()* de PHP permet de transformer une chaîne de caractère en paramètre, en un script exécutable lors du chargement des pages dans le navigateur [2]. L'analyse statique ne permet pas de déduire la malveillance des valeurs de cette fonction sans exécution. Différentes approches utilisent l'analyse statique pour la détection des vulnérabilités XSS. Nous présentons ici les différentes techniques adoptées dans la littérature.

1.1. Audit de code par modèles

Les vulnérabilités des applications Web peuvent être vérifiées statiquement grâce à l'analyse de leurs modèles abstraits dérivés directement de leurs codes source [2]. Par exemple, Shar and Tan [8] proposent une technique similaire pour vérifier la cohérence des mesures de défenses adoptées par les développeurs des applications Web. Pour cela, les pages d'une application Web sont d'abord transformées en graphes de flux de contrôle. Les nœuds des graphes générés sont classés en fonction des mesures de sécurité associés par les développeurs. Un ensemble de règles bien définies est scruté pour vérifier l'adéquation des fonctionnalités de défense utilisées afin d'empêcher les attaques XSS sur ces nœuds.

1.2. Audit de code par apprentissage automatique

Tirant parti des vulnérabilités déjà découvertes et connues, l'apprentissage automatique peut être utilisé comme une approche d'audit de code afin de vérifier la vulnérabilité aux attaques XSS des pages Web développées [2]. Des ensembles de données constitués de pages Web sécurisées et vulnérables peuvent être collectés. Les caractéristiques de ces pages sont ensuite extraites des sources des pages Web via l'ingénierie de caractéristiques ou via l'apprentissage de caractéristiques. Les vecteurs de caractéristiques sont créés et passés aux classificateurs pour apprentissage et tests. Les nouvelles pages sont finalement classées par les classificateurs entrainés (ou modèles) en page vulnérables ou sécurisés. Dans cette approche, plusieurs types d'apprentissage automatiques peuvent être utilisés (apprentissage classique, ensembliste ou profond). Par exemple, Gupta et al. [9] ont utilisés une variante de classificateur de type arbre de décision (J48) et Li et al. [10] ont utilisés l'apprentissage profond à travers un réseau de mémoire à court terme bidirectionnel (BLSTM).

1.3. Analyse de souillure statique

Cette technique consiste à suivre les données d'entrée depuis les données sources vers les points d'injection dans les codes des applications Web sans être exécutées. Une vulnérabilité XSS est signalée si une donnée d'entrée atteint un point sensible sans être correctement validée. Jovanovic et al. [11] ont développé un outil typique nommé Pixy qui permet l'analyse statique des applications en parcourant tous les chemins des données en entrée vers les points sensibles de chaque page. Pixy est conçu pour la détection des vulnérabilités aux attaques XSS reflétés dans les applications Web PHP. Wang et al. [12] ont étendu Pixy pour la prise en charge des vulnérabilités aux attaques XSS stockées. L'algorithme proposé en [12] utilise à la fois des graphes de dépendance aux données et de dépendance au contrôle pour identifier les tranches de programme à l'origine des vulnérabilités aux attaques XSS stockées.

2. Approches basées sur l'analyse dynamique

Contrairement à l'analyse statique, les approches et outils d'analyse dynamique visent à découvrir les failles des applications Web pendant leurs exécutions. Ils permettent

de tester des applications Web sur des scénarios en temps réel. Les méthodes d'analyse dynamique se concentrent sur les informations acquises au moment de l'exécution. Elles détectent la présence d'une vulnérabilité XSS en fonction des réponses HTTP en envoyant des requêtes aux serveurs. L'avantage de la méthode d'analyse dynamique est qu'elle n'a pas besoin d'obtenir le code source de la page web et le taux de faux positifs est faible [2]. Mais il y a aussi quelques limitations lorsque le nombre de points d'injection augmente ; il faut plus de temps pour détecter toutes les vulnérabilités XSS. Cela peut rendre ces méthodes impossibles à appliquer en pratique. En plus, les méthodes d'analyse dynamique peuvent obtenir un taux élevé de faux négatifs parce que les cas de tests ne couvrent pas toujours toutes les situations possibles [5]. Plusieurs techniques dynamiques sont utilisées dans la littérature. Dans ce qui suit, nous décrivons chacune de ces approches.

2.1. Analyse de souillure dynamique

Contrairement à sa version statique, la version dynamique consiste à marquer les données sensibles en entrée et à suivre leurs propagations lorsque les applications testées sont en cours d'exécution. Une vulnérabilité est signalée lorsque les données d'entrée atteignent des emplacements ou des puits de programme prédéfinis [2]. Martin et Lam [3] ont proposé une technique d'analyse de souillure dynamique armée d'un vérificateur de modèle pour prouver le succès des attaques. Le système proposé est conçu pour gérer les vulnérabilités basées sur l'injection des scripts, y compris les injections XSS et SQL. Les vulnérabilités testées sont décrites à l'aide d'un langage spécifique nommé PQL. Les utilisateurs doivent fournir les spécifications PQL des vulnérabilités testées avec un ensemble de valeurs de paramètres de requête d'entrée. Le système explore l'espace des tests d'entrée possibles et surveille l'exécution de l'application sous chaque entrée. Le vérificateur du modèle indique si les tests d'entrée atteignent avec succès les points d'injection spécifiés et signale les chemins d'attaque correspondants en cas de succès. Malheureusement, l'efficacité de cette solution repose sur la justesse des spécifications fournies par les développeurs. Elle échoue à la détection des vulnérabilités avec la présence de désinfectants avancés et elle ne s'applique qu'aux applications Web basées sur Java [2].

