View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by OpenGrey Repository

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :
Institut National des Sciences Appliquées de Toulouse (INSA Toulouse)

Discipline ou spécialité :

Systèmes Informatiques

Présentée et soutenue par :
Rim AKROUT

le : jeudi 18 octobre 2012

Titre :

Analyse de vulnérabilités et évaluation de systèmes de détection d'intrusions pour les applications Web

---

JURY

Carlos AGUILAR MELCHOR et Christophe BIDAN : Rapporteurs
Abdelmalek BENZEKRI et Alain RIBAULT : Examinateurs
Mohamed KAÂNICHE et Vincent NICOMETTE : Directeurs de thèse
Eric ALATA : Invité

Ecole doctorale :
Systèmes (EDSYS)

Unité de recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS

Directeur(s) de Thèse :

Mohamed KAÂNICHE et Vincent NICOMETTE

Rapporteurs :

Carlos AGUILAR MELCHOR et Christophe BIDAN

Résumé

Analyse de vulnérabilités et évaluation de systèmes de détection d'intrusions pour les applications Web

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

Dans cette thèse nous proposons une nouvelle méthode, basée sur des techniques de clustering de pages Web, qui permet d'identifier les vulnérabilités à partir de l'analyse selon une approche boîte noire de l'application cible. Chaque vulnérabilité identifiée est réellement exploitée ce qui permet de s'assurer que la vulnérabilité identifiée ne correspond pas à un faux positif. L'approche proposée permet également de mettre en évidence différents scénarios d'attaque potentiels incluant l'exploitation de plusieurs vulnérabilités successives en tenant compte explicitement des dépendances entre les vulnérabilités. Nous nous sommes intéressés plus particulièrement aux vulnérabilités de type injection de code, par exemple les injections SQL. Cette méthode s'est concrétisée par la mise en œuvre d'un nouveau scanner de vulnérabilités et a été validée expérimentalement sur plusieurs exemples d'applications vulnérables.

Nous avons aussi développé une plateforme expérimentale intégrant le nouveau scanner de vulnérabilités, qui est destinée à évaluer l'efficacité de systèmes de détection d'intrusions pour des applications Web dans un contexte qui soit représentatif des menaces auxquelles ces applications seront confrontées en opération. Cette plateforme intègre plusieurs outils qui ont été conçus pour automatiser le plus possible les campagnes d'évaluation. Cette plateforme a été utilisée en particulier pour évaluer deux techniques de détection d'intrusions développées par nos partenaires dans le cadre d'un projet de coopération financé par l'ANR, le projet DALI.

Mots-clés : Application Web, Attaque et vulnérabilités, Évaluation, Système de détection d'intrusions, Scanner Web,

Web applications vulnerability analysis and intrusion detection systems assessment

With the increasing development of Internet, Web applications have become increasingly vulnerable and exposed to malicious attacks that could affect essential properties such as confidentiality, integrity or availability of information systems. To cope with these threats, it is necessary to develop efficient security protection mechanisms and testing techniques (firewall, intrusion detection system, Web scanner, etc..). The question that arises is how to evaluate the effectiveness of such mechanisms and what means can be implemented to analyze their ability to correctly detect attacks against Web applications.
This thesis presents a new methodology, based on web pages clustering, that is aimed at identifying the vulnerabilities of a Web application following a black box analysis of the target application. Each identified vulnerability is actually exploited to ensure that the identified vulnerability does not correspond to a false positive. The proposed approach can also highlight different potential attack scenarios including the exploitation of several successive vulnerabilities, taking into account explicitly the dependencies between these vulnerabilities. We have focused in particular on code injection vulnerabilities, such as SQL injections. The proposed method led to the development of a new Web vulnerability scanner and has been validated experimentally based on various vulnerable applications.

We have also developed an experimental platform integrating the new web vulnerability scanner, that is aimed at assessing the effectiveness of Web applications intrusion detection systems, in a context that is representative of the threats that such applications face in operation. This platform integrates several tools that are designed to automate as much as possible the evaluation campaigns. It has been used in particular to evaluate the effectiveness of two intrusion detection techniques that have been developed by our partners of the collaborative project DALI, funded by the ANR, the French National Research Agency.

Keywords : Web applications, Web attacks, Vulnerabilities, Evaluation, Intrusion detection System, Web vulnerability scanners.

Dédicace

A la mémoire de mon père, le destin ne nous a pas laissé le temps de jouir de ce bonheur ensemble et de t'exprimer tout mon respect.

A ma mère pour tout son amour et son soutien. Quoique je puisse dire, je ne peux exprimer mes sentiments d'amour et de respect à ton égard et ma gratitude.

A ma sœur, mes deux frères, mes deux belles sœurs et mon beau frère, votre aide, votre générosité, votre soutien ont été pour moi une source de courage et de confiance.

Qu'il me soit permis aujourd'hui de vous assurer mon profond amour et ma grande reconnaissance.

A tous ceux qui comptent pour moi...

Malgré la distance sans vous aucune réussite n'aurait été possible. Aucun mot ne saurait retranscrire ici le bonheur que vous m'avez toujours apporté.

Je crois que vous êtes fiers de moi, autant que moi de vous.

Rim AKROUT

Remerciement

C'est avec un grand plaisir que je réserve cette page en signe de gratitude et de profonde reconnaissance à tous ceux qui ont bien voulu apporter l'assistance nécessaire au bon déroulement de ce travail.

Cette thèse a été effectuée au sein de l'équipe **T**olérance aux fautes et **S**ûreté de **F**onctionnement informatique (TSF) au **L**aboratoire d'**A**nalyse et d'**A**rchitecture des **S**ystèmes au **C**entre **N**ational de la **R**echerche **S**cientifique (LAAS-CNRS). Je tiens à remercier Jean Arlat, Directeur de Recherche CNRS qui assure la direction du LAAS-CNRS et Karama Kanoun, Directeur de Recherche CNRS et responsable de l'équipe TSF de m'avoir permis d'accomplir mes travaux dans ce laboratoire.

J'aimerais, tout d'abord, exprimer toute ma gratitude à mes encadrants pour la qualité et la complémentarité de leur encadrement. Je remercie Mohamed Kaâniche et Vincent Nicomette, mes directeurs de thèse, de m'avoir aidé à bien mener ces recherches, pour leur soutien scientifique et leur disponibilité. Je tiens à remercier également Eric Alata pour les multiples remarques très constructives et d'avoir trouvé le temps et la patience de m'aider dans mes travaux.

J'adresse mes sincères remerciements à tous les membres du jury qui m'ont fait l'honneur d'accepter de prendre part à ce jury et surtout de lire et d'expertiser mon travail. Je remercie Carlos Aguilar Melchor, Maître de conférence à l'Université de Limoges, et Christophe Bidan, Professeur à Supelec de Rennes, qui ont accepté d'être les rapporteurs de cette thèse. Je remercie aussi Abdelmalek Benzekri, Professeur à l'Université Paul Sabatier, et Alain Ribault Ingénieur à Kereval, d'avoir accepté de participer au jury.

Je souhaite remercier toutes les personnes avec lesquelles j'ai travaillé, particulièrement Anthony Dessiatnikoff, Yann Bachy et Guillaumme Davy pour leurs contributions. Je tiens à remercier également Yves Crouzet et Sonia De Sousa, pour leur aide précieux, ainsi que les différents services techniques et administratifs du LAAS-CNRS qui m'ont permis de travailler dans d'excellentes conditions.

Merci aussi à tous mes collègues du laboratoire. Je leur exprime ma profonde sympathie et leur souhaite beaucoup de bien et de bonne chance. Toute mon amitié à mes amis qui se reconnaîtront ici, merci pour les moments d'amitié que nous avons partagés. Vous avez toujours été présents pendant les moments les plus difficiles de ma vie professionnelle et personnelle, j'ai eu beaucoup de chance de vous avoir comme amis.

Merci d'avoir cru à ce travail, merci de l'avoir défendu.

Table des matières

Introduction Générale **1**

1 Contexte des travaux **5**

1.1 Technologies Web 6
1.2 Vulnérabilités et attaques web 6
1.2.1 Failles d'injection 9
1.2.2 Cross-Site Scripting (XSS) 12
1.2.3 Violation de gestion d'authentification et de Session 13
1.2.4 Référence directe non sécurisée à un objet 14
1.2.5 Falsification de requête inter-site (CSRF) 14
1.2.6 Mauvaise configuration de sécurité 15
1.2.7 Stockage cryptographique non sécurisé 15
1.2.8 Manque de restriction d'accès URL 16
1.2.9 Protection insuffisante de la couche de transport 16
1.2.10 Redirections et renvois non validés 17
1.3 Contre mesures et moyens de protection 17
1.3.1 Prévention des vulnérabilités 18
1.3.2 Élimination de vulnérabilités : vérification et test 19
1.3.3 Prévention, détection et tolérance aux intrusions 19
1.3.3.1 Pare-feu 19
1.3.3.2 Détection d'intrusions 20
1.3.3.3 Tolérance aux intrusions 23
1.3.4 Évaluation des techniques de protection 23
1.3.4.1 Méthodologie 23
1.3.4.2 Métriques d'efficacité des IDS 25
1.4 Conclusion 27

2 État de l'art 29
2.1 Outils de détection de vulnérabilités web : Scanner web . . . 30
2.1.1 Principe des outils de détection de vulnérabilités . . . 30
2.1.1.1 Approche par reconnaissance de messages d'erreurs . . . 31
2.1.1.2 Approche par étude de similarité des réponses . . . 32
2.1.2 Analyse critique des scanners de vulnérabilités web . . . 33
2.1.3 Évaluation expérimentale . . . 34
2.1.3.1 Injection de vulnérabilités . . . 35
2.1.3.2 Études expérimentales : Exemples . . . 36
2.2 Détection d'intrusions pour des applications Web . . . 37
2.2.1 Les approches par signature . . . 38
2.2.2 Les approches comportementales . . . 39
2.2.2.1 Approche boîte noire . . . 39
2.2.2.2 Approche boîte grise . . . 40
2.2.2.3 Approche boîte blanche . . . 41
2.2.3 Approche hybride . . . 42
2.2.4 Analyse critique . . . 42
2.3 Évaluation de systèmes de détection d'intrusions . . . 43
2.4 Conclusion . . . 45

3 Détection de vulnérabilités par classification automatique des pages html 47
3.1 Aperçu global de l'approche . . . 48
3.2 Algorithme de classification . . . 49
3.2.1 Génération des trois ensembles de requêtes . . . 50
3.2.2 La distance utilisée . . . 52
3.2.3 Regroupement des pages en grappes et choix du seuil . . . 53
3.2.4 Exemple . . . 55
3.3 Génération des requêtes : utilisation des grammaires . . . 57
3.3.1 Requêtes syntaxiquement valides : R_{iv} . . . 58
3.3.2 Requêtes aléatoires : R_a . . . 61
3.3.3 Requêtes syntaxiquement invalides : R_{ii} . . . 61
3.4 Extension à d'autres classes de vulnérabilités . . . 62
3.4.1 Xpath . . . 63
3.4.2 Os Commanding . . . 64
3.4.3 File Include . . . 65
3.5 Expérimentations . . . 65
3.5.1 Notation . . . 66
3.5.2 Expériences avec les applications modifiées . . . 67
3.5.2.1 Présentation des applications modifiées . . . 67

3.5.2.2 Résultats 68
3.5.3 Expériences avec les applications non modifiées 70
3.5.3.1 Présentation des applications non modifiées 70
3.5.3.2 Résultats 71
3.5.4 Synthèse 73
3.6 Conclusion 74

4 Dépendance entre vulnérabilités et génération de scénarios d'attaque 77
4.1 Graphe d'attaque 78
4.2 Présentation de l'approche 79
4.2.1 Définitions 79
4.2.2 Crawling 80
4.2.3 Identification des vulnérabilités 83
4.3 Algorithme de génération de graphe de navigation 84
4.4 Exemple 85
4.4.1 Construction automatique du graphe de navigation 87
4.5 Discussion sur la complexité 90
4.5.1 Pire cas 90
4.5.2 Comptage des chemins 91
4.5.3 Complexité de $crawler$ 91
4.5.4 Complexité de $search_vulns$ 92
4.5.5 Complexité de $main$ 92
4.5.6 Maîtrise de la taille du graphe – similarité des requêtes 96
4.5.7 Expérimentation 96
4.6 Conclusion 99

5 Plateforme d'évaluation 101
5.1 Architecture de la plateforme d'évaluation 102
5.2 Présentation des différents composants de la plateforme 104
5.2.1 Génération de trafic normal 104
5.2.1.1 Interaction humaine et crawler web 104
5.2.1.2 Crawler générateur automatique du trafic normal 105
5.2.2 Trafic d'attaque : Scénarios d'attaque, Ordonnanceur, Lanceur 107
5.2.2.1 Scénario d'attaque 107
5.2.2.2 Ordonnanceur 108
5.2.2.3 Le module d'exécution (*lanceur*) 111
5.2.3 Proxy enregistreur, et rejeu de traces WEB 111
5.2.4 Proxy : simulation de plusieurs sources 113
5.2.5 Application Cible et IDS 115

5.2.6 Traitement et Analyse des résultats . . . 117
5.3 Intégration . . . 119
5.4 Expérimentations . . . 120
5.4.1 Expérimentation avec trafic sain uniquement . . . 121
5.4.1.1 IDS basé sur les invariants . . . 121
5.4.1.2 IDS basé sur les contrats . . . 122
5.4.2 Expérimentation avec trafic malveillant . . . 123
5.4.2.1 Trafic généré par Wasapy . . . 123
5.4.2.2 Attaques manuelles . . . 124
5.4.3 Discussion . . . 125
5.5 Conclusion . . . 125

Conclusion Générale **127**

Bibliographie **131**

Table des figures

1.1 Application Web à architecture 3-tiers 6
1.2 Exemple d'injection SQL 10
1.3 Principales étapes de l'évaluation 24
1.4 Un exemple de courbe ROC 26

2.1 Méthodologie d'injection de vulnérabilités [FONSECA 10] 36

3.1 Algorithme d'extraction de points d'injection et de recherche de vulnérabilités 49
3.2 Distance de classification 53
3.3 Exemple de classification des réponses des requêtes 55
3.4 Dendrogramme de grappes 57

4.1 Principe de l'algorithme 80
4.2 Exemple avec des exemples positifs 82
4.3 Exemple avec des exemples positifs et négatifs 82
4.4 Structure du site Web 87
4.5 Graphe de navigation d'un utilisateur non enregistré 88
4.6 Liste des scénarios utilisés pour tester les vulnérabilités 88
4.7 Résultats de la première itération de l'algorithme 89
4.8 Graphe de vulnérabilités final 89
4.9 Exemple des scénarios d'attaque extraits du graphe final 90
4.10 Arbre de navigation associé au pire cas 91
4.11 Graphe final pour la profondeur 5 97
4.12 Graphe final pour la profondeur 6 97
4.13 Graphe final pour la profondeur 7 98
4.14 Graphe final pour la profondeur 8 98
4.15 Graphes de navigation des 3 itérations de l'application *Riotpix* 98

5.1 Architecture initiale de la plateforme d'évaluation 102

5.2 Architecture détaillée de la plateforme d'évaluation . . . 103
5.3 Algorithme du crawler générateur du trafic normal . . . 106
5.4 Illustration des deux scénarios . . . 108
5.5 Format des scénarios . . . 108
5.6 Exemple de définition d'un scénario . . . 109
5.7 Caractéristiques d'une campagne d'évaluation . . . 109
5.8 Fichier de configuration de l'ordonnanceur . . . 110
5.9 Sortie de l'ordonnanceur . . . 111
5.10 Sortie du proxy enregistreur non mis en forme . . . 113
5.11 Sortie du proxy enregistreur mis en forme . . . 114
5.12 ACL de Squid . . . 115
5.13 Configuration de sortie Squid . . . 115
5.14 Example d'invariant . . . 116
5.15 Code concerné . . . 117
5.16 Exemple de contrat . . . 117
5.17 Présentation des résultats . . . 119
5.18 Interface de la plateforme . . . 121

Liste des tableaux

1.1 Comportements possibles pour un IDS 25

2.1 Exemple de SQL Sheet 44

3.1 Calcul de distance entre les réponses 56
3.2 Résultats de détection de vulnérabilités pour les applications modifiées 69
3.3 Résultats de détection de vulnérabilités pour l'application *Cyphor* 72
3.4 Résultats de détection de vulnérabilités pour l'application *Seagull* 72
3.5 Résultats de détection de vulnérabilités pour l'application *Fttss* 73
3.6 Résultats de détection de vulnérabilités pour l'application *Riotpix* 74
3.7 Résultats de détection de vulnérabilités pour l'application *Pligg* 75
3.8 Résumé des résultats 76

4.1 Tableau de n^{d_m} 95
4.2 Résultats expérimentaux 97

5.1 Présentation des résultats selon le type des requêtes 119
5.2 Trafic normal uniquement avec l'IDS basé sur les invariants 122
5.3 Trafic normal uniquement avec l'IDS basé sur les contrats 122
5.4 Résultats du trafic malveillant pour l'IDS basé sur les invariants 123
5.5 Résultat du Trafic malveillant pour l'IDS basé sur les contrats 124

Introduction Générale

Contexte et problématique

La sécurité des serveurs Web est un problème désormais récurrent. Le nombre de vulnérabilités recensées dans ce type de logiciels s'accroît constamment, tel que décrit notamment dans le document "2011 CWE/SANS Top 25 most dangerous software errors" [1] publié par MITRE. Nous pouvons l'expliquer par plusieurs raisons : la complexité sans cesse croissante des technologies du Web, les délais sans cesse plus courts de mise sur le marché de logiciels, les compétences parfois limitées et le manque de culture en sécurité des développeurs. En conséquence, bon nombre de ces applications contiennent de multiples vulnérabilités qui peuvent être exploitées par des pirates informatiques. Ces attaques peuvent leur permettre, par exemple, d'obtenir des données confidentielles (numéros de cartes de crédit, mots de passe, etc.) qui sont manipulées par l'application, voire même de modifier ou détruire certaines de ces données. La complexité des technologies utilisées aujourd'hui pour réaliser les applications Web (Java, JavaScript, PHP, Ruby, J2E, etc.) fait qu'il est particulièrement difficile 1) d'empêcher l'introduction de vulnérabilité dans ces applications et 2) d'estimer ou de prévoir leur présence. De plus, la sécurisation des réseaux ainsi que l'installation de pare-feux ne fournit pas de protection satisfaisante contre les attaques Web car ces applications sont par définition publiques et accessibles à tous. Il est donc nécessaire d'auditer régulièrement les applications Web pour vérifier la présence de vulnérabilités exploitables et ceci peut être réalisé notamment par des scanners de vulnérabilités Web. Enfin, il est également fondamental de pouvoir détecter les tentatives d'attaques à l'aide de systèmes de détection d'intrusions.

Objectifs et contributions

Pour détecter la présence de vulnérabilités dans les applications Web, les scanners de vulnérabilités envoient des requêtes particulières à l'application cible et analysent les réponses correspondantes

1. http://www.cve.mitre.org/top25

obtenues. En fonction des réponses reçues vis-à-vis de certaines requêtes spécialement formatées, ils sont capables d'en déduire la présence d'une vulnérabilité.
Les systèmes de détection d'intrusions, quant à eux, doivent en principe réagir à toute activité malveillante ciblant l'application Web qu'ils protègent. Pour cela, ils peuvent utiliser différentes méthodologies, comme analyser tous les flux réseaux qui sont destinés à l'application Web, ou analyser les traces de l'application Web ou encore instrumenter le code de l'application Web. S'ils détectent une action malveillante, ils émettent des alertes vers une console de supervision. La source de données utilisée par le système de détection d'intrusions est un élément de choix important pour la sélection et le déploiement d'un tel outil.
Si les systèmes de détection d'intrusions sont aujourd'hui répandus et variés, leur efficacité est encore relativement mal évaluée et il n'existe pas aujourd'hui de méthode de référence permettant cette évaluation. En revanche, un certain nombre d'études montrent que cette efficacité reste souvent limitée (attaques non détectées ou alertes levées en absence d'attaque). La question qui peut se poser est donc : comment évaluer l'efficacité des systèmes de détection d'intrusions et quels moyens peut-on mettre en œuvre pour analyser leur capacité à correctement détecter les attaques ? C'est précisément l'objectif de notre travail en considérant plus particulièrement le cas des systèmes de détection d'intrusions pour des applications Web.