2.2. Essais de sécurité

Les tests de sécurité font référence à l'ensemble des techniques fournissant des preuves que les applications Web sont sûres et fiables et qu'elles n'acceptent pas les entrées non autorisées. Ils testent l'impact d'entrées malveillantes ou inattendues sur les fonctionnalités des applications Web lorsqu'elles sont en cours d'exécution. Plusieurs techniques ont été utilisées pour la détection des vulnérabilités XSS :

2.2.1. Fuzzing

Consiste à révéler les vulnérabilités des applications Web en injectant des entrées invalides, malformées ou inattendues et à observer les exceptions sur leurs comportements [2]. McAllister et al. [14] ont proposé un fuzzer guidé par un ensemble collecté de cas d'utilisation du serveur. Le fuzzer remplace les valeurs des paramètres d'entrée des requêtes du monde réel par des chaînes malveillantes provenant d'une base de données.

2.2.2. Tests aléatoires adaptatifs

Au lieu de tester des applications Web à l'aide d'un ensemble arbitraire de cas de tests, des tests aléatoires adaptatifs peuvent être adoptés pour optimiser le processus de test. Par conséquent, uniquement les cas de tests les plus distinctifs et probablement réussis sont sélectionnés pour chaque étape [2]. Lv et al. [15] ont adopté une sélection de cas de tests basée sur la distance. A chaque itération, le test sélectionné doit avoir une distance de similarité minimale avec ceux sélectionnés précédemment. Cela permet d'éviter de perdre le temps avec des tests quasi-similaires.

2.2.3. Tests à base de modèles

Dans cette technique, des modèles entraînés sont utilisés pour distinguer les pages vulnérables des pages sécurisés [2]. Avancini et Ceccato [16] ont développé un oracle de test nommé *Circe* pour tester les vulnérabilités XSS des applications Web. En tant qu'oracle de test, un modèle est construit sur la base de la structure de pages Web générées à l'aide des entrées valides. Des entrées malveillantes d'une bibliothèque sont injectées et les structures des pages résultantes sont mises en correspondance avec

celles générées à partir d'entrées valides. Toute inconsistance est considérée comme une vulnérabilité potentielle.

2.2.4. Tests unitaires dynamiques

L'approche à base de tests unitaires est une autre technique d'analyse dynamique utilisée pour la détection des vulnérabilités. Dans les tests unitaires dynamiques, une unité de programme est exécutée de manière isolée avec des entrées sélectionnées, les résultats sont ensuite comparés aux résultats attendus [2]. Mohammadi et al. [17] ont proposé l'utilisation de cette technique pour la détection des vulnérabilités XSS dans les applications Web de type JSP (JavaServer Pages). Les tests unitaires pour chaque page sont automatiquement construits et exécutés à l'aide d'un système d'exécution de tests unitaires. Pour les entrées de test, une grammaire d'attaque bien établie est utilisée pour la génération des entrées valides et malicieux.

3. Approches hybrides

Les méthodes d'analyse hybrides présentent les avantages de l'analyse statique et de l'analyse dynamique. Elles peuvent non seulement détecter tous les chemins dans le code source, mais aussi d'obtenir un faible taux de faux positifs. Dans l'analyse hybride, l'analyse statique est utilisée pour trouver les chemins parcourus par les données en entrées, ce qui améliore la vitesse de détection. L'analyse dynamique est principalement utilisée pour confirmer les vulnérabilités aux attaques XSS en testant chaque chemin sur des données d'entrés malicieux [2]. Cependant, les méthodes d'analyse hybride héritent également les défauts de l'analyse statique. Certaines méthodes d'analyse hybride ne peuvent être appliquées qu'à un seul langage, ce qui les rend peu populaires [2]. Pan et Mao [18] ont proposé une alternance d'analyse statique et dynamique pour la détection des vulnérabilités XSS causées par l'extension du navigateur Greasemonkey[1]. Greasemonkey est une extension multiplateforme qui permet aux utilisateurs d'écrire des JavaScripts pour personnaliser les apparences et les comportements des pages Web. Dans la phase d'analyse statique, les scripts utilisateur de l'extension Greasemonkey sont filtrés par des modèles correspondants liés à des directives d'octroi de privilèges spécifiques. De plus, un analyseur statique

[1] Greasemonkey : https://www.greasespot.net

est utilisé pour identifier les chemins source-puits et permettre l'élimination des contenus sûrs. Les scripts restants sont soumis à une analyse dynamique (c'est-à-dire une exécution symbolique) pour confirmer leurs vulnérabilités. L'approche a une surcharge importante et nécessite une assistance manuelle pour la génération de séquences d'événements suspects.

Van Acker et al. [19] ont utilisé une combinaison d'analyses statiques et dynamiques pour la détection des vulnérabilités des applications Internet riches (RIA) aux attaques XSS. L'analyse statique est utilisée pour identifier automatiquement l'ensemble de variables sensibles à ActionScript, utilisées dans les fichiers susceptibles d'être affectés par des entrées utilisateur. Cette étape est effectuée en décompilant les fichiers SWF (Small Web Format) et en utilisant la correspondance à base d'expressions régulières. L'analyse dynamique est utilisée pour tester les variables identifiées dont le but est de découvrir les vulnérabilités XSS. Des entrées malveillantes sont générées à l'aide de 10 modèles et injectées en tant que valeurs dans les variables sensibles et vérifiées pour les effets malveillants.

4. Conclusion

Dans ce chapitre, nous avons présenté les différentes techniques utilisées pour la détection des vulnérabilités XSS. Trois techniques de bases sont explorées dans la littérature : analyse statique, dynamique et hybride. Chacune d'elle a ses avantages et ses limites. A titre indicatif, nous avons présenté une ou deux études récentes sur chaque méthode. Bien qu'il existe de nombreuses méthodes pour détecter les vulnérabilités XSS, il est toujours très difficile de détecter toutes les vulnérabilités XSS dans une application Web simple. Cela nous a motivé pour tenter de développer un outil pour la détection automatique des vulnérabilités XSS, la conception de cet outil fait l'objet du chapitre suivant.

CHAPITRE III.

UNE APPROCHE HYBRIDE POUR LA DETECTION DES VULNERABILITES XSS

CHAPITRE III.

UNE APPROCHE HYBRIDE POUR LA DETECTION DES VULNERABILITES XSS

Ce chapitre est divisé en deux parties. Dans sa première partie, nous allons décrire en détail, notre approche proposée pour la détection des vulnérabilités des applications Web aux attaques XSS. L'approche proposée est une approche hybride. La détection hybride inclut deux phases principales : une phase *statique* et une autre *dynamique*. La phase statique permet de parcourir et identifier tous les points d'injections constituant des sources de vulnérabilité, alors que la phase dynamique confirme si ces points sont réellement vulnérables aux injections des scripts malicieux. Notre approche vise à détecter des vulnérabilités XSS en se basant sur un ensemble de tests simulant des attaques réelles. Ces tests sont passés successivement aux applications en cours d'exécution. Dans cette partie du chapitre, nous détaillons notre approche et ses différentes phases et étapes. Un prototype nommé « *XSS Checker* » est implémenté et testé sur des pages réelles, le fonctionnement de ce prototype est détaillé dans la deuxième partie de ce chapitre, ainsi qu'un aperçu sur les résultats des tests obtenus.

1. Description de l'approche proposée

La figure 3.1 montre le processus général de l'approche hybride développée pour la détection des vulnérabilités des applications Web aux attaques XSS. Ce processus comporte cinq étapes fondamentales : exploration des points d'injections présents dans la page de l'application en test, construction des attaques, lancement des

attaques, analyse de la réponse des serveurs pour chaque attaque et génération du rapport. Dans le reste de ce chapitre nous détaillons chacune des étapes.