L'objectif de la thèse est donc de développer un cadre conceptuel et des outils associés permettant d'identifier les vulnérabilités des applications Web et d'évaluer l'efficacité de systèmes de détection d'intrusions destinés à faire face aux attaques susceptibles d'exploiter ces vulnérabilités. Il s'agit en particulier de définir des mesures quantitatives de l'efficacité de la protection (comment détecter les attaques et les contrer) et de développer un environnement expérimental pour évaluer ces mesures dans un contexte qui soit représentatif des menaces auxquelles le système sera confronté en opération. Ceci nécessite la définition et la mise en œuvre d'outils de génération de scénarios d'attaques et de profils d'activation du système cible pertinents pour l'évaluation des mesures recherchées. Cette thèse a été effectué dans le cadre d'un projet coopératif financé par l'ANR, le projet DALI : "**D**esign and **A**ssessment of application **L**evel **I**ntrusion detection systems". En particulier, les expérimentations ont porté sur l'évaluation de l'efficacité de systèmes de détection d'intrusions développés par nos partenaires dans le cadre de ce projet. Pour cela, la contribution de notre travail a été double :

- Nous avons conçu et développé une nouvelle méthode pour la détection de vulnérabilités dans des applications Web, utilisant des techniques de clustering. La mise en œuvre de cette méthode, qui est basée sur l'exécution dynamique de l'application cible selon une approche boîte noire, s'est concrétisée par le développement d'un nouveau scanner de vulnérabilités qui présente les caractéristiques suivantes : 1) il permet de détecter de façon précise la présence de vulnérabilités dans une application Web dans la mesure où chaque vulnérabilité est réellement exploitée et la requête permettant cette exploitation est fournie, ce qui permet de vérifier si l'attaque a réellement réussie ou s'il s'agit d'un faux positif ; 2) il est capable de générer automatiquement des scénarios d'attaques complexes, composés de l'exploitation coordonnée de plusieurs vulnérabilités de l'application Web.

– Nous avons aussi illustré l'efficacité de cette méthode par rapport à des outils existants sur plusieurs exemples d'applications vulnérables. Nous avons aussi proposé une plateforme d'évaluation de systèmes de détection d'intrusions pour les applications Web intégrant plusieurs outils qui ont été conçus pour automatiser le plus possible les campagnes d'évaluation. Cette plateforme inclut 1) l'application Web ainsi que le système de détection d'intrusion concernés, 2) des outils permettant de générer du trafic sain et du trafic malveillant à l'application Web (intégrant le nouveau scanner de vulnérabilités mentionné ci-dessus) et 3) des outils permettant d'analyser les alertes des systèmes de détection d'intrusion et d'estimer l'efficacité de ces systèmes.

Plan de la thèse

Le chapitre 1 étudie le contexte de nos travaux : les applications Web et leur sécurité. Dans la première partie, il présente un panorama des vulnérabilités et attaques Web les plus répandues aujourd'hui, en présentant quelques exemples pour chacune d'entre elles. Dans la seconde partie, ce chapitre présente un aperçu des techniques et méthodologies permettant aujourd'hui d'assurer la sécurité des applications Web. Différents moyens de protection et de contre mesures permettant de faire face aux vulnérabilités Web sont discutés.

Le chapitre 2 propose une étude détaillée des différentes techniques utilisées par les scanners de vulnérabilités Web existants, ainsi qu'une analyse critique de ces techniques. Nous proposons également une analyse des approches utilisées par les différents systèmes de détection d'intrusions pour les applications Web, ainsi qu'un aperçu des différents travaux consacrés à l'évaluation de ces systèmes de détection d'intrusions.

Les chapitres 3 et 4 présentent notre première contribution, qui porte sur la proposition d'une nouvelle approche pour la détection de vulnérabilités Web et du scanner de vulnérabilité qui a été développé pour mettre en œuvre cette approche. Le chapitre 3 présente l'approche proposée en considérant dans un premier temps la détection de vulnérabilités élémentaires. Cette approche, basée sur des techniques de clustering, est comparée avec les approches utilisées par d'autres scanners de vulnérabilités à l'aide d'expérimentations ciblant des applications contenant des vulnérabilités variées.

Le chapitre 4 présente un autre aspect important de notre approche : la génération automatique de scénarios d'attaques permettant de mettre évidence les dépendances éventuelles entre les différentes vulnérabilités existant dans une application Web. Ces scénarios sont élaborés à partir de la génération dynamique d'un graphe d'attaque décrivant les différentes possibilités de navigation au travers du site Web en intégrant explicitement les vulnérabilités qui sont découvertes au fur et à mesure de l'exécution de l'application Web selon une approche boîte noire.

Le chapitre 5 décrit l'architecture et la mise en œuvre de notre plateforme d'évaluation. Ce chapitre décrit également des études de cas et des expérimentations qui ont été effectuées à l'aide de cette plateforme. Ces expérimentations ont porté en particulier sur l'évaluation de deux techniques de dé-

tection d'intrusions pour des applications Web développées par nos partenaires du projet DALI. La première technique est basée sur des invariants sur le flot de données qui sont déduits par apprentissage à partir de l'exécution de l'application cible. La deuxième technique est basée sur la spécification de contrats sur le format ou le contenu des requêtes envoyées par le client et qui sont vérifiées en ligne par un proxy entre le client et le serveur Web.

La conclusion générale conclut la thèse et présente les orientations de recherches futures à partir de notre travail.

1

Contexte des travaux

Introduction

Ce chapitre propose un état de l'art concernant les vulnérabilités dont souffrent aujourd'hui les applications Web ainsi que les mécanismes permettant d'assurer la protection des applications Web face aux attaques ciblant ces vulnérabilités.
Ce chapitre est structuré en quatre sections. La section 1.1 donne un aperçu des technologies Web qui ne cessent d'évoluer de nos jours. La section 1.2 décrit les principales vulnérabilités des applications Web et les attaques correspondantes, en focalisant sur les plus répandues. Un aperçu global des mécanismes permettant d'assurer la protection des applications Web est ensuite présenté dans la section 1.3. Enfin, la section 1.4 conclut ce chapitre.

1.1 Technologies Web

Une application Web n'est plus aujourd'hui limitée à un simple serveur Web gérant un ensemble de pages HTML statiques. D'une part, les pages HTML sont aujourd'hui pour la plupart élaborées dynamiquement "à la demande" et d'autre part, l'architecture d'une application Web est désormais relativement complexe, incluant plusieurs machines qui collaborent pour fournir un service. Une application Web peut donc globalement être vue comme réalisant une tâche spécifique (webmail, e-commerce, télébanking, etc...), généralement basée sur une architecture client-serveur 3-tiers, qui comprend un serveur Web, un serveur d'application (parfois confondus), et un serveur de bases de données comme illustré dans la figure 1.1. Elle utilise des technologies relativement complexes qui ne cessent d'évoluer (en particulier avec le passage au Web 2.0), que ce soit du côté du navigateur client (Ajax, JavaScript, Html, RIA- Flash, DOM) ou du côté du serveur (utilisation de serveurs de bases de données et de services Web).

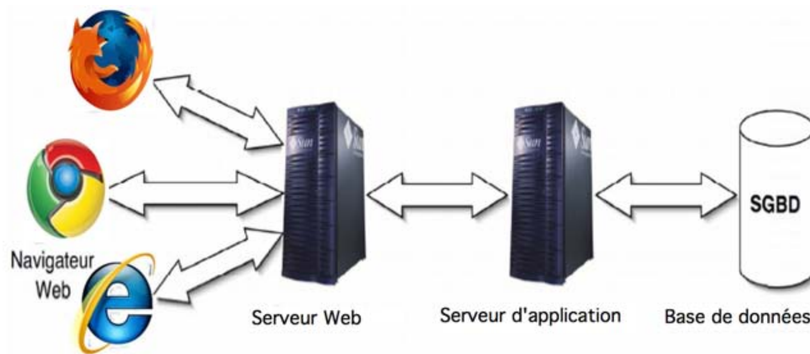

FIGURE 1.1 – Application Web à architecture 3-tiers

La plupart des applications Web implémentent également la notion de session pour garder une trace de l'utilisateur et lui proposer un contenu personnalisé. Ces sessions sont souvent mémorisées dans les navigateurs clients par l'intermédiaire de cookies. La complexité des applications Web d'aujourd'hui ne réside pas tant sur leur contenu que sur la logique de leur programmation : leur développement implique beaucoup de programmation et relativement peu de code HTML statique.

1.2 Vulnérabilités et attaques web

Au même titre qu'une application classique ou qu'un système d'exploitation, les applications Web peuvent présenter des failles de sécurité. Cela est d'autant plus grave que ces applications manipulent parfois des données confidentielles (mots de passe, numéros de cartes bancaires) et qu'elles sont généralement déployées sur Internet et donc exposées au public. Même sur un serveur Web sécurisé s'exécutant sur un système d'exploitation réputé sûr (Apache sur OpenBSD, par exemple), des failles de sécurité peuvent subsister, car elles sont la plupart du temps dues à des fautes de programmation de

l'application elle-même, et non du serveur. Comme nous l'avons précisé dans l'introduction, la complexité croissante des technologies utilisées pour le développement de ces applications, ainsi que le manque de compétence en sécurité des multiples développeurs de ce genre d'application peut en grande partie expliquer les vulnérabilités récurrentes qu'elles présentent.

Définissons à présent plus précisément les termes de vulnérabilité, attaque et intrusion que nous utiliserons tout au long de ce mémoire. Ces définitions sont extraites du projet MAFTIA [MAFTIA ET AL. 03].

- Une *vulnérabilité* est une faute accidentelle ou intentionnelle (avec ou sans volonté de nuire), dans la spécification, la conception ou la configuration du système, ou dans la façon selon laquelle il est utilisé. La vulnérabilité peut être exploitée pour créer une intrusion.
- Une *attaque* est une faute d'interaction malveillante visant à violer une ou plusieurs propriétés de sécurité. C'est une faute externe créée avec l'intention de nuire. Une attaque peut être ou non réalisée par des outils automatiques.
- Une *intrusion* est une faute malveillante interne, mais d'origine externe, résultant d'une attaque qui a réussi à exploiter une vulnérabilité.

Il existe une grande variété de vulnérabilités visant les applications Web. Toutefois certaines sont plus connues et plus dangereuses que d'autres. Plusieurs bases de données répertoriant ces vulnérabilités avec des statistiques indiquant leur importance relative existent. Nous citons par exemple les bases de données de vulnérabilités telles que CVE [1] (**C**ommon **V**ulnerabilities and **E**xposures), NVD [2] (**N**ational **V**ulnerability **D**atabase) ou VUPEN [3] (**V**ulnerability **Pen**etration testing). Ces bases de données répertorient tous types de vulnérabilités, incluant celles ciblant les serveurs et applications Web.
La multiplication des vulnérabilités et des attaques sur des sites web sur Internet ont poussé de nombreuses organisations à poser un regard critique sur la qualité de la sécurité de leurs applications web. Ainsi, plusieurs communautés ont vu le jour, dans le but d'améliorer la sécurisation des applications web. Les travaux dans ce contexte se sont traduits aussi par la proposition de taxonomies et de classifications pour les vulnérabilités et les attaques web les plus répandues. Parmi ces communautés, nous citons OWASP [4] (**O**pen **W**eb **A**pplication **S**ecurity **P**roject) et WASC [5](**W**eb **A**pplication **S**ecurity **C**onsortium).

Les membres du "Web Application Security Consortium" ont créé ce projet pour développer et promouvoir une terminologie standard décrivant les problèmes de sécurité des applications Web et permettant aux développeurs d'applications, experts en sécurité, développeurs de logiciels et les consultants en sécurité, d'utiliser un langage commun pour interagir entre eux. Une première version pour la classification des vulnérabilités composée de six classes a été proposée dans le document "Web

1. http://www.cve.mitre.org
2. http://web.nvd.nist.gov
3. http://www.vupen.com
4. https://www.owasp.org
5. http://www.webappsec.org/

Application Security Consortium : Threat Classification" [6] :

- *Authentification* : il s'agit de vulnérabilités qui concernent les fonctions du site Web permettant d'identifier un utilisateur, un service ou une application.
- *Autorisation* : cette classe regroupe les vulnérabilités liées aux fonctions destinées à vérifier les droits attachés à un utilisateur, un service ou une application.
- *Attaques côté client ("Client-side Attacks")* : il s'agit de vulnérabilités permettant aux attaquants de cibler directement les utilisateurs du site Web en leur délivrant par exemple des contenus illicites tout en faisant croire qu'il s'agit d'informations provenant du site original.
- *Exécution de commandes ("Command Execution")* : cette classe regroupe les vulnérabilités permettant l'exécution à distance de commandes sur le site Web.
- *Divulgation d'information sensible ("Information Disclosure")* : cette classe inclut les vulnérabilités dont l'exploitation permet l'obtention d'informations sur le système (système d'exploitation, version, etc.).
- *Erreurs logiques et bug logiciel ("Logical Attacks")* : les vulnérabilités appartenant à cette classe peuvent conduire à des attaques permettant de détourner la logique d'implémentation de l'application pour réaliser des actions illicites.

Cette classification mélange parfois les faiblesses et les attaques permettant de les exploiter. Une nouvelle version a été développée par la suite pour pallier à ce problème en distinguant ces deux dimensions et en fournissant une liste plus riche des menaces (WASC WSTC v2) [7].
D'un autre côté, l'Open Web Application Security Project (OWASP) a défini dans l'un de ses projets nommé "TOP 10" les dix classes de vulnérabilités Web les plus critiques. L'objectif principal du Top 10 de l'OWASP est d'informer les développeurs, concepteurs, architectes, managers, et les entreprises au sujet des conséquences des faiblesses les plus importantes inhérentes à la sécurité des applications Web. Le Top 10 fournit des techniques de base pour se protéger contre ces vulnérabilités.
La dernière version la plus récente date de 2010. Les vulnérabilités listées sont les suivantes, classées par ordre d'importance :

1. Failles d'injection
2. Cross-Site Scripting (XSS)
3. Violation de gestion d'authentification et de session
4. Référence directe non sécurisée à un objet
5. Falsification de requêtes inter-sites (CSRF)
6. Mauvaise configuration de sécurité
7. Stockage cryptographique non sécurisé
8. Manque de restriction d'accès d'URL

6. `http://projects.webappsec.org/w/page/13246973/ThreatClassificationPreviousVersions`
7. `http://projects.webappsec.org/Threat-Classification`

9. Protection insuffisante de la couche transport
10. Redirections et renvois non validés

Nous les détaillons dans les sous-sections suivantes.

1.2.1 Failles d'injection

Une faille d'injection se produit quand une donnée non fiable est envoyée à un interpréteur en tant qu'élément d'une commande ou d'une requête. Les données hostiles de l'attaquant peuvent duper l'interpréteur afin de l'amener à exécuter des commandes malveillantes ou à accéder à des données non autorisées [HALFOND 06].
On cherche ici à tirer profit d'une entrée utilisateur dont l'utilisation n'est pas suffisamment protégée et assainie. En effet, l'assainissement d'une entrée utilisateur consiste à transformer le contenu, avant le traitement de celui-ci, de telle manière qu'il ne puisse être exécuté comme du code informatique.
On distingue généralement différentes familles d'injection en fonction du protocole visé (SQL, XML, XPATH, LDAP) ou du type d'injection (par exemple injection d'un programme exécutable dans le cas des failles de type “OsCommanding”, ou chargement d'un fichier dans le cas des failles de type “FileUpload”).
Les injections “FileUpload” concernent les sites permettant à leurs utilisateurs de charger des fichiers, tels que des photos par exemple. Il est généralement difficile de vérifier si le fichier que l'utilisateur envoie est malveillant ou pas. Ceci peut permettre à l'utilisateur d'envoyer du code informatique (par exemple PHP) dans un fichier nommé *maphoto.jpg*. Le code informatique pourra alors être exécuté si le site Web inclut cette image sur l'une de ses pages sans le vérifier.
Une injection peut avoir des conséquences graves, puisqu'elle peut mener à la perte ou corruption de données et au déni d'accès au service. Elle peut mener parfois jusqu'à la prise de contrôle total du serveur par l'attaquant. Nous présentons dans la figure 1.2 un exemple illustrant une injection SQL.

La partie gauche de la figure 1.2 décrit un scénario d'utilisation normale et présente les données fournies par l'utilisateur au niveau du navigateur et les requêtes correspondantes au niveau du serveur Apache et du serveur de bases de données MySQL. Si l'application utilise des données non assainies, la requête SQL envoyée s'écrit sous la forme :

```
query="SELECT id, login FROM users WHERE login=' ''.$_GET['login']. '' AND
password=' ''.$_GET['password'])." ' " ;
```

L'injection SQL considérée dans la figure 1.2 consiste à modifier le paramètre `'login'` dans la requête en insérant la tautologie :`' or '1'='1`. Cette tautologie est utilisée lors de l'évaluation de la requête SQL qui, par conséquent, est toujours valide, quel que soit le mot de passe, ce qui permet de contourner le processus d'authentification.