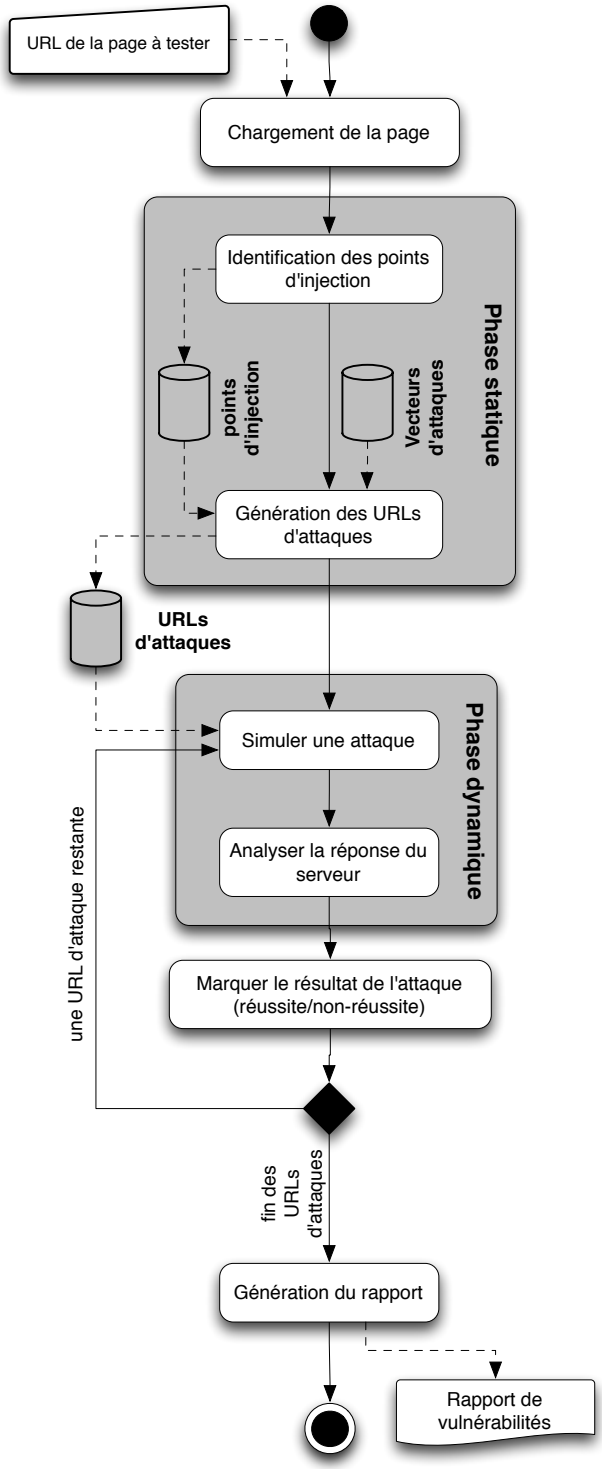

Figure 3.1. Processus de détection des vulnérabilités XSS

À partir d'une URL donnée en entrée, une analyse statique commence par déterminer tous les endroits de la page permettant l'insertion des données par l'utilisateur. Spécifiquement, ce processus vise à déterminer tous les champs de saisies disponibles sur la page en test. Ces champs sont généralement les sources des vulnérabilités XSS. L'objectif est de vérifier si des filtres appropriés sont déjà établis pour chaque entrée utilisateur. Pour cela, un ensemble de tests, simulant des attaques réelles, est envoyé comme valeur à ces champs, et la réponse des serveurs est analysée ; si le vecteur d'attaques envoyé apparait dans la réponse du server, une vulnérabilité XSS est signalée.

1.1. Identification des points d'injections potentiels

L'objectif de cette étape est d'identifier tous les endroits dans les pages Web de l'application où les utilisateurs peuvent saisir des données et les envoyer sous forme de requêtes au serveur. Cette phase consiste à identifier tous les éléments HTML de type `input` ou `textarea` dans la page ; cela inclut des éléments sans type ou avec un type spécifique comme `text` ou `search`. Une expression régulière est utilisée dans ce but.

1.2. Création des URLs d'attaques

La création des URLs d'attaques consiste à insérer des scripts d'attaques (appelés aussi *vecteurs d'attaques*) sous forme de chaînes de caractères aux points d'injections identifiés dans la phase précédente. Prenons comme exemple, le moteur de recherche Google ; la page d'accueil de cette application est accessible à travers l'URL de base suivant :

```
https://www.google.com/
```

Cette page contient un champ de saisie nommé `q` ; une URL d'attaque est construite en ajoutant un vecteur d'attaque XSS comme valeur à ce champ. Une URL d'attaque typique est de la forme :

```
https://www.google.com/?q=<script>alert('XSS')</script>
```

Dans le cas où la page comporte plusieurs champs de saisies, le script malicieux est ajouté à tous ces champs avant de soumettre le formulaire au serveur. Le tableau 3.1

suivant montre la liste des vecteurs d'attaques considérés pour la génération des URLs d'attaques. Afin d'assurer un temps de réponse raisonnable, un nombre limité de ces attaques (7 dans notre cas) a été sélectionné. Ces vecteurs d'attaques visent à générer des boites de dialogue dès le chargement des pages Web dans le navigateur. Les vecteurs d'attaques sont sélectionnés soigneusement dans la liste des attaques XSS connues[1] pour couvrir la majorité des formes d'attaques et les tactiques de codage utilisées par les attaquants.

#	Vecteur d'attaque	Description
1	`<script>alert("xss")</script>`	Script clair.
2	`<script /*%00*/>/*%00*/alert(1)/*%00*/</script /*%00*/ &#34;&#62;<h1/onmouseover='\u0061lert(1)'>%00`	Script déguisé.
3	`')%3Balert(1)%3Bvar b=('`	Scripts encodés.
4	`&lt;INPUT TYPE=\"IMAGE\" SRC=\"javascript&#058;alert('XSS');\"&gt;`	
5	`<SCRIPT String.fromCharCode(97, 108, 101, 114, 116, 40, 49, 41)</SCRIPT>`	
6	`<img src=# onerror\x3D"javascript:alert(1">`	Script inclut dans une balise HTML.
7	`<--`<img/src=` onerror=alert(1)> --!>`	Script inclut dans un commentaire.

Tableau 3.1. Liste représentative des attaques XSS sélectionnées

1.3. Simulation des attaques

La simulation des attaques consiste à envoyer des requêtes URL malformées (URL d'attaques) au serveur hébergeant l'application Web cible et à attendre la réponse du serveur. Comme plusieurs tests doivent être élaborés sur la même application, plusieurs URLs sont envoyées et testées, vérifiées une à la fois.

1.4. Vérification des réponses du serveur

Cette phase consiste à faire correspondre les scripts inclus dans l'URL d'attaque à ceux renvoyés dans la réponse du serveur. S'il y a correspondance, l'attaque XSS est considérée réussite et donc une vulnérabilité XSS est signalée. Pour des raisons de

[1] Liste des vecteurs d'attaques XSS connus : https://github.com/payloadbox/xss-payload-list

simplification, nous avons utilisé des attaques visant à ouvrir une boite de dialogue lors du chargement de la page ; si la boite de dialogue s'affiche, le vecteur d'attaque n'a pas été filtré proprement, ce qui confirme une vulnérabilité aux attaques XSS.

1.5. Génération de rapport

Cette phase consiste à informer le développeur du résultat des tests. Dans notre application, le développeur est informé si la page en test est vulnérable ou pas aux attaque XSS. Si au moins une des attaques de test est réussite, une vulnérabilité est signalée, sinon, la page est considérée comme sûre.

2. Implémentation de l'approche proposée

Dans cette section, nous détaillons les aspects techniques liés au développement d'un prototype « *XSS Checker* » implémentant notre approche décrite dans la section 1.

2.1. Outils et langages de développement

Pour le développement de *XSS Checker*, un ensemble des outils a été utilisé. Nous décrivons chacun de ces outils et leurs rôles dans la mise en œuvre de notre application :

- ***JavaScript :*** est un langage de script utilisé par de nombreux navigateurs pour réaliser des actions dynamiques sur les pages Web qui ne peuvent pas être codés par HTML et CSS. JavaScript est le langage le plus utilisé pour coder la majorité des attaques d'injections, y compris XSS [2]. Dans notre application, le JavaScript est utilisé pour modéliser des attaques XSS affichant des boites de dialogues de manière automatique sur les pages Web testées.

- ***Node.js*** : est un environnement d'exécution open source et multiplateforme utilisé pour exécuter des applications Web en dehors du navigateur du client [20]. Dans notre application, Node.js est utilisé pour la mise en œuvre de notre application. Il est principalement utilisé pour simuler les actions d'un serveur Web qui envoie et reçoit des requêtes HTTP, simule l'exécution d'un navigateur qui charge des pages

Web et identifie la présence ou l'absence des vulnérabilités aux attaques XSS. Pour cela nous avons utilisé les modules suivants :

a. ***Module http* :** il inclut des classes, des méthodes et des événements pour créer un serveur HTTP par Node.js [20].

b. ***Module Puppeteer*:** une bibliothèque qui fournit une API de haut niveau pour contrôler un navigateur Chrome [20]. Elle est utilisée pour simuler les actions d'un navigateur qui prend en charge le chargement des pages testées avec des URLs d'attaques.

c. ***Module Query-string*:** fournit des utilitaires pour l'analyse et le formatage des URLs construits sous forme de chaînes de caractères. [20] Dans notre cas, il est utilisé pour construire les URL d'attaques.

d. ***Module body-parser*:** un middleware d'analyse du corps de Node.js. Il est responsable de la préparation des requêtes entrant dans un middleware avant d'être soumises à des analyses spécifiques et personnalisées, sinon des erreurs se produiront [20].

e. ***Module express*:** une bibliothèque responsable de mettre en place des middlewares pour répondre aux requêtes HTTP, définir une table de routage utilisée pour effectuer différentes actions en fonction de la méthode HTTP et de l'URL [20].

- ***Visual Studio Code (VSCode)* :** est un éditeur de code simplifié, qui est gratuit et développé en open source par Microsoft. Il est multiplateforme, fonctionne sous Windows, Mac OS et Linux. Il fournit aux développeurs à la fois un environnement de développement intégré avec des outils permettant de faire avancer les projets techniques : de l'édition à la construction, jusqu'au débogage. Les fonctionnalités proposées par VSCode sont nombreuses [W5]. Il nous a également permis de créer notre application en profitant de son support pour JavaScript et Node.js.

2.2. Mode d'installation

Afin d'utiliser notre application, il est indispensable d'installer l'environnement Node.js. Sous Windows, cela peut se faire simplement par le téléchargement et le lancement du fichier installateur : *Windows Installer (.msi)* depuis le site officiel de Node.js (https://nodejs.org/en/download/). Ensuite, il faut installer les modules décrits dans la section 7.2. Pour simplifier la tâche, nous avons configuré un fichier nommé *package.json* qui liste l'ensemble de ces modules ainsi que leurs versions compatibles :

```
"dependencies":{
                        "express": "^4.17.1",
                        "body-parser": "^2.0.4",
                        "puppeteer": "^2.0.0",
                        "query-string": "^6.10.1"
}
```

L'installation de ces dépendances peut se faire automatiquement en utilisant la commande : `npm install`.

2.3. Mode de fonctionnement

Pour lancer le prototype *XSS Checker*, il suffit de lancer la commande `npm start` et lancer le navigateur avec l'adresse locale : http://localhost:4000/. L'interface graphique de XSS Checker s'affiche comme le montre la figure 3.2.

Pour tester la vulnérabilité d'une page Web aux attaques XSS, il suffit de tapez l'URL de cette page dans le champ de saisie de la page principale et cliquer sur le bouton Check. Selon le cas, l'interface change et indique si la page en test est vulnérable ou pas, comme le montrent les figures 3.3 et 3.4 respectivement.