Ce type d'injection peut être classé dans la catégorie des “Blind SQL Injections”. On les utilise dans le cas de scripts à réponse binaire, c'est à dire qui retournent une réponse du type soit vrai, soit

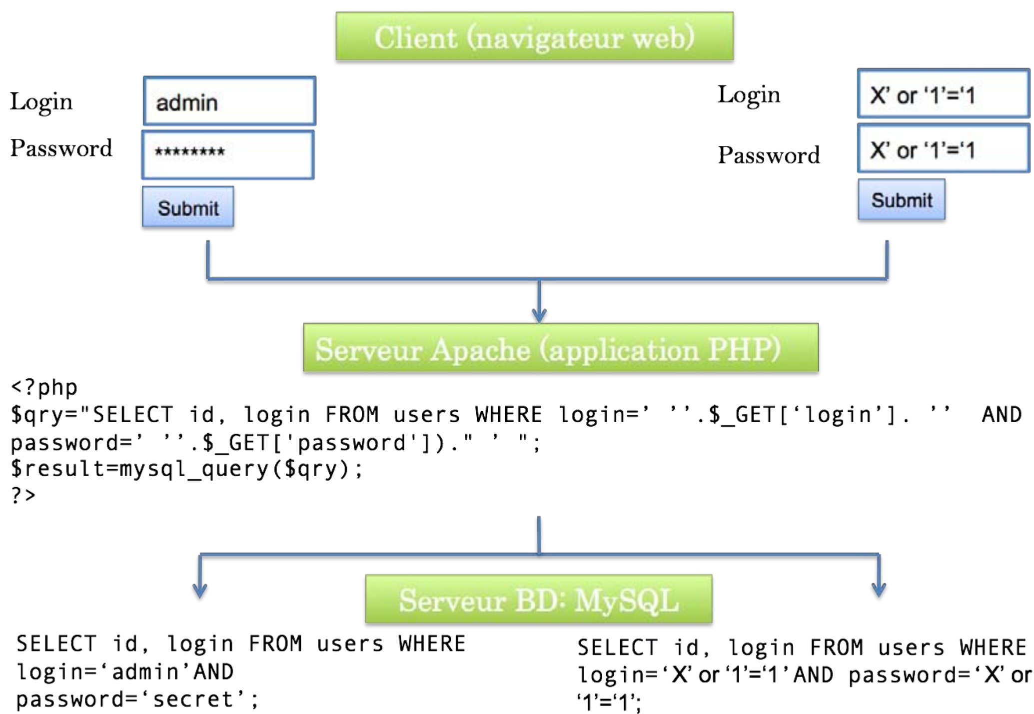

FIGURE 1.2 – Exemple d'injection SQL

faux. C'est le cas par exemple des formulaires d'authentification tel que nous l'avons présenté dans l'exemple précédent. Ce type de script n'affiche pas le résultat d'une injection mais indique simplement s'il y a erreur ou succès. Il existe d'autres catégories d'injection SQL présentées dans [HALFOND 06] qui consiste à insérer, extraire ou modifier des informations de la base de données. Halfon et al. distingue quatre catégories :

– *Injection à travers les entrées utilisateur* : Les attaquants injectent des commandes SQL en fournissant les entrées conçues convenablement pour un objectif particulier. Ces entrées utilisateur proviennent généralement des soumissions de formulaires qui sont envoyés à l'application Web via les requêtes HTTP GET ou POST.
– *Injection à travers les cookies* : Les *cookies* sont des données générées par une application Web puis transmises et stockées par les clients. Ces cookies sont propres à chaque client, le caractérisent du point de vue de l'application Web et sont retournés à l'application Web à chaque fois que le client navigue sur l'application. Ces cookies peuvent être utilisés par exemple pour établir un suivi de session du client (notion de "panier") ou pour l'authentifier. Beaucoup d'applications Web aujourd'hui nécessitent l'activation des cookies par les navigateurs clients pour fonctionner correctement. Ces cookies sont consultables dans les menus de configuration de tous les navigateurs actuels. Pour pouvoir modifier leur contenu ou créer d'autres cookies, il faut en revanche installer des *plugins* particuliers tel que "Cookie Manager+" ou "Tamper Data" de Firefox. Dès lors, un client malveillant installant un tel *plugin* peut altérer le contenu

de ces cookies. Si l'application Web utilise le contenu du cookie pour construire des requêtes SQL, un attaquant peut donc forger une attaque SQL en l'intégrant dans le cookie.
Dans l'exemple de requête HTTP ci-dessous, nous remarquons que les identifiants d'authentification d'un client sont stockés dans deux cookies `guest_id` et `pid`, et peuvent donc constituer la source d'une attaque.

```
GET / HTTP/1.1

Connection: Keep-Alive

Keep-Alive: 300

Accept:*/*

Host: host

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;

rv:1.9.2.16) Gecko/20110319 Firefox/3.6.16 ( .NET CLR 3.5.30729; .NET4.0E)

Cookie: guest_id=v1\%3A1328019064; pid=v1\%3A1328839311134
```

– *Injection à travers les variables du serveur* : Les variables du serveur sont un ensemble de données qui contiennent les en-têtes HTTP et des variables d'environnements. Les applications Web utilisent ces variables du serveur de plusieurs façons. Si ces variables sont enregistrées dans une base de données, sans assainissement, alors elles peuvent être utilisées pour réaliser des injections SQL.
Par exemple, la variable d'environnement `X_FORWARDED_FOR` permet d'identifier l'origine de l'adresse IP du client connecté à l'application Web. Si un client suspecte l'application d'utiliser cette variable pour constuire une requête SQL d'authentification, alors, il peut y insérer une tautologie, et par la même une injection SQL destinée à contourner cette authentification :

```
GET /index.php HTTP/1.1

Host: [host]

X_FORWARDED_FOR :127.0.0.1' or 1=1#
```

– *Injection du second ordre* : Pour cette catégorie d'injection, des attaquants injectent des entrées malveillantes dans la base de données pour déclencher indirectement une SQLI lorsque ces entrées seront utilisées à un moment ultérieur. Les injections de second ordre ne visent pas à provoquer l'attaque directement lorsque l'entrée malveillante atteint la base de données. Pour clarifier, voici un exemple classique d'une attaque par injection de second ordre. Un utilisateur s'enregistre sur un site Web en utilisant un nom d'utilisateur déjà utilisé, tel que `"admin" – "`. L'application échappe correctement l'apostrophe dans l'entrée avant de le stocker dans la base de données, ce qui empêche son effet potentiellement malveillant. A ce

stade, l'utilisateur modifie son mot de passe, une opération qui implique généralement (1) de vérifier que l'utilisateur connaît le mot de passe actuel et (2) changer le mot de passe si la vérification est réussie. Pour ce faire, l'application Web peut construire la commande SQL comme suivante :

```
queryString="UPDATE users SET password='" + newPassword + "' WHERE
userName='" + userName + "' AND password='" + oldPassword + "'"
```

`newPassword` et `oldpassword` sont le nouveau et l'ancien mot de passe respectivement, et `userName` est le nom de l'utilisateur connecté (à savoir, "`admin'-`"). Par conséquent, la chaîne de la requête envoyée à la base de données est (supposer que `newPassword` et `oldPassword` sont "`newpwd`" et "`oldpwd`") :

```
UPDATE users SET password='newpwd' WHERE userName= 'admin'--' AND
password='oldpwd'
```

Parce que "-" est l'opérateur de commentaire SQL, tout ce qui suit est ignoré par la base de données. Par conséquent, le résultat de cette requête est que la base de données change le mot de passe de l'administrateur ("`admin`") à une valeur spécifiée par l'attaquant.

1.2.2 Cross-Site Scripting (XSS)

Les failles XSS ont lieu lorsque l'application génère des pages contenant des données soumises au préalable par un client sans les avoir validées ou assainies. Ces pages, renvoyées aux clients, peuvent donc inclure du code exécutable malveillant qui va s'exécuter dans le navigateur de ces clients. Cette attaque vise donc indirectement les utilisateurs d'un site Web, au travers de l'exploitation d'une vulnérabilité de ce site (d'ou le terme cross-site). On distingue généralement deux types de vulnérabilités XSS :

1. Persistant (*"Stored"*) : Il s'agit ici d'exploiter une vulnérabilité d'un site Web de façon à y stocker de façon permanente du code exécutable malveillant (par l'intermédiaire de l'écriture de messages dans un forum par exemple). Ce code sera par la suite exécuté par tous les utilisateurs qui visiteront ensuite la partie forum du site Web.
2. Non persistant (*"Reflected"*) : Le principe de l'attaque reste le même que dans le cas persistant, à la différence que le code malveillant n'est pas stocké de façon permanente sur le serveur vulnérable. Il peut, par exemple, être inclus dans un paramètre de requête que l'on soumet au site vulnérable. L'attaquant, dans ce cas, doit trouver un moyen de forcer sa victime à invoquer cette URL avec ce paramètre particulier (par exemple, en lui proposant de cliquer sur un lien dans un email).

Le code malveillant téléchargé et exécuté dans le navigateur de la victime peut avoir différents objectifs. L'attaquant peut, par exemple, faire exécuter à sa victime un script dans son navigateur afin de rediriger automatiquement ce client vers une autre URL (qui peut être une copie conforme du site

légitime) afin de voler ses identifiants de session, etc.
Les technologies Web 2.0 notamment, telles que AJAX, rendent plus complexe la détection de vulnérabilité XSS. Dans l'exemple ci-après, l'application réutilise des données soumises dans la requête par l'utilisateur pour élaborer du contenu HTML, sans les assainir au préalable :

```
(String) page += "<input name='creditcard' type='TEXT value='"+
request.getParameter("CC")+"'>";
```

L'attaquant peut alors construire une requête attribuant au champ `'CC'` la valeur suivante :

```
http://example.com/?CC='><script>document.location='http://www.attacker.com/
cgi-bin/cookie.cgi?foo='+document.cookie</script>'.
```

L'exécution de cette requête dans le navigateur de la victime déclenchera l'envoi de l'identifiant de session (sessionID) sur le serveur de l'attaquant, lui permettant ainsi d'opérer un vol de la session en cours.

1.2.3 Violation de gestion d'authentification et de Session

Dans ce type de vulnérabilités, l'attaquant exploite des fuites ou faiblesses des gestionnaires de sessions et d'authentification (ex : comptes exposés, mots de passes, jetons de session) pour s'approprier l'identité d'un autre utilisateur. Les développeurs peuvent être tentés de créer leur propre gestionnaire de sessions et d'authentification, mais il s'agit d'une tâche complexe. Il en résulte souvent des implémentations contenant des faiblesses de sécurité dans des fonctions réalisant la déconnexion de sessions, les fonctions liées au stockage et à la récupération des mots de passe, la gestion des profils, etc. La diversité des implémentations rend la recherche de vulnérabilités complexe. En exploitant ces faiblesses, un attaquant accède au système sous une autre identité et obtient les privilèges correspondant à cette autre identité. Les comptes privilégiés sont visés en priorité, d'où l'impact élevé de cette attaque.
Les deux exemples de scénarios d'attaque suivants illustrent plus l'impact.
Exemple 1 :
Un site sur lequel l'expiration des sessions n'est pas correctement appliquée. Un utilisateur s'y authentifie depuis un ordinateur public, puis ferme le navigateur avant de s'en aller. Une heure après, un attaquant relance le navigateur et accède à la session restée ouverte, grâce au cookie de session stocké dans le navigateur.
Exemple 2 :
Le système de réservation d'une compagnie aérienne réécrit les URLs en y plaçant le jeton de session :

```
http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV
?dest=Hawaii
```

Un utilisateur authentifié souhaite recommander une offre à ses amis. Il leur envoie le lien par e-mail, sans savoir qu'il y inclut l'ID de session. Quand les amis cliquent sur le lien, ils récupèrent sa session, ainsi que ses données de paiement.

1.2.4 Référence directe non sécurisée à un objet

Les applications incluent souvent les identifiants techniques des ressources (ID, clé, etc.) au sein des pages Web générées. Lorsque le contrôle d'accès n'est pas effectué à chaque fois que l'utilisateur demande l'accès à une ressource particulière, il peut en résulter des références directes non sécurisées. Pour cela, l'attaquant ayant accès au système remplace la valeur d'un paramètre identifiant une ressource par une autre valeur existante. Les ressources identifiées par le paramètre vulnérable sont alors compromises. Seuls des identifiants aléatoires (ex : table d'index) empêchent l'attaquant de deviner les valeurs. Nous présentons ci-dessous un exemple de scénario illustrant cette attaque. L'application utilise un paramètre non validé pour construire la requête SQL d'accès aux informations d'un compte :

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt = connection.prepareStatement(query , ? );

pstmt.setString( 1, request.getparameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

Si l'attaquant remplace simplement la valeur du paramètre `acct` dans son navigateur par une autre valeur `notmyacct`, l'application lui retournera les détails d'un compte potentiellement non autorisé :

```
http://example.com/app/accountInfo?acct=notmyacct
```

1.2.5 Falsification de requête inter-site (CSRF)

Une attaque CSRF (Cross Site Request Forgery) force le navigateur d'une victime authentifiée à envoyer à une application Web vulnérable une requête HTTP forgée, comprenant le cookie de session de la victime ainsi que toute autre information automatiquement inclue. En effet, cette attaque tire avantage des applications Web dont les structures des requêtes sont prédictibles. Parce que l'envoi des données de sessions telles que des cookies dans ces applications peut se faire automatiquement, les attaquants peuvent insérer des pages Web malveillantes invisibles qui génèrent des requêtes forgées qui ne sont pas distinguables des légitimes. L'attaquant forge une requête HTTP et amène une victime à la soumettre via une balise d'image (<IMG>), ou de nombreuses autres techniques. Si l'utilisateur est authentifié lors de l'exécution de cette requête, elle va s'exécuter avec succès, permettant ainsi à l'attaquant de réaliser son opération malveillante. Les attaquants peuvent par exemple faire modifier à sa victime une donnée dont elle est propriétaire, ou exécuter une action sous son identité.
Il est à préciser que la présence d'une vulnérabilité XSS rend généralement inopérantes les défenses contre les attaques CSRF.

Exemple de scénario d'attaque CRSF :
L'application permet à un utilisateur de soumettre un changement d'état qui ne contient aucun secret.
Exemple :

```
http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243
```

L'attaquant construit une requête qui transfèrera un montant d'argent de la victime vers son propre compte. Il imbriquera ensuite cette attaque sous une balise d'image ou un IFRAME, pour finalement les placer dans différents sites sous son contrôle.

```
<imgsrc="http://example.com/app/transferFunds?amount=1500&destinationAccount=

AttackerAccount? width="0" height="0"/>
```

Si la victime visite un de ces sites pendant qu'elle est authentifiée à `example.com`, les requêtes forgées vont inclure les informations de la session de l'utilisateur, et la requête s'exécutera avec les privilèges de la victime.

1.2.6 Mauvaise configuration de sécurité

La mauvaise configuration de sécurité peut se manifester à différents niveaux, incluant le système d'exploitation, le serveur Web, le framework applicatif, l'application elle-même. Les développeurs et les administrateurs réseau doivent s'assurer que toutes les couches sont configurées correctement. En effet, l'attaquant peut accéder facilement à des comptes par défaut, pages non utilisées, vulnérabilités non corrigées, fichiers et répertoires mal protégés, etc., afin d'obtenir des accès non autorisés, des informations confidentielles, voire même, de pouvoir modifier des données ou prendre le contrôle de la machine.
Exemple 1 :
La console de gestion applicative d'une application Web est automatiquement installée et non désactivée. Les comptes par défaut sont inchangés. L'attaquant découvre la console, utilise le compte par défaut, et prend le contrôle.
Exemple 2 :
L'affichage du contenu des répertoires d'un serveur Web est possible, suite à une mauvaise configuration du serveur. L'attaquant le découvre et télécharge les classes java compilées, qu'il décompile pour obtenir le code source qu'il peut ensuite analyser pour y chercher des vulnérabilités.

1.2.7 Stockage cryptographique non sécurisé

Cette vulnérabilité englobe plusieurs scénarios. Elle peut signifier que des données sensibles qui devraient être en principe chiffrées ne le sont pas ou le sont trop faiblement. Elle peut signifier également que les clés de chiffrement ne sont pas suffisamment protégées et peuvent être trouvées par un attaquant. Elle peut enfin signifier que les opérations qui déchiffrent des données chiffrées sont mal protégées.

Exemple :
Une application chiffre des cartes de crédit dans une base de données. La BD est configurée pour automatiquement déchiffrer les requêtes sur la colonne des cartes, permettant une faille d'injection SQL afin de récupérer tous les numéros en clair. Le système devrait avoir été configuré afin de ne permettre qu'aux applications internes de les déchiffrer, et non l'application Web publique.

1.2.8 Manque de restriction d'accès URL

Pour protéger l'accès à certaines pages et fonctions sensibles, beaucoup d'applications Web cachent les liens URL permettant d'y accéder. Cependant, ce moyen de protection n'est pas suffisamment efficace, car un attaquant motivé, compétent ou très chanceux peut trouver ces liens et accéder à des fonctions non autorisées (les fonctions d'administration sont notamment les cibles clés de ce type d'attaque). Par conséquent, le principe de "sécurité par obscurité" n'est suffisant dans ce contexte. Des mécanismes de contrôle d'accès plus rigoureux sont nécessaires pour vérifier les droits avant qu'une requête d'accès à des fonctions sensibles soit autorisée.
Exemple de scénario d'attaque :
L'attaquant force simplement la navigation d'URLs cibles. Considérons les URLs suivantes censées toutes deux exiger une authentification. Des droits Administrateur sont également requis pour accéder à la page `'admin_getappInfo'`.

```
http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo
```

Si l'attaquant n'est pas authentifié et que l'accès à l'une des pages est accordé, alors un accès non autorisé est permis. Si un utilisateur non administrateur authentifié est autorisé à accéder à la page `'admin_getappInfo'`, il existe une faille pouvant conduire l'attaquant à accéder à d'autres pages non protégées réservées aux administrateurs. De telles failles sont fréquemment introduites lorsque des liens et des boutons sont simplement masqués aux utilisateurs non autorisés et que l'application ne protège pas les pages ciblées.

1.2.9 Protection insuffisante de la couche de transport

Cette vulnérabilité est liée au fait que les applications Web ne protègent pas ou protègent mal le trafic réseau. Si par exemple, l'utilisation de SSL/TLS n'est faite que durant la phase d'authentification, des données et des identifiants de session peuvent être exposés dans les flux réseaux applicatifs. Des certificats expirés ou mal configurés peuvent également être à l'origine de ce type de vulnérabilité. Une mauvaise configuration de SSL peut notamment faciliter des attaques de type "phishing", "man in the middle", etc.
De telles failles exposent des données utilisateurs et peuvent conduire à leur usurpation.
Exemple 1 :
Un site n'utilise pas SSL pour les pages nécessitant une authentification. L'attaquant écoute simplement

le trafic réseau et capture un mot de passe ou un cookie de session d'un utilisateur authentifié. L'attaquant peut alors réutiliser ce mot de passe ou ce cookie afin d'obtenir la session de l'utilisateur.
Exemple 2 :
Un site utilise simplement la norme ODBC/JDBC pour la connexion à la base de données, sans réaliser que tout le trafic est en clair.

1.2.10 Redirections et renvois non validés

Les applications utilisent fréquemment les redirections et les renvois pour rediriger les utilisateurs vers d'autres pages. Parfois la page cible est spécifiée dans un paramètre non validé, permettant à un attaquant de choisir la page de redirection. Un attaquant peut donc essayer de faire en sorte (en envoyant un email par exemple) qu'une victime clique sur un lien pointant sur un site de confiance, mais contenant un paramètre de redirection pointant vers un site malveillant, copie conforme du site de confiance. La victime sera probablement encline à cliquer sur le lien inclus dans l'email, puisqu'il pointe vers un site valide. De telles redirections peuvent permettre par exemple d'installer des logiciels malveillants, de capturer des informations confidentielles de l'utilisateur ou de contourner les contrôles d'accès.
Exemple 1 :
L'application Web sur le site `www.example.com` possède une page nommée `"redirect.jsp"` prenant en compte un seul paramètre nommé "url". L'attaquant fabrique une URL malveillante redirigeant les utilisateurs vers un site réalisant de l'hameçonnage (phishing).

```
http://www.example.com/redirect.jsp?url=evil.com
```

Exemple 2 :
L'application Web sur le site `www.example.com` utilise des renvois pour acheminer des requêtes entre différentes parties du site. Pour faciliter cela, certaines pages utilisent un paramètre indiquant vers quelle page l'utilisateur doit être redirigé en cas de succès de la transaction. Dans ce cas, l'attaquant fabrique une URL satisfaisant les contrôles d'accès de l'application et le dirigeant ensuite vers une fonction administrateur à laquelle il ne devrait pas avoir accès.

```
http://www.example.com/forward.jsp?fwd=admin.jsp
```

1.3 Contre mesures et moyens de protection

Différentes méthodes et techniques peuvent être mises en œuvre par les développeurs et les administrateurs en charge de la sécurité informatique pour faire face aux diverses menaces qui visent les applications Web. Dans la suite nous présentons quelques exemples de moyens permettant d'assurer la sécurité, en considérant les objectifs suivants :

– *Prévention de vulnérabilités* : empêcher l'introduction de vulnérabilités par l'application de méthodes de développement rigoureuses.

– *Élimination des vulnérabilités* : identifier les vulnérabilités et les éliminer en utilisant des techniques de vérification et de test.
– *Prévention, détection et tolérance des intrusions* : protéger le système pendant l'exploitation vis-à-vis des attaques et des intrusions en mettant en œuvre des barrières de défense (pare-feux, systèmes de détection, prévention ou de tolérance aux intrusions) permettant à l'application de fournir un service correct en dépit des attaques.
– *Évaluation* : estimer par évaluation l'impact des vulnérabilités et des attaques ainsi que l'efficacité des mécanismes de protection mis en œuvre.

1.3.1 Prévention des vulnérabilités

La prévention de vulnérabilités s'appuie généralement sur l'utilisation de processus et de règles de développement et d'exploitation rigoureux afin de réduire les risques d'introduction de vulnérabilités durant les différentes phases du cycle de vie de l'application.
Que l'on soit développeur débutant ou déjà expérimenté dans la sécurité des applications Web, il peut s'avérer difficile de réaliser une nouvelle application Web, ou bien d'en sécuriser une existante, en maîtrisant tous les risques sans appliquer des processus et des outils appropriés. Pour cette raison, il est fondamental de prendre en compte les aspects sécurité tout au long de la phase de développement de l'application, en particulier dès les premières phases.
Tout d'abord, il faut respecter les exigences de sécurité des applications c'est-à-dire préciser ce que sécurité veut dire, car il est beaucoup plus rentable de développer une application en la sécurisant dès sa conception plutôt que combler ses faiblesses a posteriori. Par exemple Owasp recommande d'utiliser son guide "Application Security Verification Standard" (ASVS) [8] pour définir les exigences de sécurité des applications ou encore le modèle "OWASP Software Assurance Maturity Model" (SAMM) [9], qui permet aux organisations de définir et de mettre en œuvre une stratégie de sécurité adaptée à leurs risques spécifiques, améliorant ainsi leur processus de réalisation d'applications sécurisées.
Par la suite, il est nécessaire de respecter, au cours de l'écriture du code de l'application, les contrôles de sécurité standard. Par exemple, le projet Owasp "Enterprise Security API" (ESAPI) propose notamment un Template pour la sécurisation des interfaces des applications Web. L'ensemble des contrôles standard qu'il contient peut faciliter grandement le développement d'applications en Java, .NET, PHP, Classic ASP et Python. Halfond et al. énumèrent également dans leur travail [HALFOND 06] les différentes techniques de prévention que les développeurs doivent prendre en compte et ils s'intéressent plus particulièrement aux vulnérabilités d'injection SQL.
L'application de ces règles peut contribuer grandement à l'amélioration de la sécurité des applications Web. Cependant, face à la complexité des applications et des technologies utilisées, ces mesures ne sont pas suffisantes pour garantir un développement exempt d'erreurs et doivent être complétées par d'autres moyens. Nous les abordons dans les sections suivantes.

8. https://www.owasp.org/index.php/ASVS
9. https://www.owasp.org/index.php/SAMM

1.3.2 Élimination de vulnérabilités : vérification et test

L'élimination des fautes vise à identifier les vulnérabilités résiduelles introduites lors de développement de l'application et à les corriger. Elle s'appuie sur différentes techniques de vérification complémentaires qui peuvent être utilisées avec ou sans l'activation de l'application. La vérification sans activation réelle correspond à la vérification statique, qui peut concerner les différentes phases du développement (spécification, conception, codage). Dans l'autre cas, il s'agit de vérification dynamique s'appuyant généralement sur des techniques de test, en particulier des tests de pénétration.
Il est nécessaire d'appliquer les deux techniques de vérification aussi souvent que nécessaire, celles-ci étant complémentaires. De plus, en raison de la complexité croissante des applications Web, le développement d'outils d'aide à la vérification devient indispensable.

Plusieurs travaux récents ont été effectués sur des techniques d'analyse statique de code pour l'identification de vulnérabilités dans des applications Web (ASP, PHP, Java, ...). On peut citer par exemple ([Wassermann et Su 07],[Lam et al. 08],[Huang et al. 04],[Livshits et Lam 05], [Minamide 05],[Jovanovic et al. 06], [Jovanovic et al. 10],[Xie et Aiken 06]). L'avantage d'une telle méthode est qu'elle permet d'éliminer les fautes dans le logiciel avant que celui-ci ne soit déployé. Cependant cette méthode n'est pas adaptée aux cas où le code n'est pas disponible.

Concernant la vérification dynamique, outre des tests de pénétration classiques qui s'appuient généralement sur l'expertise et le savoir faire des évaluateurs en charge de ces tests, plusieurs travaux ont été menés sur le développement de techniques et d'outils permettant de faciliter la recherche de vulnérabilités en adoptant une approche boîte noire (c'est-à-dire sans avoir accès au code source). Les outils correspondants sont en général nommés scanners de vulnérabilités Web [Stefan et al. 06], ou outils de détection de vulnérabilités. L'effort a porté principalement sur l'identification des principales classes de vulnérabilités ciblant les applications Web (injections SQL, XSS, injection de commandes, etc.) que nous avons décrites dans la section 1.2 [Martin et Lam 08],[Kosuga et al. 07],[Halfond 06],[Kiezun et al. 09]. Nous détaillons le principe de fonctionnement de ces scanners dans la section 2.1.

1.3.3 Prévention, détection et tolérance aux intrusions

Dans ce paragraphe nous décrivons les principes de bases des techniques visant à prévenir ou détecter des requêtes malveillantes ciblant les applications Web par l'utilisation de pare-feu ou de systèmes de détection d'intrusions plus sophistiqués.

1.3.3.1 Pare-feu

Les pare-feux ("Firewall" en Anglais) sont généralement mis en œuvre pour bloquer et restreindre certains accès et implémenter les règles de la politique de sécurité [Cheswik et Bellovin 94]. Ils comportent essentiellement une fonction de filtrage, permettant par exemple de ne laisser passer que les paquets provenant de certaines adresses autorisées (adresse IP + numéro de port) et à destination de

certaines adresses autorisées. Ces composants peuvent aussi être utilisés pour tracer voire détecter des comportements précurseurs d'attaques.

Traditionnellement, les pare-feux sont souvent utilisés pour contrôler le trafic rentrant et sortant au niveau d'une machine, d'un réseau local ou du réseau d'une entreprise. Pour les applications Web, des pare-feux applicatifs dont le rôle est d'effectuer des contrôles sur les requêtes échangées entre les clients et l'application Web ont aussi été explorés récemment. Ils agissent en général comme mandataires ("reverse proxy") [BARNETT ET B.RECTANUS 09] et sont appelés WAF (Web Application Firewall). Le mode reverse-proxy consiste à faire apparaître le WAF comme le serveur Web du point de vue du client. Ceci permet ainsi de masquer l'infrastructure hébergeant l'application Web du point de vue réseau (adresse IP inaccessible) ainsi que du point de vue applicatif (en cachant certaines informations pouvant permettre d'identifier la nature du serveur Web). On peut citer comme exemples *ModSecurity* [10]ou *BeeWare* [11].

ModSecurity permet de filtrer les requêtes entrant sur un serveur HTTP Apache. Il se présente sous la forme d'un module apache, qui analyse les requêtes reçues grâce à l'emploi d'une base des règles de requêtes considérées comme non souhaitées. Ces règles sont codées sous forme d'expressions régulières. *ModSecurity* peut également être utilisé en tant que reverse proxy ce qui permet notamment de faire du virtual patching, c'est-à-dire d'empêcher une vulnérabilité d'être exploitée dans une application Web le temps que le correctif soit disponible pour cette application.

Ces outils assurent une protection côté serveur, construite à partir d'une base de signatures d'attaques. Les requêtes envoyées par les utilisateurs sont vérifiées en utilisant cette base d'attaques connues pour détecter d'éventuelles tentatives d'intrusions. Ces règles sont principalement destinées à constituer une barrière contre des tentatives d'exploitation de failles applicatives notamment les injections de code. Elles peuvent être utilisées également pour prévenir des attaques de déni de service par limitation du nombre de sessions applicatives. Ces outils sont donc capables de détecter et bloquer la plupart des attaques connues ou simples. Mais il sont loin d'être suffisants : ils sont en effet souvent contournés via des techniques d'encodage, l'utilisation d'attaques non connues ou l'emploi de vecteurs d'attaques généralement non surveillés par ces outils de protection (cookies, en-tête http, etc).

Les pare-feux applicatifs décrits ci-dessus intègrent des fonctionnalités plus sophistiquées que celles qu'on retrouve dans des pare-feux traditionnels et s'apparentent déjà à des outils de détection d'intrusions.

1.3.3.2 Détection d'intrusions

La détection d'intrusions vise à identifier les actions et les tentatives qui essaient de contourner la politique de sécurité pour compromettre la confidentialité, l'intégrité ou la disponibilité d'une ressource,

10. `http://www.modsecurity.org/`
11. `http://www.bee-ware.net/fr/`

et à lever des alertes en cas de détection. Elle peut être effectuée manuellement ou automatiquement. Dans le processus de détection d'intrusions manuelle, un analyste humain procède à l'examen de fichiers de logs à la recherche de tout signe suspect pouvant indiquer une intrusion. Un système qui effectue une détection d'intrusion automatisée est appelé système de détection d'intrusion (noté IDS pour "Intrusion Detection System"). Les actions typiques qu'il peut entreprendre sont par exemple enregistrer l'information pertinente dans un fichier ou une base de données et générer une alerte.

Déterminer quelle est réellement l'intrusion détectée et entreprendre certaines actions pour y mettre fin ou l'empêcher de se reproduire, ne font généralement pas partie du domaine de la détection d'intrusions. Cependant, quelques formes de réaction automatique peuvent être implémentées par l'interaction de l'IDS et de systèmes de contrôle d'accès tels que les pare-feux. Il s'agit dans ce cas d'un IDS défensif (aussi nommé "Intrusion Prevention System" ou IPS). Il existe principalement deux types d'IDS [DESWARTE 03A] :

- les *IDS sur réseau* ("Network based IDS"), qui observent les paquets circulant sur le réseau (*Snort* [12] [CASWELL ET AL. 03] et *NetRanger* [13] en sont des exemples)
- les *IDS sur hôte* ("Host based IDS"), qui observent le comportement du système, en particulier les appels systèmes, ou qui analysent les informations d'audit enregistrées dans des fichiers de log (*NetIQ Security Manager* [14]en est un exemple).

Les principes fondateurs des systèmes de détection d'intrusions ont été proposés aux Etats-Unis au début des années 1980 [ANDERSON 80],[DENNING 87].
Deux types d'approches permettant de détecter des actions malveillantes sont généralement distingués [MCHUGH 00],[PROCTOR 01],[DEBAR ET AL. 00] :

- *L'approche par scénario* (appelée aussi approche basée sur la connaissance [DEBAR ET AL. 00], détection d'abus ou bien détection d'attaques [DESWARTE 03A]), qui s'appuie sur la comparaison du comportement observé avec une référence correspondant à des signatures ou des scénarios d'attaques connus (motifs définis, caractéristiques explicites). Si une telle signature est identifiée au sein de l'information recueillie, l'activité correspondante est considérée comme une attaque (avec différents niveaux de sévérité). On peut citer comme exemple l'outil STAT (State Transition Analysis Toolkit) [ILGUN 95].
- *L'approche comportementale* (appelée aussi détection d'anomalies), qui consiste à comparer le comportement observé à une référence de comportement normal (c'est-à-dire en l'absence d'intrusion) et à émettre une alerte quand une déviation entre les deux comportements est détectée.

Notons que si l'approche par scénario est efficace pour reconnaitre les attaques connues, elle est malheureusement mise en défaut lorsque de nouvelles attaques interviennent et certains logiciels

12. http://www.snort.org/
13. http://www.cnetfrance.fr/telecharger/en/netranger-39069179s.htm
14. http://www.netiq.com/products/sm/default.asp

profitent de cette faille pour passer outre ces systèmes de détection [FOGLA ET LEE 06]. En effet, l'inconvénient majeur de cette approche est qu'elle nécessite d'avoir une connaissance préalable de la nature des intrusions possibles. Les IDS basés sur une approche par scénario ne seront pas en mesure de détecter de nouvelles intrusions, puisque les signatures caractéristiques ne seront pas présentes dans la base de données. L'efficacité de cette technique repose donc totalement sur la capacité à entretenir une base de données des signatures à jour. Par ailleurs, ces IDS peuvent aussi générer des faux positifs car l'écriture de signatures d'attaque réellement discriminante est difficile.

Dans le cadre de l'approche comportementale, le modèle de référence décrivant le comportement normal peut être obtenu à partir des spécifications du système, ou bien par apprentissage à partir de l'observation. Dans ce denier cas, différentes techniques d'apprentissage supervisé ou non ont été proposées. Par exemple, dans [GIACINTO ET AL. 06] et [MUNZ ET AL. 07], les auteurs appliquent des algorithmes de "Clustering" pour déterminer les différentes classes de modèles. Dans cette méthode, il s'agit de construire des modèles correspondant à tous les usages normaux. Ensuite, les nouveaux comportements sont comparés à ces modèles et ceux qui s'en éloignent trop sont considérés comme atypiques.
L'avantage de cette approche est qu'elle permet de détecter les nouvelles attaques sans intervention additionnelle sur l'IDS. L'enjeu est de créer un modèle de référence suffisamment complet et précis pour détecter les déviations et réduire le nombre de faux positifs.
Par rapport à l'approche par scénario, les IDS basés sur la détection d'anomalies ont l'avantage d'être moins dépendants des attaques antérieures. En revanche, ils ne sont pas capables de prendre en compte les comportements normaux non prévus ou les évolutions des applications ou des systèmes et peuvent ainsi engendrer un grand nombre de fausses alarmes.
Cette discussion montre que les deux approches de détection d'intrusions, par scénario et comportementale, ont des avantages et des inconvénients, et s'avèrent complémentaires. Il en découle aussi l'importance d'évaluer l'efficacité de ces IDS vis-à-vis de leur capacité à détecter des attaques. Cet aspect est discuté dans la section 1.3.4.

Nous présentons également dans la section 2.2 un état de l'art sur la détection d'intrusions dans le contexte des applications Web.

Pour résoudre ce problème, de nouvelles approches ont été proposées et tentent de maintenir les signatures de manière automatique [ESPOSITO ET AL. 05], [LI 05] et [YEUNG ET DING 03]. Des techniques basées sur la logique floue, les algorithmes génétiques ou les réseaux de neurones sont également utilisées dans [DESASILVA ET AL. 07], [SANIEE ET AL. 07] et [NEWSOME ET AL. 05], mais à cause de la structure complexe des requêtes, elles sont généralement difficiles à mettre en œuvre et sont souvent pénalisées par des temps de réponses prohibitifs. Aussi, l'évolution des signatures se résume très souvent à l'intervention d'un expert du domaine et les changements sont lents et couteux. Il est donc nécessaire de proposer des systèmes qui soient capables d'apprendre automatiquement les signatures des requêtes valides mais qui soient aussi capables de les maintenir afin de réduire le nombre

de fausses alarmes. Dans [GUPTA ET AL. 08], les auteurs proposent un IDS basé sur la méthode de détection d'anomalies. Leur objectif est de se focaliser plus particulièrement sur la détection de nouvelles attaques et sur les modifications apportées aux anciennes attaques qui ne peuvent pas être détectées par les IDS actuels.

1.3.3.3 Tolérance aux intrusions

Un système tolérant aux intrusions est un système capable de s'auto-diagnostiquer, se réparer et se reconfigurer tout en continuant à fournir un service acceptable aux utilisateurs légitimes pendant les attaques[DESWARTE ET AL. 91],[DESWARTE ET POWELL 06]. La tolérance aux intrusions peut être appliquée avec différentes techniques de sécurité. Parmi les techniques classique de sécurité, nous pouvons citer le chiffrement, la réplication, et le brouillage des données.
Tel que c'est présenté dans [DESWARTE 03B],la technique de fragmentation, redondance et dissémination a pour but une approche globale de la tolérance aux fautes accidentelles et intentionnelles, pour le traitement, le stockage et la transmission d'informations confidentielles. Pour atteindre ce but elle découpe l'information en fragments, duplique ces fragments et les disperse sur différents sites. Les conditions que doit remplir un pirate pour reconstituer la donnée sont plus nombreuses, rendant sa tâche nettement plus difficile.

1.3.4 Évaluation des techniques de protection

Les différentes techniques décrites dans les sections précédentes jouent un rôle très important pour assurer la sécurité des applications Web. La sécurité de ces applications dépend du niveau de la menace ciblant ces applications et de l'efficacité des contre-mesures mises en œuvre pour faire face à ces menaces. Le niveau de la menace dépend du nombre de vulnérabilités résiduelles dans ces applications et de la fréquence d'occurrence de tentatives d'attaques ayant pour objectif d'exploiter ces vulnérabilités. Le succès ou non d'une attaque dépendra de l'efficacité des mécanismes de protection. Nous focalisons ici sur les méthodes et les mesures permettant de quantifier l'efficacité des techniques de protection, plus particulièrement des IDS, ou bien des outils destinés à révéler la présence de vulnérabilités tels que les scanners de vulnérabilités décrits dans la section 1.3.2. Dans la suite, nous présentons la méthodologie généralement utilisée pour mener ces évaluations et les principales métriques considérées.

1.3.4.1 Méthodologie

L'évaluation d'un système de détection d'intrusions nécessite généralement une phase d'expérimentation pour s'assurer qu'il s'adapte correctement à l'environnement qu'il protège. Afin d'obtenir des évaluations représentatives, fiables et facilement réalisables en pratique, il est nécessaire de définir des protocoles expérimentaux rigoureux et automatisés.

Historiquement, les premières approches étaient basées sur des expérimentations informelles par des équipes d'experts en sécurité, appelées "red teams", dont l'objectif était d'essayer de contourner les

mécanismes de protection pour compromettre la sécurité des systèmes considérés. Cependant, le besoin s'est ressenti de développer des approches plus structurées basées sur un protocole expérimental systématique permettant de caractériser de façon rigoureuse l'efficacité d'un IDS et éventuellement de comparer différents systèmes de détection d'intrusion avec une approche de type étalonnage ("Benchmarking"). La Figure 1.3 présente un schéma typique d'une évaluation d'un IDS. On peut distinguer trois principales étapes dans l'approche expérimentale conduisant à l'évaluation des IDS :

1. *Préparation de la campagne d'évaluation* : Cette étape comprend la spécification des objectifs de l'évaluation et la définition des données d'entrée qui seront utilisées pendant l'expérimentation pour activer le système cible et l'IDS. Deux types de données sont nécessaires pour ce type d'évaluation : des données reflétant l'utilisation du système dans des conditions normales appelées aussi "trafic sain", et des données, appelées "trafic malveillant", correspondant à des scénarios d'attaque représentatifs des menaces auxquelles le système cible sera confronté. Cette étape inclut également la définition des campagnes d'évaluations et de l'environnement expérimental permettant le lancement de ces campagnes, la récupération des résultats et enfin l'analyse de ces résultats.
2. *Expérimentation* : Cette étape consiste à lancer les campagnes d'évaluation et à récupérer les relevés des expérimentations (incluant les alertes levées par l'IDS).
3. *Traitement des relevés de l'expérimentation* : Le but de cette étape est de traiter les relevés issus des expérimentations afin d'évaluer différentes métriques permettant de caractériser l'efficacité des mécanismes de protection mis en œuvre au niveau du système cible. Des exemples de métriques sont décrits dans la section 1.3.4.2

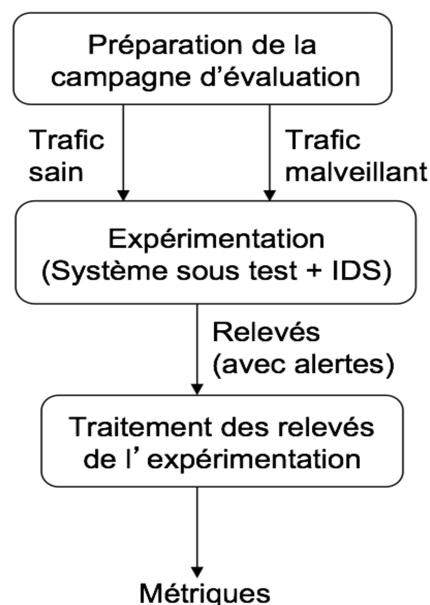

FIGURE 1.3 – Principales étapes de l'évaluation

1.3.4.2 Métriques d'efficacité des IDS

L'efficacité d'un IDS peut se mesurer notamment par le taux de faux positifs et faux négatifs. Le tableau 1.1 présente les différentes situations qui peuvent se présenter en considérant le cas où une attaque est en cours et en absence d'attaque.