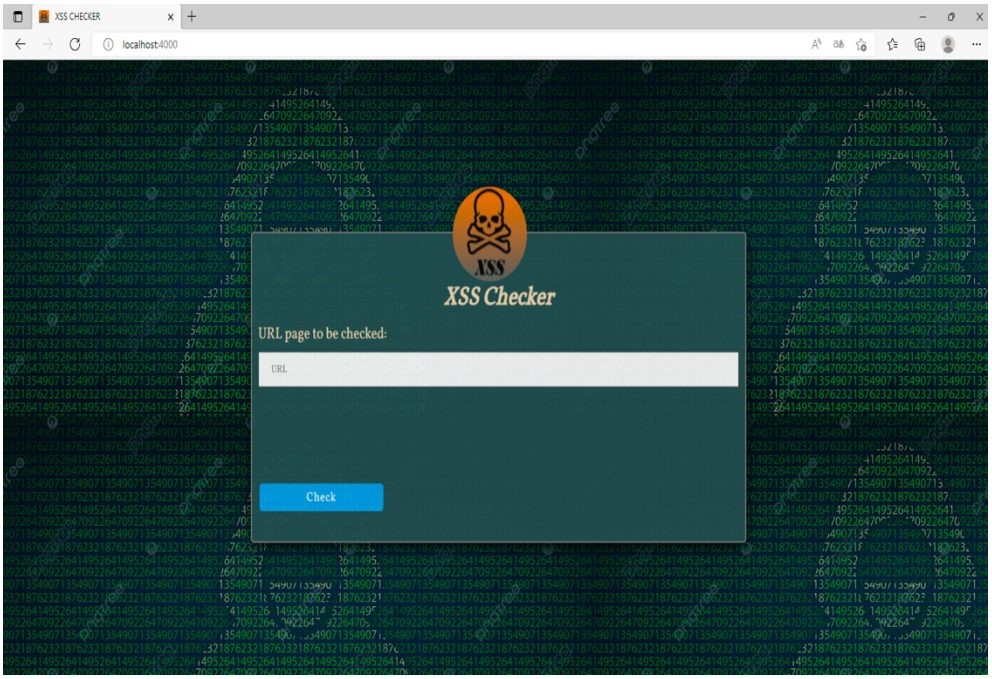

Figure 3.2. Page principale de XSS Checker.

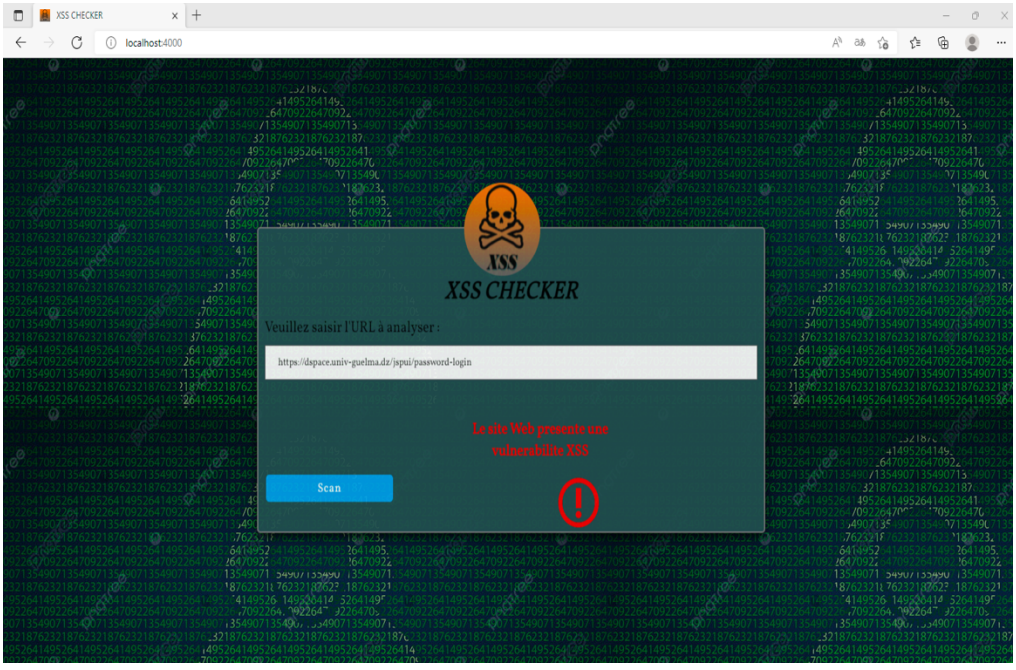

Figure 3.3. XSS Checker dans le cas d'une page vulnérable.

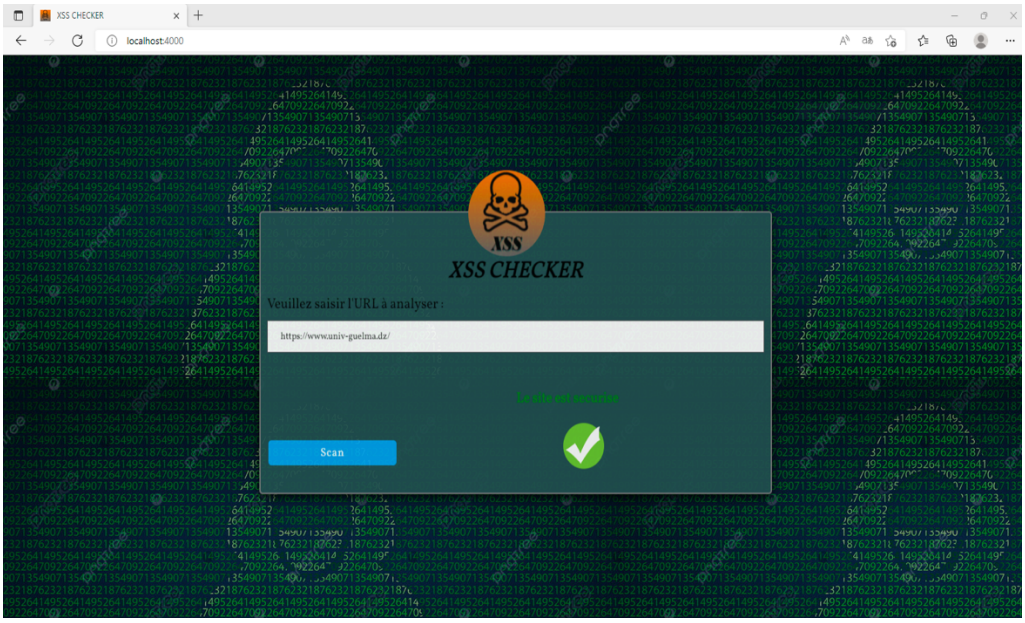

Figure 3.4. XSS Checker dans le cas d'une page non vulnérable.