		Réaction de l'IDS	
		Pas d'alerte	Alerte
Expérimentation	Pas d'attaque	Vrai négatif	Faux positif
	Attaque	Faux négatif	Vrai positif

TABLE 1.1 – Comportements possibles pour un IDS

On distingue les cas suivants :

- **Un vrai négatif** est une activité normale correctement considérée comme telle par l'IDS.
- **Un faux positif** appelé aussi fausse alerte, est une activité normale considérée à tort comme une attaque par l'IDS.
- **Un faux négatif** est une attaque non détectée, considérée donc à tort comme une activité normale par l'IDS.
- **Un vrai positif** est une attaque correctement considérée comme telle par l'IDS.

Parmi ces quatre comportements, les vrais négatifs et vrais positifs correspondent aux comportements souhaités. Toutefois un IDS est généralement imparfait et conduit à l'apparition des deux autres comportements non désirés. Les différents IDS souffrent généralement d'imperfections donnant lieu à l'apparition de ces comportements non désirés, mais selon des axes différents suivant les méthodes de détection qu'ils utilisent. En effet, l'efficacité d'un IDS dépend de ces facteurs. Nous citons ici les paramètres quantitatifs permettant d'analyser la qualité de détection. Il est d'abord question du taux de détection et du taux de fausses alertes.

Le taux de détection également appelé taux de vrais positifs, est le pourcentage des intrusions correctement détectées par rapport aux nombre total d'intrusions sur lesquelles l'IDS est évalué. Il fournit un indice global de la qualité de détection. Le taux de fausses alertes représente la proportion des activités normales pour lesquelles des alertes d'intrusions ont été levées. Ainsi un dysfonctionnement de l'IDS peut correspondre à une fausse alerte ou une intrusion non détectée. Un taux de faux positifs trop élevé nuit à la crédibilité de l'IDS et un taux de faux négatifs élevé rend, quant à lui l'IDS inefficace dans la mesure où la détection de la majorité des intrusions est ratée.
Bien que les faux négatifs représentent effectivement le premier des comportements indésirables pour un IDS, les faux positifs sont importants également : ils peuvent conduire à une réelle perte de confiance

dans les capacités de détection de l'IDS de la part des administrateurs. C'est même une des voies d'attaque envisageables contre un système équipé d'un IDS : générer un nombre suffisamment important de fausses alertes pour réduire l'attention des administrateurs et dissimuler une attaque réelle. Dans la pratique, les faux positifs dus à l'environnement de l'IDS ou à des signatures d'attaque un peu trop affirmatives sont souvent nombreux ; et ceci nécessite généralement un re-paramétrage de l'IDS pour faciliter son exploitation, au prix de l'introduction éventuelle de faux négatifs. La gestion des faux positifs est le premier problème auquel sont confrontés les administrateurs d'un IDS, et il est généralement de taille.

Une des techniques utilisées pour l'évaluation des IDS est la courbe ROC "Receiver Operating Characteristic" [FAWCETT 06] [BRADLEY 97]. Cette courbe traduit graphiquement la corrélation entre le taux de détection et le taux de faux positifs sous-jacent. Un exemple est présenté dans la figure 1.4 pour illustration. L'axe des abscisses représente le taux de faux positifs et l'axe des ordonnées le taux de détection.

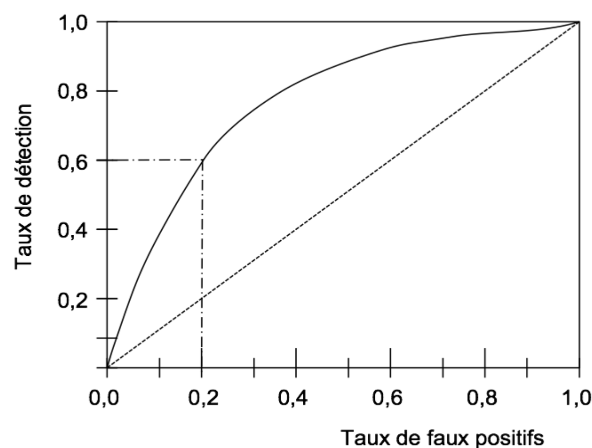

FIGURE 1.4 – Un exemple de courbe ROC

Souvent, les algorithmes de détection sont élaborés de telle façon à pouvoir fixer un seuil pour le taux de fausses alertes à ne pas dépasser. Ainsi en faisant varier ce dernier paramètre, nous obtenons à chaque fois le taux de détection correspondant. Les résultats permettent de tracer une courbe de l'évolution du taux de détection en fonction du taux de fausses alertes générées. Cette courbe peut aussi servir une fois tracée à localiser le taux de détection qui correspond à un taux de fausses alertes choisi. D'autres métriques et analyses sont plus au moins utilisées selon les besoins et les contextes tels que :

- Taux de couverture : il s'agit de la proportion d'attaques différentes qu'un IDS peut détecter correctement par rapport au nombre total d'attaques différentes existantes.
- Capacité de détecter de nouvelles attaques.
- Capacité de détecter les attaques avec leur signature.
- Capacité de détecter le succès d'une attaque.

- Débit réseau réellement supporté.
- Degré d'imperméabilité et résistance aux attaques ayant pour cible l'IDS lui-même.
- Capacité de fonctionner continuellement et avec un minimum d'intervention humaine.
- Tolérance aux fautes : capacité de recouvrement après un bug ou une défaillance causé intentionnellement ou accidentellement et capacité de retrouver l'état précédent la défaillance
- L'interopérabilité avec d'autres systèmes et outils de sécurité informatique.

1.4 Conclusion

Tenant compte de l'augmentation des menaces ciblant les applications Web, il est nécessaire de mettre en œuvre des contre-mesures de sécurité efficaces et performantes en créant de nouveaux mécanismes de sécurité ou en améliorant les technologies existantes.
Nous avons présenté dans ce chapitre les différentes classes de vulnérabilités Web pouvant être exploitées par les attaquants. Nous avons aussi exploré différents moyens de protection existants. Néanmoins, nul ne peut affirmer aujourd'hui qu'il existe des mécanismes de sécurité informatique infaillibles, d'une part, parce que les vulnérabilités ne sont pas toutes connues, d'autre part, parce-que les systèmes et technologies évoluent rapidement avec à chaque fois un lot de nouvelles vulnérabilités. Par conséquent, il est nécessaire d'évaluer l'efficacité de ces moyens. Nous avons alors présenté dans ce chapitre la méthodologie générale pour l'évaluation.
Nous nous focalisons dans le chapitre suivant sur deux de ces mécanismes qui sont les systèmes de détection d'intrusions et de détection de vulnérabilités Web et nous détaillons leurs principes, leurs limites et comment remédier à ces limites.

2

État de l'art

Introduction

De nombreuses attaques se focalisent sur les serveurs Web car ils sont souvent mal configurés ou mal maintenus. Les attaques les plus récentes profitent des failles de sécurité des applications Web que nous avons présentées dans le chapitre précédent. La détection de ces failles d'une part et des intrusions d'autre part trouve alors tous ses justificatifs en ce sens qu'à défaut de prévenir et protéger de façon sûre contre les attaques, les détecter permet néanmoins de limiter les dégâts et de réagir à temps.
Dans ce chapitre nous présentons un état d'art des techniques de protection pour les applications Web et nous nous focalisons principalement sur les systèmes de détection d'intrusion et les scanners de vulnérabilités Web. La section 2.1 est consacrée aux scanners de vulnérabilités Web tandis que la section

2.2 décrit en détail les différentes familles de systèmes de détection d'intrusion, en mettant l'accent sur leurs limites. La section 2.3 montre alors la nécessité de l'évaluation de ces techniques. Enfin la section 2.4 conclut ce chapitre.

2.1 Outils de détection de vulnérabilités web : Scanner web

Dans cette section, nous décrivons d'abord les principes des scanners de vulnérabilités et nous analysons les limites de ces outils en nous basant, d'une part, sur l'analyse des algorithmes mis en œuvre dans certains scanners disponibles en source libre, et d'autre part, sur des études expérimentales publiées dans la littérature.

2.1.1 Principe des outils de détection de vulnérabilités

Les attaques les plus courantes concernant les serveurs Web sont les attaques d'injection SQL (lorsque le serveur Web est connecté à une base de données SQL) et d'injection de code *Javascript* (réalisées sous la forme d'attaques de type *Cross Site Scripting* ou *XSS*). Ces injections de code proviennent de l'exploitation du même type de vulnérabilité des serveurs Web : l'absence de test de conformité des paramètres d'URL ou des données fournies dans les champs des formulaires.
Pour vérifier si ces attaques d'injection de code sont possibles, les outils de détection de vulnérabilités envoient des requêtes particulières et analysent les réponses retournées par le serveur. Un serveur peut répondre avec une *page de rejet* ou une *page d'exécution*. La page de rejet correspond à la détection par le serveur de valeurs d'entrée mal-formées ou invalides. Une page d'exécution est renvoyée par le serveur suite à l'activation réussie de la requête. Elle peut correspondre soit au scénario "normal", dans le cas d'une utilisation légitime du site, soit à un détournement de son exécution via l'exploitation réussie d'une injection de code (via des entrées non conformes). Pour identifier les vulnérabilités d'un site Web, les outils de détection de vulnérabilités soumettent au site des requêtes contenant des données non conformes correspondant à des attaques potentielles. Les réponses sont alors analysées afin d'identifier les pages d'exécution. Si une page d'exécution est identifiée, la page correspondante est considérée vulnérable. C'est ainsi que les outils détectent l'absence de test de conformité des paramètres. Tout le problème vient donc de l'analyse des réponses pour déterminer s'il s'agit réellement d'une page de rejet ou d'une page d'exécution.
Prenons l'exemple d'une page d'authentification qui utilise une base de données SQL pour conserver les couples *nom d'utilisateur / mot de passe* valides. Un outil de détection de vulnérabilités doit déterminer si la page d'authentification est vulnérable à une injection SQL (une telle injection permettant à un attaquant de contourner l'authentification). A la requête d'authentification soumise, le serveur peut retourner deux types de réponses : succès ou échec de l'authentification. Ces deux catégories de réponses se traduisent généralement par l'affichage de deux pages différentes au niveau du navigateur du client en terme de code HTML de façon à ce que l'utilisateur puisse constater qu'il a entré un couple valide ou pas. Ces pages peuvent varier dans leur forme, en fonction du langage utilisé, du site lui-même, du développeur, etc. Les outils de détection de vulnérabilités doivent donc automatiquement classifier la

réponse retournée afin de déterminer de façon correcte si la vulnérabilité est présente ou pas. Dans la suite nous analysons les stratégies adoptées par différents outils de détection de vulnérabilités en considérant des outils en source libre tels que *W3af 1.1* [1], *Skipfish 1.9.6b* [2] et *Wapiti 2.2.1* [3].
Le choix s'est porté sur des outils en source libre afin de disposer des sources des programmes et de pouvoir analyser les algorithmes mis en œuvre. Une telle démarche n'est pas possible avec des outils commerciaux tels que *Acunetix*, *WebInspect*, *AppScan*, etc.
On peut distinguer deux principales classes d'approches adoptées par les scanners de vulnérabilités :

1. par reconnaissance de message d'erreurs dans les requêtes renvoyées par le serveur
2. par l'étude de similarité des pages renvoyées.

Dans la suite nous décrivons les principes de ces deux approches en considérant la détection d'injections SQL comme exemple. On peut noter aussi que certains travaux, par exemple [HUANG ET AL. 03] utilisent ces deux approches de façon combinée.

2.1.1.1 Approche par reconnaissance de messages d'erreurs

Pour identifier les injections SQL, cette approche consiste à envoyer des requêtes d'un format particulier et chercher des motifs spécifiques dans les réponses tels que les messages d'erreurs de base de données. L'idée fondamentale est que la présence d'un message d'erreur SQL dans une page HTML de réponse signifie que la requête correspondante n'a pas été vérifiée par l' application Web avant d'être transmise au serveur de bases de données. Par conséquent, le fait que cette requête a été envoyée inchangée au serveur SQL révèle la présence d'une vulnérabilité. Les scanners tels que *W3af* (module SQLI), *Wapiti* et *Secubat* [STEFAN ET AL. 06] adoptent une telle approche. Nous analysons plus en détail leur fonctionnement dans les paragraphes suivants.

W3af a été créé par Andres Riancho en 2006. Il est considéré comme l'un des scanners les plus performants tel que mentionné dans le classement "Top 10 vulnerability scanners, Sectools Website" [4]. Il est écrit en Python. Son architecture modulaire permet aux utilisateurs d'importer et modifier facilement les différents modules qui le composent. En particulier, le module `sqli` vise à détecter des injections SQL dans les formulaires d'authentification, composés d'un champ permettant la saisie d'un nom d'utilisateur et d'un champ permettant la saisie du mot de passe. Plus précisément, il utilise trois requêtes formées à partir de l'injection SQL `d'z"0` (ou `d%2Cz%220` en code ASCII). Par exemple, pour détecter si le serveur Web est vulnérable à une injection SQL au travers du fichier `login.php`, à l'aide de la méthode POST et des paramètres `login` et `password`, *W3af* envoie les trois requêtes HTTP suivantes :

```
request("POST","login.php","login=&password=d%2Cz%220")
```

1. `http://w3af.sourceforge.net`
2. `http://code.google.com/p/skipfish`
3. `http://wapiti.sourceforge.net`
4. `http://sectools.org/web-scanners.html`

```
request("POST","login.php","login=d%2Cz%220&password=")
request("POST","login.php","login=&password=")
```

Les trois réponses associées à ces requêtes sont ensuite analysées. Si elles contiennent des messages d'erreur SQL, *W3af* informe l'utilisateur que l'application est vulnérable à une injection SQL. Pour détecter la présence d'un tel message, cet outil effectue une recherche, dans la page, de motifs caractéristiques des messages d'erreurs SQL provenant d'une base de données (en l'occurrence `Mysql_` et `Mysql_fetch_array()`). Aucun mécanisme supplémentaire n'est implémenté pour vérifier si la vulnérabilité existe réellement ou pas, c'est-à-dire si elle est réellement exploitable.
Wapiti est un autre exemple qui suit le même principe. Cet outil développé en Python, est capable de détecter des injections SQL, des injections XSS, des mauvaises manipulations de fichiers, des injections LDAP et des exécutions de commandes du système d'exploitation à partir d'une URL. Pour identifier des injections SQL, il envoie les deux requêtes suivantes :

```
request("POST","login.php","login="+randomChar(8)+"&password="+randomChar(8))
request("POST","login.php","login="+randomChar(8)+"&password=")
```

Une vulnérabilité est déclarée présente si un message d'erreur est identifié dans les réponses produites.
Enfin, *Secubat* développé dans [STEFAN ET AL. 06] adopte aussi une approche similaire. Il utilise une liste de messages d'erreurs obtenue par l'analyse de réponses des pages de sites web vulnérables, qui est destinée à couvrir un large éventail de réponses d'erreurs et une variété de serveurs de base de données.

2.1.1.2 Approche par étude de similarité des réponses

Le principe de cette approche consiste à envoyer différentes requêtes spécifiques aux types de vulnérabilités recherchées et à étudier la similitude des réponses renvoyées par l'application en utilisant une distance textuelle. En fonction des résultats obtenus et de critères bien définis, on conclut sur l'existence ou non d'une vulnérabilité. Prenons comme exemple l'approche adoptée par *Skipfish* pour détecter les vulnérabilités d'injection SQL.
Skipfish est un outil développé par Google afin de détecter des vulnérabilités sur des serveurs Web. Il procède en deux étapes. Dans une première étape, il parcourt le site et collecte toutes les pages qui lui semblent stables. Les autres sont ignorées. Pour détecter si une page est stable, *Skipfish* envoie 15 requêtes identiques et compare les réponses correspondantes. Si les réponses sont similaires, la page est considérée stable. Dans la deuxième étape, plusieurs tests sont réalisés sur ces pages stables, en fonction du type de vulnérabilités recherchées.
En particulier, un de ces tests concerne les injections SQL. Cette vulnérabilité est testée grâce à 3 requêtes A, B et C incluant chacune une injection SQL : A) `'"`, B) `\'\"` et C) `\\'\\"`.

Les réponses à ces 3 injections SQL sont comparées deux à deux dans le but d'identifier la présence d'une injection SQL. Selon *Skipfish*, une vulnérabilité est présente si les réponses associées

aux injections A et B ne sont pas similaires et si les réponses associées aux injections A et C ne sont pas non plus similaires. Le test de similarité utilise les fréquences d'apparition des mots dans les réponses. On peut faire les deux observations suivantes par rapport à la méthode utilisée dans *Skipfish* :

– le nombre de requêtes envoyées au serveur est faible. Cependant, il n'est pas inhabituel pour un site web de retourner des pages de rejet différentes. Par exemple, une page qui réagit à la requête A par une page de rejet contenant le message saisie incorrecte, et qui réagit aux requêtes B et C par une autre page de rejet contenant un message d'erreur SQL, sera considérée vulnérable à tort. Dans ce cas, il est nécessaire d'envoyer un nombre plus important de requêtes pour assurer une couverture plus large des différents cas de figure qui peuvent se présenter.
– La distance considérée pour l'étude de similarité considère la fréquence des mots sans tenir compte de l'ordre des mots dans un texte. Ignorer l'ordre des mots peut amener à ignorer la sémantique d'une page et à nouveau peut amener à mal juger si deux pages sont identiques ou non. Par exemple, les pages suivantes partagent le même vocabulaire, mais elles correspondent à une authentification réussie et échouée respectivement :

  Your are authenticated, you have not entered a wrong login.
  Your are not authenticated, you have entered a wrong login.

2.1.2 Analyse critique des scanners de vulnérabilités web

Les deux approches que nous venons de présenter dans la section 2.1.1 présentent un certain nombre de limites. Nous en présentons une synthèse dans les paragraphes suivants. L'efficacité de l'approche par reconnaissance de messages d'erreurs est liée à la complétude de la base de connaissance regroupant les messages d'erreurs susceptibles de résulter de l'exécution des requêtes soumises à l'application Web. Généralement, comme c'est dans le cas de *W3af*, on considère principalement les messages d'erreurs issus de la base de données. Cependant, les messages d'erreurs qui sont inclus dans des pages HTML de réponse ne proviennent pas forcément du serveur de bases de données. Le message d'erreur peut également être généré par l'application qui peut aussi reformuler le message d'erreur issu du serveur, par exemple pour le rendre compréhensible par le client. Par ailleurs, même si le message est généré par le serveur de base de données, la réception de ce message n'est pas suffisante pour affirmer que l'injection SQL est possible. En effet, ce message signifie que, pour cette requête particulière, les entrées n'ont pas été assainies, mais ne permet pas de conclure par rapport à d'autres requêtes SQL, en particulier celles qui seraient susceptibles de correspondre à des attaques réussies.
En ce qui concerne l'approche par similarité, elle se base sur l'hypothèse que le contenu d'une page de rejet est généralement différent du contenu d'une page d'exécution. Pour que cette comparaison puisse cependant être efficace, il est important d'assurer une large couverture des différents types de pages de rejet qui pourraient être générés par l'application. Ceci peut être réalisé en générant un grand nombre de requêtes visant à activer le plus grand nombre possible de pages de rejet variées. Cependant, les implémentations existantes de cette approche, en particulier dans Skipfish, génèrent trop peu de requêtes. *Skipfish* utilise seulement 3 requêtes. Si les réponses correspondent à différentes pages de

rejet, il conclut à tort que la vulnérabilité est présente conduisant ainsi à un faux positif. Par ailleurs, pour l'approche par similarité, comme dans tout problème de classification, le choix de la distance est très important. Celle utilisée dans *Skipfish* ne prend pas en compte les ordres des mots dans un texte. Cependant, cet ordre définit généralement la sémantique de la page. Il est donc important d'en tenir compte comme par exemple dans [HUANG ET AL. 03].
Enfin, il est à noter qu'aucun des outils que nous avons analysés dans la section 2.1.1 n'a été conçu pour générer automatiquement des requêtes d'attaque qui mènent à l'exploitation réussie de la vulnérabilité identifiée. Cependant, une telle possibilité serait utile pour déterminer si la vulnérabilité suspectée peut être effectivement exploitée et réduire ainsi le taux de faux positifs. Outre les analyses que nous venons de faire en considérant principalement les algorithmes implémentés dans *Skipfish*, *W3af* et *Wapiti*, d'autres études basées sur des analyses expérimentales ont aussi fait état de certaines limitations des scanners web, incluant des outils commerciaux. Nous détaillons ces études dans la sous-section 2.1.3.2.

Ces analyses montrent clairement le besoin de développer de nouvelles approches permettant d'améliorer l'efficacité des outils de détection de vulnérabilités et les capacités d'automatisation des campagnes d'évaluation. Les travaux présentés dans le cadre de ce manuscrit s'inscrivent dans cette optique.

2.1.3 Évaluation expérimentale

Outre les analyses que nous venons de faire dans la section 2.1.1 en considérant les algorithmes implémentés dans *Skipfish*, *W3af* et *Wapiti*, d'autres études récentes ont été effectuées afin d'évaluer l'efficacité des scanners web en utilisant plutôt des approches expérimentales. Ces études s'appuient sur l'utilisation d'applications vulnérables et l'activation de ces applications avec des entrées qui sont conçues pour activer ces vulnérabilités.
On peut distinguer trois types d'applications vulnérables utilisées dans ce type d'études :

1. Des applications qui n'ont pas été volontairement conçues pour être vulnérables et pour lesquels un certain nombre de vulnérabilités ont été découvertes et publiées. Il s'agit souvent d'applications développées dans le cadre de projets en source libre. On peut citer par exemple *WordPress* [5] et *phpBB* [6].

2. Des applications qui ont été volontairement conçues pour être vulnérables en ciblant différents types de vulnérabilités. On peut citer par exemple *WackoPicko* [7] [DOUPE ET AL. 10] et *Insecure* [8].

3. Des applications vulnérables qui ont été générées automatiquement à partir d'applications non vulnérables en utilisant une approche d'injections de vulnérabilités.

5. `http://fr.wordpress.org/`
6. `http://www.phpbb.fr/`
7. `https://github.com/adamdoupe/WackoPicko`
8. Application vulnérable développée dans le cadre du projet ANR DALI

Dans la suite nous décrivons brièvement le principe des approches par injection de vulnérabilité, et nous présentons ensuite des exemples d'études expérimentales sur l'évaluation de scanners de vulnérabilités Web qui ont utilisé ces différents types d'applications vulnérables.

2.1.3.1 Injection de vulnérabilités

L'approche par injection de vulnérabilités a été considérée par exemple dans [FONSECA 10],[FONSECA ET AL. 09] et dans [NEVES ET AL. 06]. Cette approche résulte de l'adaptation au domaine de la sécurité des techniques d'injection de fautes qui ont été largement utilisées pour évaluer les systèmes tolérants aux fautes [ARLAT ET AL. 93],[IYER 95], et également dans le contexte du logiciel [CHRISTMANSSON ET CHILLAREGE 96], [CROUZET ET AL. 98], [DURAES ET MADEIRA 06]. La technique consiste à injecter artificiellement des fautes ou des erreurs qui sont censées produire des effets similaires à ceux provoqués par des fautes ou erreurs observées dans des conditions opérationnelles. Les fautes injectées peuvent affecter par exemple le contenu des données, les caractéristiques temporelles, le flot de contrôle, etc. L'objectif est de tester et évaluer l'efficacité des mécanismes et des outils destinés à identifier ces erreurs quand elles sont activées au niveau du système.
L'utilisation de techniques d'injection de vulnérabilités pour évaluer la sécurité est en fait un cas particulier de l'injection de fautes logicielles, où les fautes injectées sont volontairement choisies pour qu'elles puissent être exploitées et mener à la compromission du logiciel. Elles opèrent sur le code source de l'application. A titre d'exemple la figure 2.1 résume la méthodologie décrite dans [FONSECA 10] et [FONSECA ET AL. 09]. On distingue les étapes suivantes :

1. L'analyse statique du code source conduisant à l'identification des variables d'entrée et des variables de sortie ainsi que les dépendances entre les différents fichiers manipulés par l'application.

2. Identification des éléments du code source dans lesquels il est possible d'injecter des vulnérabilités en fonction du type de vulnérabilités considéré.

3. La mutation du code conformément à des règles et des patterns de vulnérabilités prédéfinis [FONSECA ET AL. 08B]. Cette étape produit un mutant qui correspond à une version de l'application contenant la vulnérabilité injectée.

Cette procédure peut être répétée automatiquement pour chaque élément du code identifié à l'étape 2 et pour chaque type de vulnérabilité à injecter, de façon à produire un ensemble de fichiers, chacun avec une vulnérabilité différente injectée.
La définition des types de mutation et l'identification des éléments du code où ces mutations sont mises en œuvre ont été guidées par une analyse des caractéristiques des vulnérabilités observées sur des applications réelles en considérant principalement des injections SQL et des XSS [FONSECA ET VIEIRA 08A].

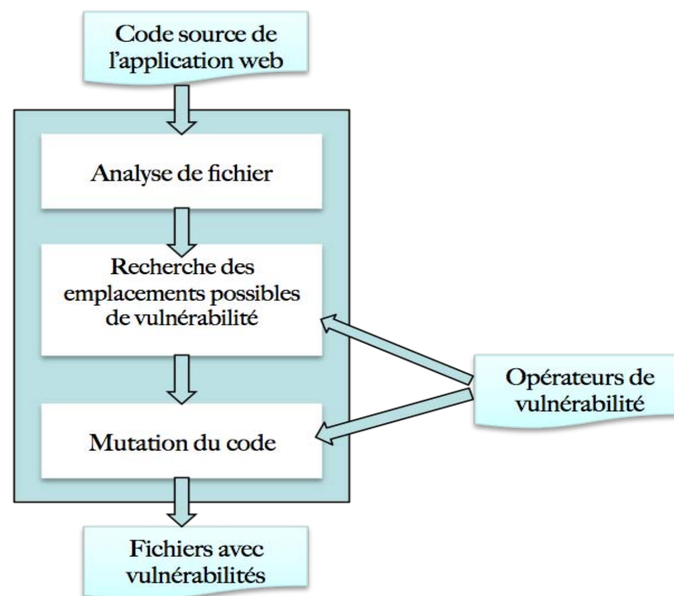

FIGURE 2.1 – Méthodologie d'injection de vulnérabilités [FONSECA 10]

2.1.3.2 Études expérimentales : Exemples

Plusieurs études expérimentales ont été effectuées ces dernières années pour évaluer l'efficacité de scanners de vulnérabilités Web. On peut citer par exemple les études comparatives récentes publiées dans [BAU ET AL. 10], [DOUPE ET AL. 10] et [FONSECA ET AL. 07B]. Ces études ont aussi fait état de certaines limitations des scanners web et ont observé en particulier des taux relativement élevés de faux positifs et de faux négatifs pour certains scanners.
En particulier, dans [BAU ET AL. 10], les auteurs ont analysé huit scanners, dont des scanners commerciaux réputés dans le domaine de l'analyse de la sécurité des applications web. Trois applications web contenant des vulnérabilités connues (SQL, XSS, CSRF, etc.) ont été utilisées : *Drupal*, *phpBB2*, et *Wordpress*. Les expérimentations ont révélé des taux moyens de détection de vulnérabilités variant de 0 à $62,5\%$ selon le type de vulnérabilités considérées. Pour les injections SQL, ce taux moyen en considérant les différents scanners est de l'ordre de $21,4\%$ (sachant que le taux de détection pour le meilleur d'entre eux était de l'ordre de 40%), ce qui est très modeste comparé aux performances espérées.
Des observations similaires ont été faites dans l'étude présentée dans [DOUPE ET AL. 10] dans laquelle onze scanners de vulnérabilités ont été analysés, dont W3aF. Dans cette étude les auteurs ont développé leur propre application Web vulnérable, *WackoPicko*. Il s'agit d'un site de partage et d'achat de photos qui contient 16 vulnérabilités qui ont été conçues pour être représentatives du top 10 de l'OWASP (cf. Section 1.2). Cette étude a montré aussi le besoin d'améliorer les capacités d'automatisation des campagnes effectuées avec ces outils et des possibilités d'exploration des applications web permettant d'identifier les pages vulnérables ainsi que les entrées et les paramètres correspondants.
Les résultats observés dans les deux études présentées dans [DOUPE ET AL. 10] et [BAU ET AL. 10] ont

aussi été confirmées par les analyses expérimentales effectuées dans [FONSECA ET AL. 09] en considérant des applications dans lesquelles des vulnérabilités ont été injectées automatiquement en utilisant la technique décrite dans la section 2.1.3.1. Trois applications web ont été considérées et 54 vulnérabilités ont été injectées au total. Les évaluations menées sur deux scanners commerciaux ont révélé des taux de détection faibles inférieurs à 10%.
Ces analyses montrent clairement le besoin de développer de nouvelles approches permettant d'améliorer l'efficacité des outils de détection de vulnérabilités et les capacités d'automatisation des campagnes d'évaluation. Les travaux présentés dans le cadre de ce manuscrit s'inscrivent dans cette optique.

2.2 Détection d'intrusions pour des applications Web

Les serveurs Web sont un environnement de test intéressant pour la détection d'intrusions, d'une part, par leur importance et par l'universalité du protocole HTTP [FIELDING ET AL. 99] (Hypertext Transfert Protocol) et, d'autre part, par le nombre de vulnérabilités frappant.
Les outils de détection d'intrusions "génériques" que nous avons présentés dans la section 1.3.3.2 du premier chapitre, peuvent être utilisés pour détecter les intrusions contre les serveurs web : les NIDS comme *Bro* [9] [PAXON 99], *NSM* [HEBERLEIN ET AL. 90], [MUKHERJEE ET AL. 94] (*Network Security Monitor*) ou *Snort* [ROESCH 99] ou les Host-based IDS qui surveillent le comportement de programmes tels que ceux développés par Forrest et al. [FORREST ET AL. 96],[WARRENDER ET AL. 99] ou Ghosh [GHOSH ET AL. 00] par exemple. Bien que ces IDS n'aient pas été spécialement évalués dans le domaine de la détection d'intrusions Web, le Web reste un domaine de prédilection pour la détection d'intrusions notamment pour les IDS au niveau réseau : dans la version 2.3.3 de *Snort*, 1064 signatures sur les 3111 sont consacrées à la détection des attaques web.

D'autres techniques de protection spécifiques à certains types d'attaques Web comme le XSS (Cross-Site-Scripting) ou les attaques par injection de code (comme les injections SQL, XML ou XPATH) ont été proposées [FONSECA ET AL. 07A]. Ces techniques sont efficaces pour détecter et stopper le type d'attaque pour lesquelles elles ont été conçues. Cependant, elles sont souvent trop intrusives dans l'application, donc lourdes à intégrer ou à exploiter à grande échelle dans les applications industrielles. D'autre part, ces solutions dépendent généralement du code de l'application et sont donc spécifiques à un langage ou une technologie (J2EE ou PHP par exemple), autrement dit, elles nécessitent le code source de l'application.
L'interception et l'assainissement des requêtes émises par le client avant qu'elles n'atteignent le serveur permet également de protéger une application Web d'éventuelles attaques. Les outils qui emploient cette technique assainissent toute donnée suspecte envoyée au serveur. Ces outils peuvent être installés soit directement sur le serveur [HUANG ET AL. 03],[PIETRAZEK ET BERGHE 05],[NGUYEN ET AL. 05], soit entre le client et le serveur [MOUELHI ET AL. 10], sous forme de proxy. Un exemple d'un tel outil est Noxes [KIRDA ET AL. 06]. Potentiellement, tout type de pare-feu applicatif peut réaliser ces vérifications.

9. `http://bro-ids.org/`

Les approches de détection d'intrusions, par scénario et comportementale, ont été explorées dans le monde de la recherche et se sont concrétisés aussi par le développement d'outils commerciaux, pour la protection des applications Web. Nous allons donc présenter les différents IDS spécifiques au Web suivant leur approche de détection.

2.2.1 Les approches par signature

Les IDS par signatures spécifiques au Web sont pour la plupart des HIDS au niveau applicatif. Ils évitent certains écueils des NIDS : reconstruction des paquets, perte de paquets en cas de charge, vulnérabilité aux techniques d'évasion, gestion de la cryptographie, etc. La plupart de ces outils utilisent les fichiers d'audit des serveurs web comme source d'événements.
[MCHUGH 00] et [PROCTOR 01] adoptent le principe de cette approche qui consiste selon eux à appliquer des techniques d'apprentissage sur les attaques connues de manière à en définir leurs signatures. Ensuite, à l'aide d'expressions régulières ou de correspondance de motifs ces dernières sont utilisées pour reconnaître les attaques dans les flots de requêtes.
Citons également le travail de Vigna et al. [VIGNA ET AL. 03] qui se place dans le cadre de la détection d'intrusion par scénarios et a mené à la réalisation d'un IDS nommé *WebSTAT*. Les auteurs s'intéressent à la détection d'intrusion sur les serveurs web. Les attaques sont dans un premier temps modélisées dans un langage de haut niveau, grâce au framework STAT [10] puis automatiquement compilées pour être utilisées comme signature lors de la détection d'intrusion. STAT leur permet d'exprimer des attaques complexes en terme d'états et de transitions.
WebSTAT se positionne ainsi comme un IDS à états. Il a en effet la capacité de détecter des attaques temporelles, en prenant en compte l'historique des évènements passés. Lors de la détection d'attaques, *WebSTAT* a la particularité de prendre en compte différentes sources d'évènements et de les corréler entre elles pour améliorer la détection. Ainsi, des évènements bas niveaux issus des paquets réseaux et des logs du système d'exploitation sont mis en relation avec les logs du serveur. Cette caractéristique permet de gagner en efficacité en réduisant le nombre de faux positifs. Nous citons également *Snort*[CASWELL ET AL. 03] et *BRO*[PAXON 99], IDS qui sont très répandus et en source libre open source et se basent sur cette approche.

Almgren et Lundqvist [ALMGREN ET LINDQVIST 01] ont proposé un système de détection d'intrusions intégré à un serveur Apache. L'avantage de cette solution réside dans sa capacité à détecter les intrusions à différents stades du traitement de la requête. Cette méthode entraîne également une dégradation dans les performances du serveur mais est également spécifique au serveur Apache. Cet outil a également la possibilité de recevoir des informations provenant d'autres sources, notamment des flux réseaux. Nous constatons que cet IDS est à rapprocher de *ModSecurity* que nous avons présenté dans la section 1.3.3.1 du premier chapitre, qui est un module pour Apache permettant d'écrire des règles pour détecter, bloquer, modifier les requêtes parvenant au serveur Apache.

Bien que les approches par signature soient effectives, elles posent certains problèmes. La plupart

10. `http://www.cs.ucsb.edu/?seclab/projects/stat/software/statframework.html`

des applications Web sont spécifiques et sont développées rapidement sans souci de sécurité particulier. Il est difficile d'écrire des signatures pour ces applications car il n'y a pas forcément de caractéristiques communes contrairement aux buffer overflows. Les entreprises n'ont pas forcément le temps et les ressources nécessaires pour employer un expert pour écrire ces signatures. L'approche comportementale que nous présentons dans la section suivante semble ici adaptée à la nature des vulnérabilités.

2.2.2 Les approches comportementales

Pour cette catégorie d'IDS, nous distinguons trois approches qui présentent des niveaux d'analyse différents : une approche "boîte noire", "boîte grise" et "boîte blanche". Chacun d'eux est basé sur le type d'information disponible pour construire le modèle de référence de l'application.

2.2.2.1 Approche boîte noire

Les approches de type "boîte noire" n'utilisent aucune information interne du programme. Le modèle de comportement de référence à définir dans ces approches peut être issu de la spécification de l'application ou bien déduit par apprentissage à partir de l'exécution de l'application.
L'approche considérée dans les travaux de Forest et al. [FORREST ET AL. 96] et dans [HOFMEYR 98] se base sur l'analyse des enchaînements des appels systèmes des processus lors de l'exécution, donc sur des informations externes au programme. L'expérience a en effet montré que de courtes séquences d'appels système génèrent une signature stable pour modéliser le comportement normal d'un processus par rapport à son environnement.

Robertson et al. [ROBERTSON ET AL. 06] présentent une amélioration des travaux précédents et ils s'intéressent aux problèmes récurrents de la détection d'intrusions comportementale. Ils présentent deux techniques permettant à un administrateur de gagner du temps en connaissant la nature des intrusions et leur criticité en ajoutant à la détection d'anomalies deux composants : un composant permettant la génération de signatures d'alertes et groupant les alertes suivant les signatures et un composant permettant d'identifier les anomalies suivant des heuristiques.
Contrairement à la détection d'intrusions comportementale, l'approche par scénarios présente l'avantage d'avoir moins de faux positifs et de connaître le type de l'attaque détectée, grâce à la base de signatures. Robertson et al. désirent se rapprocher de ces caractéristiques pour la détection d'intrusion comportementale. Ils proposent tout d'abord une étape de généralisation, qui à partir d'une requête Web, crée une signature. Les caractéristiques telles que la taille des paramètres ou la distribution des caractères de la requête Web sont utilisées pour créer un modèle afin de pouvoir regrouper ensuite les requêtes présentant des propriétés similaires. Par ailleurs, cette approche propose une étape d'inférence des classes d'attaques des intrusions. Les intrusions sont confrontées à des heuristiques pour savoir s'il s'agit d'une attaque visant un débordement de buffer, une exploitation XSS, une injection SQL, etc.
Précédemment à ces travaux, des approches modélisant le comportement normal d'un utilisateur ont été proposées [GILHAM ET AL. 92], [VALDES ET AL. 95]. Ces dernières modélisent des profils en analysant les logs du système. Il s'est avéré que l'analyse des appels système est plus avantageuse en tout

point de vue. En effet, le nombre de comportements différents d'un programme est borné, contrairement à celui d'un utilisateur qui est susceptible de générer un très grand nombre d'actions différentes.

Ces approches en "boîte noire" ont cependant des faiblesses. En effet, elles sont sujettes aux attaques dans lesquelles un attaquant imite le comportement attendu de l'application surveillée avec pour objectif d'agir sur le flot de contrôle de l'application. Cela aura pour conséquence de générer des appels système valides du point de vue de l'IDS et donc d'éviter la détection de l'attaque. Pour tenter de pallier ces attaques, différents travaux ont proposé d'utiliser des informations additionnelles sur l'état interne de l'application et de les ajouter au modèle basé sur les séquences d'appels système. Ces approches sont appelées approches en "boîte grise" (voir section2.2.2.2).