3. Expérimentation

Nous avons expérimenté XSS Checker avec un certain nombre de pages Web réelles. Pour des raisons de sécurité, nous ne montrons qu'un nombre limité de ces pages, le résultat des tests de chacune est inclus dans le tableau 3.2:

#	Pages testés	Résultat du test
1	https://dspace.univ-guelma.dz/jspui/password-login	Vulnérable
2	https://www.albaraka-bank.dz	Sûre
3	https://www.univ-guelma.dz	Sûre
4	https://xss-game.appspot.com/level1/frame	Vulnérable
5	https://xss-game.appspot.com/level2/frame	Vulnérable
6	https://badrbanque.dz	Sûre

Tableau 3.2. Quelques URLs testées par XSS Checker

4. Conclusion

La méthode de détection de la vulnérabilité XSS conçue dans ce chapitre possède de faibles exigences environnementales, un taux de détection élevé et une architecture flexible. La méthode inclut deux phases complémentaires de détection : (1) une phase statique qui consiste à identifier tous les points d'injections potentiels en analysant le code source de chaque page de l'application à tester, (2) une phase dynamique qui consiste à simuler des attaques sur l'application et surveiller les résultats du serveur. Comparée à des outils similaires de détection dynamique des vulnérabilités de type XSS, l'approche proposée couvre plus de vulnérabilités, qui ne peuvent être détectées par les approches statiques et dynamiques (Voir chapitre II). En plus, la différence majeure entre notre approche et les autres approches hybrides réside dans la sélection des vecteurs d'attaques diversifiés qui couvrent la majorité des attaques courantes.

CONCLUSION GENERALE

CONCLUSION GENERALE

Les individus et les entreprises deviennent de plus en plus dépendants de l'Internet : les applications Web représentent le moyen d'accès principal aux services en ligne ; elles sont aussi devenues une plateforme standard pour la représentation, le stockage et la manipulation des données. De ce fait, les technologies Web deviennent de plus en plus sophistiquées avec des codes sources de plus en plus longs, ce qui augmente le nombre de leurs vulnérabilités. Elles sont donc une proie facile pour les attaquants, et les cyber-attaques préoccupent de plus en plus le monde de l'entreprise, des gouvernements et en général, la sécurité des biens et des personnes. Le cross-site scripting, ou XSS, est une attaque qui exploite ces vulnérabilités de manière simple et efficace ; via les zones de saisie et les formulaires d'inscription en général. Vu la multitude d'outils et de langages de programmation qui sont en constante évolution, la détection de ce type d'attaque devient de plus en plus difficile.

Dans ce cadre, nous avons proposé une approche hybride pour la détection des vulnérabilités des applications Web aux différentes attaques XSS. Cette approche se base sur l'analyse du contenu des pages Web afin de déterminer les différents points d'injection possibles. Ces points d'injections sont examinés par des tests sur un ensemble d'attaques XSS diverses. Le comportement des applications est observé pour déterminer le succès ou l'échec de chaque vecteur d'attaque testé. Le succès d'un vecteur d'attaque indique une faille et donc l'application est signalée vulnérable aux attaques XSS et nécessite d'être réparée. Les expérimentations élaborées par l'application de notre approche aux applications Web réelles, nous a montré que différentes applications, souvent considérées comme des applications sécurisées sont vulnérables aux attaques XSS.

Au terme de ce travail, nous espérons continuer dans ce domaine en améliorant l'approche et l'outil proposés et pallier leurs limites. Cela inclut l'intégration d'un programme d'exploration du Web (Web crawler en anglais) afin de permettre

l'identification des différents points d'injection, non seulement dans des pages individuelles mais dans toutes les pages de l'application Web, en parcourant l'arborescence de l'application et en spécifiant uniquement l'adresse de sa page d'accueil. En plus, nous planifions d'améliorer la qualité du rapport de vulnérabilité généré par notre outil en indiquant le détail des vulnérabilités identifiées ; notamment, la page et le(s) champs vulnérables ainsi que la liste des attaques réussites.

BIBLIOGRAPHIE & WEBOGRAPHIE

BIBLIOGRAPHIE

[1] Amy Shuen, *Web 2.0: A Strategy Guide Business thinking and strategies behind successful Web 2.0 implementations.* O'Reilly Media, Inc., 2008. ISBN: *0596529961.*

[2] Abdelhakim Hannousse, Salima Yahiouche, Mohamed Cherif Nait-Hamoud, Twenty-two years since revealing cross-site scripting attacks: a systematic mapping and a comprehensive survey. arXiv, CoRR abs/2205.08425, 2022. doi: 10.48550/arXiv.2205.08425.

[3] Abdelhakim Hannousse, Salima Yahiouche, *Towards benchmark datasets for machine learning based website phishing detection: An experimental study*, Engineering Applications of Artificial Intelligence, Vol. 104, 2021, 104347, doi: 10.1016/j.engappai.2021.104347.