La deuxième catégorie d'approches de détection de type "boîte noire", est basée sur les spécifications connues "a priori". On peut citer en particulier les travaux de Ko et al. [KO ET LEVITT 94], [KO ET LEVITT 97] et Sekar et al. [SEKAR ET AL. 02]. Les auteurs proposent de surveiller la séquence des actions réalisées par un programme en cours d'exécution et de comparer cette séquence à la spécification d'un modèle de comportement attendu. Dans le contexte plus général du génie logiciel, les approches basées sur les spécifications, utilisant par exemple la notion de contrat, ont été traditionnellement utilisées pour la détection d'erreurs à l'exécution. En effet, la conception par contrat [MEYER 92] est une technique permettant d'embarquer des éléments de spécification formelle (invariants, pré-et post-conditions) dans une conception généralement pour des programmes orientés objet.
L'extension de ces techniques au contexte de la sécurité et de la détection d'intrusions a fait l'objet de plusieurs travaux récents. On peut citer par exemple, l'étude présentée dans [MOUELHI ET AL. 10] qui s'est intéressée à l'adaptation et l'utilisation des contrats comme une technique possible pour détecter les intrusions. En effet, leur approche s'appuie sur la spécification de contrats construits automatiquement à partir de l'analyse des formulaires de l'application protégée ou bien définis par l'utilisateur. Ces contrats décrivent des contraintes s'appliquant aux entrées des utilisateurs. Cette approche se veut déclarative puisqu'elle définit le comportement autorisé par des utilisateurs d'une part et elle est capable de détecter de nouvelles attaques, si ces attaques violent des contraintes, d'autre part. Mais elle reste incomplète car elle est basée principalement sur la définition syntaxique des contrats liés aux différents paramètres à saisir dans un formulaire de l'application.

2.2.2.2 Approche boîte grise

Tout comme l'approche en boîte noire, l'approche en boîte grise est fondée sur les séquences d'appels système. Cependant, elle extrait des informations additionnelles du processus, notamment en utilisant la mémoire.[GAO ET AL. 04]. L'expérience a montré que la présence d'une attaque se manifeste souvent dans les arguments des appels systèmes. Se basant sur ce constat, Kruegel et al. [KRUEGEL ET AL. 03] proposent de prendre en compte les arguments des appels système pour améliorer la technique introduite par Forest et al. [FORREST ET AL. 96] et [HOFMEYR 98]. Pour cela, les arguments des appels système sont analysés suivant plusieurs modèles et chacun de ces modèles est instancié pour chaque appel système.

Dans [GAO ET AL. 04], les auteurs introduisent un nouveau modèle de détection, c'est un modèle qui présente le comportement des appels systèmes sous forme de graphe, nommé graphe d'exécution. L'objectif de ce travail est de se rapprocher le plus possible d'une analyse de type boîte blanche sans que cela requière une analyse statique des sources du programme. Par conséquent, les auteurs proposent cette solution lorsqu'une approche boîte blanche n'est pas envisageable.

2.2.2.3 Approche boîte blanche

Dans le cas des approches en "boîte blanche", les informations présentes dans les sources du programme peuvent être utilisées pour construire un modèle de détection d'intrusion au niveau applicatif. En effet, nous considérons ici le code du programme qui peut être exploité par analyse statique ou dynamique afin d'en dériver un modèle approprié. Cette approche peut être utilisée pour détecter à la fois des attaques contre le flot de contrôle de l'application et des attaques contre les données. En particulier, des attaques contre les données peuvent avoir pour conséquence de modifier le flot de contrôle de l'application. Parmi les approches en "boîte blanche", nous avons exploré celles qui construisent le modèle de référence en se fondant sur les invariants d'un programme, en particulier pour la détection d'attaques contre les données d'un programme. Il s'agit dans ce cas d'attaques qui auront comme effet de corrompre des données saines d'un programme de façon à invalider certains invariants définis dans le modèle de référence [CHEN ET AL. 05]. Dans ce cadre, [COVA ET AL 07] propose de surveiller l'état interne d'une application Web afin de détecter des attaques susceptibles d'altérer le flot de contrôle ou bien les données. L'état de l'application à un point d'exécution donné est défini par les valeurs des variables caractérisant la session de l'utilisateur. Dans cette approche, nommée *Swaddler*, différents modèles sont utilisés pour surveiller l'état interne de l'application. Durant la phase d'apprentissage, l'application est dans un premier temps instrumentée avec du code pour extraire les valeurs des variables en différents points critiques. L'instrumentation est ainsi réalisée au début de chaque bloc de base ; un bloc de base étant un bloc de code sans possibilités d'arrêt ou de branchements. Les valeurs récupérées sont ensuite analysées pour créer un profil en différents points de l'application. Plusieurs propriétés sont ainsi capturées : des propriétés sur des variables uniques mais aussi sur des relations entre différentes variables. *Swaddler* introduit la notion d'invariant et utilise en partie l'outil Daikon pour établir des relations complexes entre variables. Les profils créés sont ensuite utilisés pour détecter des attaques à l'exécution. L'implémentation de *Swaddler* a été réalisée en PHP et d'après les tests effectués, deux facteurs sont déterminants sur les performances de l'application surveillée : le nombre de variables analysées dans chaque bloc de base et le nombre de blocs de base.

Une autre approche, développée par Ludinard et al. dans [LUDINARD ET AL. 11], basée sur un principe similaire que celui de *Swaddler* consiste à définir le comportement légitime d'un programme par des contraintes - ou invariants - sur les variables critiques. Les auteurs se sont servis ensuite de la modélisation de ce comportement normal pour détecter des infractions à l'exécution du programme. Contrairement à *Swaddler*, ils ne se sont pas concentrés uniquement sur les variables caractérisant les sessions des utilisateurs. Il ont identifié les variables qui peuvent potentiellement être utilisées dans une attaque et ils ont considéré tous les types de variables : attributs d'un objet, variables locales, sessions,

cookies. Ces travaux se sont focalisés sur la détection des attaques ciblant les données de l'application et qui se manifestent par l'invalidation d'un des invariants décrivant le comportement normal de l'application. Ludinard et al. visent à déterminer un certain nombre de propriétés vérifiées par les données manipulées en interne par l'application, et qui sont susceptibles d'avoir une influence importante, directe ou indirecte, sur les séquences d'appels système émises et sur leurs arguments, à chaque exécution normale du programme (c'est-à-dire pour toutes les exécutions dépourvues d'attaques). Ces propriétés permettent de définir des invariants.

2.2.3 Approche hybride

Une approche hybride a été proposée par Tombini et al. [TOMBINI ET AL. 04], [DEBAR ET TOMBINI 05]. Cette approche consiste en la sérialisation d'un IDS comportemental suivi d'un IDS par signature. L'IDS comportemental permet de filtrer les requêtes normales et ainsi seules les requêtes détectées comme anormales sont passées à l'IDS par signature. Bien que l'IDS comportemental utilisé soit simple, ceci permet de réduire le nombre de faux positifs générés globalement. La source d'entrées est le fichier d'audit du serveur Web. Cet IDS est donc soumis aux mêmes problèmes que les autres utilisant cette source de données.

2.2.4 Analyse critique

Les différents types d'IDS, classés selon différentes caractéristiques et décrits aux paragraphes 2.2.2.1 et 2.2.2.2 présentent des limites que nous avons citées au fur et à mesure. Dans cette section, nous résumons ces limites en les classant. La plupart des IDS souffrent au moins des problèmes suivants selon les études faites dans [AXELSSON 00] :

- *Problème d'intégrité des données analysées* : l'information, pour parvenir à l'IDS depuis sa source, traverse un certain parcours. Elle peut alors, en cours de route, faire l'objet de manipulation et suppression de la part d'un intrus de telle manière à cacher ses traces et de passer inaperçu vis-à-vis de l'IDS. Il se pose alors le problème de l'intégrité des informations analysées par l'IDS par rapport aux informations au niveau de leur source.
- *Problème de fiabilité* : un IDS est lui-même un système pouvant subir des attaques spécialement dirigées contre les différents modules qui le composent. Il peut alors être modifié, reconfiguré ou complètement désactivé. De plus, un IDS est implémenté sous forme d'un ensemble de programmes dont le bon fonctionnement peut, à cause de certains imprévus (problème de concurrence par exemple), être corrompu. Il n'est pas non plus à exclure qu'un IDS contienne comme tous les systèmes, des défauts de conception, implémentation, configuration, etc. Il se pose ici le problème de la fiabilité de ses résultats dès lors qu'il est lui même vulnérable et qu'il peut faire l'objet de manipulations malveillantes.
- *Problème d'utilisation de ressources* : un IDS devrait fonctionner de préférence continuellement et en temps réel, ce qui implique une utilisation importante de ressources de stockage et de calcul. Ceci est particulièrement le cas pour les IDS réseaux confrontés d'un côté à des

débits réseaux de plus en plus élevés et des bases de signature dont la taille augmente de manière exponentielle. Ce problème peut être présent aussi dans d'autres types d'IDS au niveau applicatif par exemple dans le cas de la protection d'applications Web.

– *Faux positifs et faux négatifs* : Normalement, l'avantage de la détection d'intrusions par scénario (détection d'abus) devrait être un faible taux de faux positifs (fausses alarmes) en partant de l'hypothèse qu'il est possible de définir des signatures précises des attaques connues. Cependant, l'expérience montre que ce n'est pas toujours le cas. Ce constat a été observé dans plusieurs travaux. Le problème se pose également pour la détection d'anomalies. En fait, la détection d'anomalies est capable de détecter les attaques inconnues ; toutefois, elle n'est pas nécessairement aussi efficace que la détection d'abus pour les attaques connues. Notamment, un fort taux de faux positifs ou négatifs peut être rencontré si le paramétrage de l'IDS n'a pas été réalisé avec soin.

Ces limites montrent l'importance de mettre en œuvre des expérimentations afin d'évaluer l'efficacité des techniques de détection d'intrusion en soumettant les IDS à différentes conditions d'utilisation et différents hypothèses de scénarios d'attaque. La connaissance issue de ces expérimentations sera aussi utile pour identifier des pistes pour améliorer la conception de ces IDS.

2.3 Évaluation de systèmes de détection d'intrusions

Dans cette section, nous nous intéressons plus particulièrement aux travaux menés sur l'évaluation des IDS.

Historiquement, l'un des projets les plus ambitieux dans ce domaine a été sponsorisé par la DARPA. Les expérimentations sur l'évaluation ont débuté en 1998 par le MIT Lincoln Laboratory, puis ont continué en 2000 dans le projet Lincoln Adaptable Real-time Information Assurance Test-bed (LARIAT) [HAINES ET AL. 01][LIPPMANN ET AL. 00]. Le but était de fournir un ensemble significatif de données de test, comprenant du trafic du fond et des activités intrusives. Les jeux d'attaques utilisés pour mener ces expérimentations sont devenus une référence dans la communauté et ont largement été repris pour évaluer de nouvelles techniques de détection d'intrusion [ATHANASIADES ET AL. 03]. D'autres travaux sur l'évaluation des IDS ont aussi été menés, couvrant différentes techniques de détection d'intrusions au niveau des protocoles réseau [VIGNA ET AL. 04],[GADELRAB 08],[MARTY 02][MASSICOTTE ET AL. 06], du système hôte [PUKETZA ET AL. 96], au niveau applicatif ou plus particulièrement dans le contexte des applications et serveurs Web [11][FONSECA ET AL. 09]. Une comparaison des techniques précédentes est présentée par Ingham et Inoue [INGHAM ET AL. 07]. Ils ont constitué une base de 63 attaques Web et ont collecté un ensemble de requêtes HTTP pour 4 sites distincts. Ils proposent et mettent à disposition de la communauté un framework permettant la comparaison des outils de détection d'intrusions spécifiques au Web. Ils montrent que les méthodes s'attachant aux unités lexicales (token) sont sensiblement plus performantes que les méthodes s'attachant à modéliser les requêtes au niveau des octets.

11. `http://www.caw.com/product/index.shtml`

Les travaux effectués le cadre du projet de la DARPA ont aussi fait l'objet de plusieurs critiques [McHugh 00], concernant en particulier la représentativité des attaques, et la méthodologie utilisée pour analyser les résultats et évaluer les mesures. De plus, les jeux d'attaque utilisés, ciblant principalement le trafic réseau et les systèmes d'exploitation, sont maintenant obsolètes et ne sont pas pertinents pour évaluer les techniques de détection d'intrusions pour des applications web.

La génération de jeux d'attaques constitue une étape importante dans l'évaluation des IDS. Dans les premiers travaux sur l'évaluation des IDS, les jeux d'attaques étaient définis manuellement de façon ad-hoc en se basant sur les attaques connues. Dans le contexte des applications Web, il existe aussi des listes qui répertorient des exemples d'attaques ciblant en particulier des injections SQL ou des vulnérabilités XSS (appelées "SQL sheets" [12] [13] ou "XSS sheets" comme par exemple dans [Kiezun et al. 09]). Ces approches manuelles ne sont cependant pas flexibles.
Nous présentons dans le tableau 2.1 quelques exemples de SQL Sheet.

MySQL Injection Cheat Sheet	
Basique	SELECT * FROM login
	SELECT * FROM login WHERE id = 1 or 1=1
	SELECT * FROM login WHERE id = 1 or 1=1 AND user LIKE "%root%"
Insertion d'un nouveau utilisateur SQL : 1-Normal 2- bypass	1- insert into login set user = 'root', pass = 'root'
	2- insert into login set user = 0x726F6F74, pass = 0x726F6F74
Affichage de la table	SELECT * FROM login WHERE id = 1 or 1=1 ; SHOW TABLES
Sélection de la version	SELECT * FROM login WHERE id = 1 or 1=1 ; SELECT VERSION()
Collecte d'informations	SELECT COUNT(*) FROM tablename
Sélection hôte, utilisateur, BD de mysql.db	SELECT * FROM login WHERE id = 1 or 1=1 ; select host,user,db from mysql.db ;
Insertion d'un nouveau utilisateur dans la base	INSERT INTO login SET user = 'r00t', pass = 'abc'
Changer l'e-mail admin, pour "forgot login retrieval."	UPDATE users set email = 'mymail@site.com' WHERE email = 'admin@site.com' ;

TABLE 2.1 – Exemple de SQL Sheet

Il existe aussi des études qui se sont intéressées à la définition d'approches automatisées permettant de générer un nombre important d'attaques pour les campagnes d'évaluation des IDS. Dans ce contexte, on peut citer les travaux sur les stimulateurs d'IDS qui se sont intéressés à la génération automatique d'attaques à partir de l'analyse des signatures utilisées par certains IDS, par exemple *Snort* pour la détection d'intrusions. C'est le cas par exemple des recherches qui se sont concrétisées par le

12. http://www.justinshattuck.com/2007/01/18/mysql-injection-cheat-sheet,
13. http://ha.ckers.org/sqlinjection/

développement des outils Mucus[MUTZ ET AL. 03], *Snot* [14], *IDSWakeup* [15] et *Stick* [16] .
Au lieu de s'appuyer sur les signatures des attaques, les auteurs de [MUTZ ET AL. 03] proposent de générer automatiquement une large variété de jeux d'attaques par mutation d'exploits (c'est-à-dire, d'attaques réussies). Ces attaques sont construites en appliquant des opérateurs de mutation à des patterns construits à partir de l'analyse d'attaques connues. Les jeux d'attaques ainsi obtenus sont ensuite soumis à l'IDS selon une approche de test en boîte noire. Une approche similaire a été implémentée dans l'outil `Thor` [MARTY 02]. A notre connaissance, cette technique a été étudiée dans le cas d'attaques au niveau réseau.
Cette approche diffère d'autres techniques basées elles aussi sur le principe de mutation, mais qui considèrent plutôt l'injection de vulnérabilités dans le code source des applications cibles en adoptant une approche boîte blanche. C'est le cas par exemple de l'approche présentée dans [FONSECA ET AL. 09] qui a été décrite dans la Section 2.1.3.1, dans laquelle les vulnérabilités injectées dans le code sont ensuite utilisées pour construire des jeux d'attaques spécialement conçus pour exploiter les vulnérabilités injectées.

Une des difficultés partagées par ces différentes approches concerne la possibilité de définir automatiquement des attaques qui vont réussir à compromettre la cible. Les techniques utilisées dans les approches traditionnelles d'injection de fautes ne fournissent pas nécessairement des entrées valides pour l'application. De plus, il n'est pas toujours facile de déterminer si une attaque a effectivement réussi ou non.

2.4 Conclusion

Nous nous sommes focalisés dans ce chapitre sur les outils de détection de vulnérabilités Web, appelés scanners Web, et les systèmes de détection d'intrusions, ainsi que sur les approches existantes pour évaluer l'efficacité de ces outils. A la lumière de cette étude, nous avons constaté à partir des analyses critiques basées sur les algorithmes mis en œuvre dans ces outils, ainsi que sur les résultats issus d'études expérimentales, que ces différents outils révèlent certaines limitations et que leur efficacité est encore relativement modeste par rapport aux attentes.
Nous avons relevé en particulier la nécessité d'améliorer les possibilités d'automatisation des campagnes d'identification des vulnérabilités dans les applications Web et d'évaluation des mécanismes de protection mis en œuvre dans ces applications. Il s'agit en particulier de développer de nouvelles méthodes permettant de générer de façon systématique plusieurs scénarios d'attaques pour analyser la robustesse des applications cibles et l'efficacité des IDS destinées à les protéger vis-à-vis des malveillances. Les travaux menés dans le cadre de cette thèse visent à contribuer à cet objectif.
Les contributions présentées dans ce manuscrit concernent :

1. le développement d'une méthodologie permettant de générer automatiquement des scénarios d'at-

14. `http://www.securityfocus.com/tools/1983`
15. `http://www.hsc.fr/ressources/outils/idswakeup/index.html.en`
16. `http://www.packetstormsecurity.nl/distributed/stick.htm`

taque à partir de l'analyse selon une approche boîte noire de l'application cible et d'évaluer le comportement de l'application et des IDS associées en présence de ces attaques.

2. le développement d'une plateforme d'évaluation mettant en œuvre cette approche et permettant d'évaluer l'efficacité des mécanismes de protection des applications cibles.
3. la validation de la méthode proposée et de la plateforme d'évaluation en considérant différentes applications vulnérables et des techniques de détection d'intrusions développées en particulier dans le cadre du projet DALI.

Ces contributions sont développées dans les chapitres suivants. Dans le chapitre 3 nous présentons notre première contribution qui concerne une nouvelle approche pour la détection de vulnérabilités Web et un outil mettant en œuvre cette approche. Dans le chapitre 4, nous nous intéressons plus particulièrement à la génération de scénarios d'attaques permettant de prendre en compte les dépendances entre vulnérabilités. Enfin, le chapitre 5 présente la plateforme d'évaluation et son utilisation pour l'évaluation de techniques de détection d'intrusions pour des applications.

3

Détection de vulnérabilités par classification automatique des pages html

Introduction

A l'issue de l'analyse des différents scanners de vulnérabilités effectuée dans le chapitre précédent, nous constatons qu'il y a matière à améliorer leurs performances. La contribution décrite dans ce chapitre vise à atteindre cet objectif. En effet, nous proposons une nouvelle approche permettant la détection automatisée des différents types de vulnérabilités Web, correspondant aux attaques de type injections SQL, OsCommanding, File Include et XPath. Afin de valider et d'évaluer notre approche,

nous avons réalisé deux séries d'expériences, sur différents types d'applications.
Ce chapitre présente tout d'abord une vue de haut niveau de notre approche puis détaille l'algorithme de classification de pages Web sur lequel elle s'appuie. Un exemple concret permettant d'illustrer notre algorithme est ensuite proposé. Dans la première partie du chapitre, nous faisons le choix d'illustrer notre algorithme en nous focalisant sur les injections SQL. Dans une seconde partie, nous montrons comment notre approche peut également s'adapter à d'autres types d'injections, comme les attaques de type OsCommanding, File Include et Xpath. Enfin, nous terminons ce chapitre par la présentation de deux séries d'expérimentations réalisées sur différents types d'applications. Ces expérimentations nous ont permis de valider notre approche.

3.1 Aperçu global de l'approche

L'approche proposée [AKROUT ET DESSIATNIKOFF 10] s'appuie sur certains concepts hérités des outils existants et inclut des extensions significatives. Elle est basée sur la classification automatique des réponses retournées par les serveurs Web en utilisant les techniques de regroupement de données (ou clustering) et permet d'identifier les requêtes qui sont capables d'exploiter avec succès des vulnérabilités présentes. La génération automatique des requêtes permettant l'exploitation réussie des vulnérabilités est particulièrement utile pour faciliter la validation des applications Web et l'élaboration de tests de pénétration.