[4] Guillaume Harry, Failles de sécurité des applications Web Principes, parades et bonnes pratiques de développement, Rapport technique, hal-00736013, 2012.

[5] Jeremiah Grossman, *XSS Attacks: Cross-site Scripting Exploits and Defense*, Syngress, 2007. ISBN: 1597491543.

[6] Park Joon S. and Ravi Sandhu, *Secure cookies on the Web,* IEEE internet computing, Vol. 4, N. 4, pp. 36-44, 2000.

[7] Keith Jeremy and Jeffrey Sambells. *DOM scripting: Web design with Javascript and the document object model*. Apress, Second Edition, 2011. ISBN: 1430233907.

[8] L. Shar and H. Tan, *Auditing the XSS defence features implemented in web application programs*, IET Software, Vol. 6 N. 4, pp. 377–390, 2012. doi:10.1049/iet-sen.2011.0084.

[9] S. Gupta, B. Gupta, *XSS-safe: A server-side approach to detect and mitigate cross-site scripting (XSS) attacks in JavaScript code*, Arabian Journal for Science and Engineering, Vol. 41, n. 3, pp. 897–920, 2016. doi:10.1007/s13369-015-1891-7.

[10] C. Li, Y. Wang, C. Miao, C. Huang, *Cross-site scripting guardian: A static xss detector based on data stream input-output association mining*, Applied Sciences (Switzerland) Vol. 10, n. 14, pp. 1–20, 2020. doi:10.3390/app10144740.48.

[11] N. Jovanovic, C. Kruegel, E. Kirda, *Pixy: A static analysis tool for detecting web application vulnerabilities*, In Proceedings of the 2006 IEEE Symposium on Security and Privacy, SP, IEEE, Berkeley/Oakland, CA, USA, pp. 1–6, 2006. doi:10.1109/SP.2006.29.

[12] Y. Wang, Z. Li, T. Guo, *Program slicing stored XSS bugs in web application*, In Proceedings of the 5th International Conference on Theoretical Aspects of Software Engineering, TASE, IEEE, Xi'an, China, pp.191–194, 2011. doi:10.1109/TASE.2011.43.

[13] M. Martin and M. Lam, *Automatic generation of XSS and SQL injection attacks with goal-directed model checking*, In proceedings of the 17th USENIX Security Symposium, USENIX, USENIX, San Jose, CA, USA, pp. 31–43, 2008.

[14] S. McAllister, E. Kirda, C. Kruegel, *Leveraging user interactions for in-depth testing of web applications*, In proceedings of the International Workshop on Recent Advances in Intrusion Detection, RAID, Springer, Cambridge, MA, USA, pp. 191–210, 2008. doi:10.1007/978-3-540-87403-4_11.

[15] C. Lv, L. Zhang, F. Zeng, J. Zhang, *Adaptive random testing for XSS vulnerability*, In proceedings of the 26th Asia-Pacific Software Engineering Conference, APSEC, IEEE, Putrajaya, Malaysia, pp. 63–69, 2019. doi:10.1109/APSEC48747.2019.00018.

[16] A. Avancini, M. Ceccato, *Circe: A grammar-based oracle for testing cross-site scripting in web applications*, In proceedings of the 2013 20th Working Conference on Reverse Engineering, WCRE, IEEE, Koblenz, Germany, pp. 262–271, 2013. doi:10.1109/WCRE.2013.6671301.

[17] M. Mohammadi, B. Chu, H. Lipford, *Detecting cross-site scripting vulnerabilities through automated unit testing*, In proceedings of the 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS, IEEE, Prague, Czech Republic, pp. 364–373, 2017. doi:10.1109/QRS.2017.46.

[18] J. Pan, X. Mao, *Detecting DOM-sourced cross-site scripting in browser extensions*, In proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME, IEEE, Shanghai, China, pp. 24–34, 2017. doi:10.1109/ICSME.2017.11.

[19] S. Van Acker, N. Nikiforakis, L. Desmet, W. Joosen, F. Piessens, *Flashover: Automated discovery of cross-site scripting vulnerabilities in rich internet applications*, In proceedings of the 7th ACM Symposium on Information, Computer and Communications Security, ASIACCS, ACM, Seoul, Korea, pp. 12–13, 2012. doi:10.1145/2414456.2414462.

[20] Basarat Ali Syed. *Beginning Node.js.* 1st edition, Apress, USA, 2014.

WEBOGRAPHIE

[W1] Open Web Application Security Project (OWASP), https://owasp.org, last access May 2022.

[W2] Amit Klein, DOM Based Cross Site Scripting or XSS of the Third Kind: A look at an overlooked flavor of XSS, webappsec, http://www.webappsec.org/projects/articles/071105.txt, July 2005, last access May 2022.

[W3] Kaspersky.org, Qu'est-ce qu'une attaque XSS (Cross-Site Scripting) ? Définition et explication, https://www.kaspersky.fr/resource-center/definitions/what-is-a-cross-site-scripting-attack, last access May 2022.

[W4] Guy Podjarny, *XSS Attacks: The Next Wave*, https://snyk.io/blog/xss-attacks-the-next-wave/, June 2017, last access May 2022.

[W5] Visual Studio Code documentation. https://code.visualstudio.com/docs, last access May 2022.

[W6] Riadh Hajji, Les injections HTML : XSS, https://apcpedagogie.com/les-injections-html-xss/, last access May 2022.