Notre approche comporte deux étapes : i) la recherche de points d'injection dans les pages du site et ii) la détection de vulnérabilités dans ces points d'injection. Nous définissons un point d'injection par une entrée d'une page dans laquelle il est possible d'injecter du code : un paramètre d'une URL ou un champ d'un formulaire.

Notre algorithme de détection de vulnérabilités vise à réduire le nombre de faux positifs en fournissant les requêtes qui ont réellement permis d'exploiter chaque vulnérabilité. Cet avantage est double puisque l'exploitation effective des vulnérabilités nous permet également de découvrir de nouvelles pages de l'application Web que nous ne pouvions atteindre auparavant. Ces nouvelles pages peuvent contenir de nouveaux points d'injection et éventuellement de nouvelles vulnérabilités.

La figure 3.1 présente une vue de haut niveau de l'approche proposée. Elle commence par l'analyse de l'URL initiale (qui correspond, la plupart du temps, à la page principale de l'application). A partir de cette URL, l'exploration identifie tous les points d'injection potentiellement vulnérables. Cette première étape se termine lorsque tous les points d'injection accessibles ont été atteints. La deuxième étape correspond à l'exécution de notre algorithme de classification, basé sur des techniques de clustering (que nous détaillons dans la section 3.2) sur chaque point d'injection. Cet algorithme permet l'identification et l'exploitation effective de vulnérabilités présentes. Il permet ainsi de découvrir de nouvelles pages qui peuvent contenir de nouveaux points d'injection qui n'étaient pas accessibles dans la première étape. Par conséquent, un nouveau domaine de l'application devient accessible. Par la suite, de façon itérative, la première étape de l'approche est ré-exécutée sur ce nouveau domaine, etc.

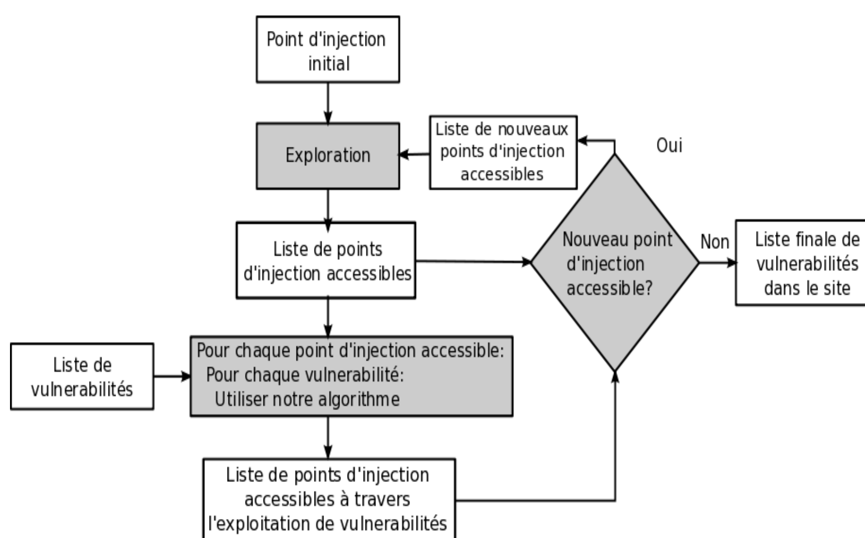

FIGURE 3.1 – Algorithme d'extraction de points d'injection et de recherche de vulnérabilités

Notre algorithme ([AKROUT ET AL. 11], [DESSIATNIKOFF ET AL. 10]) a été implémenté dans un outil développé en utilisant le langage Python, langage qui facilite grandement la gestion des concepts HTTP (cookies, paramètres, etc.). Cet outil utilise également le logiciel d'analyse statistique R [1]. Ce logiciel intègre un ensemble de programmes de clustering que nous allons détailler par la suite. Ils ont été utilisés pour développer notre algorithme de classification.
Cet outil est nommé *Wasapy*, qui signifie **W**eb **A**pplication **S**ecurity **A**ssessment in **Py**thon.

3.2 Algorithme de classification

Notre algorithme de classification [DESSIATNIKOFF ET AL. 11B] a pour but d'identifier, de manière automatique, si un point d'injection contient une vulnérabilité qu'il est possible d'exploiter avec succès. Pour cela, nous soumettons à chaque point d'injection identifié un ensemble de requêtes contenant :

1. des requêtes générées aléatoirement, notées R_a
2. des requêtes d'injection de code syntaxiquement invalides, notées R_{ii}
3. et des requêtes d'injection de code syntaxiquement valides, notées R_{iv}.

Les deux premiers ensembles de requêtes visent à identifier les différents types de pages de rejet qui peuvent être générées par l'application, suite à une exécution non réussie de la requête envoyée par un client. L'identification des requêtes d'injection de code réussies peut être ainsi effectuée en comparant les pages retournées avec celles obtenues à partir des deux premiers ensembles. Celles qui s'en

1. `http://www.r-project.org/`

écartent significativement correspondront vraisemblablement à des attaques réussies.
C'est cette idée de base qui est mise en œuvre dans notre approche. Grâce à une technique de regroupement en grappes (ou clusters) des réponses associées à ces trois ensembles de requêtes, notre algorithme peut automatiquement déterminer les requêtes qui ont réellement permis l'exploitation de la vulnérabilité, requêtes baptisées *requêtes d'injection réussie*.
Cet algorithme nécessite, en entrée, un point d'injection. Dans la suite, sans perte de généralité, nous supposons que cette entrée est un champ d'authentification et nous nous intéressons uniquement aux injections SQL. Nous présentons tout d'abord les trois ensembles de requêtes cités ci-dessus, puis la technique de regroupement des réponses associées. Cette technique est basée sur l'utilisation d'une distance et d'un seuil que nous présentons également.

3.2.1 Génération des trois ensembles de requêtes

Dans le contexte d'une authentification sur une application Web, notre objectif est d'identifier, parmi un ensemble d'injections SQL possibles, celles qui permettent effectivement de contourner l'authentification. Le principal défi réside dans l'automatisation de ce processus. Nous proposons une méthode qui vise à réduire à la fois le nombre de faux positifs et faux négatifs, comparé aux solutions des outils que nous avons analysés au chapitre 2.
Cette méthode se base sur plusieurs constats :

- les pages de *rejet* sont différentes des pages d'*exécution* en terme de contenu textuel
- deux pages de *rejet* peuvent être différentes
- deux pages d'*exécution* peuvent être aussi différentes

Pour s'en rendre compte, il suffit d'accéder à une page d'authentification d'un site et de tester la saisie de données valides et invalides. Par exemple, les réponses associées à des données valides contiendront des messages de bienvenue et les réponses associées à des données invalides contiendront des messages d'erreur relatifs au langage de programmation (PHP, ruby on rails, etc.) ou des messages d'erreur SQL. Le point important est l'existence de différences entre les pages de rejet et les pages d'exécution.

Notre approche vise à étudier ces différences afin de déterminer, parmi des réponses différentes, celles qui sont des pages d'exécution.
Pour débuter cette classification, nous avons besoin de requêtes initiales dont nous connaissons le type des réponses associées (rejet ou exécution). Clairement, il est plus facile de générer des requêtes qui engendrent une page de rejet. Il suffit par exemple de générer aléatoirement les noms d'utilisateur et mots de passe permettant de renseigner le formulaire d'authentification. Un autre exemple concerne les requêtes volontairement malformées générant des erreurs SQL.

Dans notre approche, nous distinguons les trois ensembles de requêtes R_a, R_{ii}, et R_{iv} définis au début de la section 3.2. Afin de les illustrer avec des exemples, nous considérons que nous envoyons ces requêtes à un serveur Web exécutant le script PHP (`page.php`) décrit dans la suite (ce script n'assainit

pas correctement ses paramètres) :

```
<?php
\$login = \$_GET['login'] ;
\$pass = \$_GET['pass'];
\$req = "SELECT * FROM users WHERE login=?\$login' and pass='\$pass'";
\$res = mysql_query(\$req);
if (\$row = mysql_fetch_assoc(\$res)) {
  print("Authentification réussie");
} else {
  print("Authentification échouée");
}
?>
```

R_a est l'ensemble des requêtes générées à partir de mots aléatoirement choisis dans la liste [a-zA-Z0-9]+. Vraisemblablement, ces requêtes engendreront des pages de rejet.
Par exemple, la requête suivante :

```
http://address/directory/page.php?login=agfe&pass=mlrd
```

génère la requête SQL suivante :

```
SELECT * FROM users WHERE login='agfe' and pass='mlrd'
```

Cette requête SQL a toutes les chances d'échouer, bien qu'il existe une probabilité négligeable que le mot de passe et le login générés aléatoirement soient valides.

R_{ii} est l'ensemble des requêtes d'injection SQL syntaxiquement invalides pour le point d'injection. Elles sont construites de façon à ce que le serveur SQL qui interprète ces requêtes génère une erreur. Cette erreur peut être propagée directement au client (c'est-à-dire qu'elle sera sur son navigateur) ou interceptée par le serveur Web qui peut alors la reformuler et renvoyer un message d'erreur différent du message original qui serait plus parlant pour le client. Par exemple, la requête suivante :

```
http://address/directory/page.php?login=test&pass='
```

génère la requête SQL suivante :

```
SELECT * FROM users WHERE login='test' and pass='''
```

Cette requête SQL est syntaxiquement invalide puisqu'elle contient un nombre impair d'apostrophes. L'exécution de cette requête génère un message d'erreur du genre :

The login/password combination you have entered is invalid

R_{iv} est l'ensemble des requêtes d'injection SQL syntaxiquement valides qui sont construites dans le but de générer des pages d'exécution. Par exemple, la requête suivante :

```
http://address/directory/page.php?login=test&pass='or'1'='1
```

génère la requête SQL suivante :

```
SELECT * FROM users WHERE login='test' and pass=''or'1'='1'
```

Cette requête SQL est syntaxiquement valide et le paramètre `"pass"` envoyé dans le navigateur avec la valeur : `' or '1'='1` permet de changer la sémantique de la requête SQL puisqu'il introduit une tautologie. L'authentification est alors acceptée quel que soit le login fourni. Par conséquent, une page d'exécution est bien retournée au client.

Notons que l'automatisation de notre approche réside sur le fait que nous soyons capables de générer automatiquement ces trois ensembles de requêtes. Nous reviendrons plus en détails sur ce point dans la section 3.3.

Nous notons S_a, S_{ii} et S_{iv} les réponses associées aux requêtes Ra, Rii et Riv respectivement.

Le principe de notre algorithme est alors le suivant :
"les requêtes Riv dont les réponses S_{iv} ne sont similaires à aucune des réponses S_{ii} et S_a, sont considérées comme des injections SQL réussies (i.e., la vulnérabilité a été exploitée avec succès)."

Pour évaluer la similarité entre les pages renvoyées par les différentes requêtes, nous utilisons une technique de classification basée sur le calcul de la distance entre les réponses. La section suivante présente cette distance.

3.2.2 La distance utilisée

Pour analyser la similarité entre deux pages HTML, nous avons besoin d'une distance permettant d'évaluer la différence entre deux chaînes de caractères. L'ordre des mots dans un texte peut avoir une grande importance. En effet, les mêmes mots dans un ordre différent peuvent complètement changer la sémantique de la réponse.
Il existe plusieurs distances permettant de mesurer la similarité entre deux chaînes de caractères. Généralement, le principe consiste à calculer la taille du plus petit script d'édition permettant de modifier la première chaîne pour obtenir la seconde en autorisant les opérations de remplacement, suppression et ajout de caractères. Un des problèmes à résoudre pour obtenir ce script est l'identification de la plus longue sous-chaîne commune entre deux chaînes. Ce problème peut être résolu par diffé-

$$\mathsf{diff}(a_i, b_j) = \begin{cases} n - i + m - j & i = n \text{ ou } j = m \\ \mathsf{diff}(a_{i+1}, b_{j+1}) & a_i = b_j, i < n, j < m \\ 1 + \min(\mathsf{diff}(a_{i+1}, b_j), \mathsf{diff}(a_i, b_{j+1})) & \\ & a_i \neq b_j, i < n, j < m \end{cases}$$

$$d(a, b) = \frac{\mathsf{diff}(a_1, b_1)}{(n + m)}$$

FIGURE 3.2 – Distance de classification

rentes approches, notamment par programmation dynamique. La distance de *Levenshtein* [2] en est un exemple[LEVENSHTEIN 66]. La commande `diff` [HUNT ET MCILROY 76] disponible sur tout système Unix est un autre exemple de programme qui résout ce problème. Une particularité de ce programme est qu'il considère les chaînes de caractères lignes par lignes et non caractère par caractère. L'opérateur de différence que nous utilisons est ainsi une version légèrement modifiée de la distance de *Levenshtein*. Il s'exprime formellement comme suit.

Soient a et b deux réponses de longueur n et m. Notons ai $(i = 1, \cdots, n)$ les lignes de la première réponse a, et bj $(j = 1, \cdots, m)$ les lignes de la seconde réponse b. La distance est formalisée dans la figure 3.2.2 :

3.2.3 Regroupement des pages en grappes et choix du seuil

Notre algorithme de classification des réponses va ainsi soumettre à l'application des requêtes issues des 3 ensembles présentés dans la section précédente et calculer la distance deux à deux entre toutes les réponses associées à ces requêtes. Ensuite, en fonction de la valeur de ces distances, l'algorithme va regrouper les requêtes afin d'obtenir différentes grappes. L'idée sous-jacente est de chercher une grappe ne contenant que des requêtes appartenant à l'ensemble R_{iv}. Cette grappe constitue alors les requêtes ayant réellement permis d'exploiter la vulnérabilité, que nous appelons requêtes d'injection réussie.

De façon générale, les techniques de regroupement s'appuient sur deux stratégies différentes. La première est guidée par le nombre de grappes que l'on désire obtenir. Il s'agit alors de partir d'une grappe unique contenant toutes les requêtes et de la diviser petit à petit, en s'appuyant sur les distances, jusqu'à obtenir le nombre de grappes souhaitées. La seconde est utilisée dans le cas où on ne connaît pas a priori ce nombre. Il s'agit alors de regrouper ensemble au sein d'une même grappe les requêtes

2. `http://en.wikipedia.org/wiki/Levenshtein_distance`

dont la distance deux à deux est inférieure à un seuil. Dans notre situation, le nombre de grappes n'est pas déterminé a priori et nous utilisons donc la seconde stratégie, appelé regroupement hiérarchique [JOHNSON 67] qui nécessite le choix d'un seuil.
Le seuil de regroupement des requêtes peut varier d'un point d'injection à l'autre. En effet, il dépend de la taille des réponses et de la quantité de données qui changent entre deux réponses associées à des requêtes du même type. Il varie également d'une application Web à une autre puisque leur mise en œuvre peut être très différente. Ce seuil doit donc être adapté à chaque application Web. Le principe fondamental est que ce seuil doit permettre de déterminer la valeur limite permettant de définir les pages qui se ressemblent (quelques caractères modifiés) et les pages qui ne se ressemblent pas (beaucoup de caractères modifiés).
Afin de respecter ce principe, notre choix a été de définir ce seuil par la plus petite distance entre i) la distance la plus longue entre deux réponses dans S_a et ii) la distance la plus longue entre deux réponses dans S_{ii}.

Plus formellement, ce seuil s et les grappes C sont définis ainsi :

$$s = \min(\max_{(a,b)\in S_{\mathrm{ii}}}(d(a,b)), \max_{(a,b)\in S_{\mathrm{is}}}(d(a,b)))$$

$$S = S_{\mathrm{ii}} \bigcup S_{\mathrm{vi}} \bigcup S_{\mathrm{is}} \bigcup S_{\mathrm{rr}}$$

$$\mathsf{path}(s_i, s_j) = \begin{cases} \exists s_k, d(s_i, s_k) \leq s, \mathsf{path}(s_k, s_j) & s_i \neq s_j \\ true & s_i = s_j \end{cases}$$

$$C = \{c_i\} \text{ est une partition de } S \; / \forall_{(s_i,s_j)\in c_i^2} \mathsf{path}(s_i, s_j)$$

A partir de cette classification et de ce seuil, nous sommes en mesure d'identifier des groupes de réponses qui sont similaires. En sachant qu'une requête est associée à une réponse et vice versa, nous pouvons donc identifier des groupes de requêtes qui sont similaires. Pour déterminer les requêtes d'injection réussie (qui sont un sous-ensemble des requêtes d'injection syntaxiquement valides R_{iv}), le principe est le suivant :

- Une requête R_{iv} faisant partie d'une grappe qui contient également une requête de R_a ou de R_{ii} n'est pas considérée comme une requête d'injection réussie
- Une requête faisant partie d'une grappe qui ne contient aucune requête de R_a et de R_{ii} est considérée comme une requête d'injection réussie.

En somme, si nous identifions une grappe contenant des réponses provenant uniquement de requêtes de l'ensemble R_{iv}, alors cette grappe identifie les requêtes d'injection réussie.
Ce principe est illustré dans la figure 3.3. Elle a été obtenue en utilisant l'algorithme de Kamada et Kawai [KAMADA 89] pour placer les points représentant les pages en fonction de la distance qui les

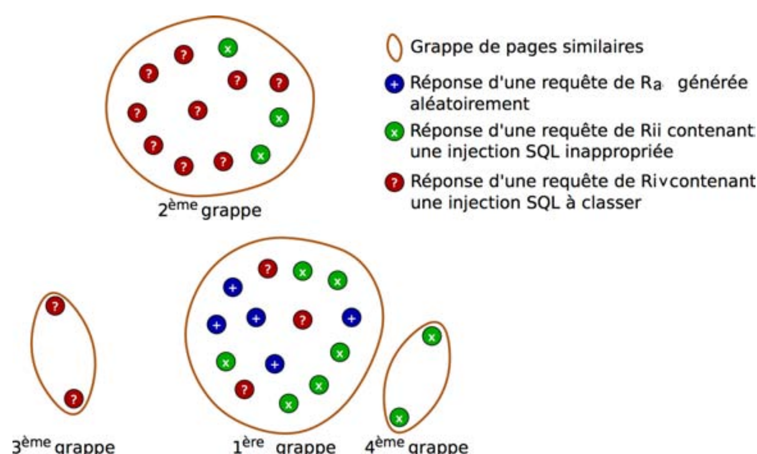

FIGURE 3.3 – Exemple de classification des réponses des requêtes

sépare et l'algorithme de Graham [GRAHAM 72] pour représenter l'enveloppe convexe de la grappe. Cette représentation présente l'avantage de bien faire apparaître visuellement les différentes grappes.

Dans cet exemple, la troisième grappe contient seulement des requêtes de R_{iv} : ce sont des requêtes d'injection réussie. La première grappe, en revanche, contient à la fois des requêtes de R_a, R_{ii} et R_{iv}. Les requêtes de R_{iv} de cette grappe ne sont donc pas considérées comme des requêtes d'injection réussie. Il est important de noter que cette grappe n'est pas la seule qui contient des requêtes qui engendrent des pages de rejet, ce qui signifie qu'il existe donc différents messages d'erreurs pour cette page du site. D'autres outils (comme *Skipfish*) peuvent considérer les pages incluses dans la seconde grappe comme un succès simplement parce qu'elles sont éloignées des requêtes de la première grappe (voir Chapitre 2). Cela mène à des faux positifs.

3.2.4 Exemple

Nous considérons comme exemple, un attaquant qui veut contourner un formulaire d'authentification en exploitant une vulnérabilité permettant d'effectuer une injection SQL. Nous avons utilisé comme cas d'étude l'application vulnérable *Insecure* développée en Ruby dans le cadre du projet DALI. Cette application, correspond à un site Web de commerce électronique. Elle est présentée dans le Chapitre 5.

Pour cet exemple, nous avons procédé comme suit :

- Envoi des 10 requêtes suivantes pour le serveur : A_{req}, B_{req}, C_{req}, D_{req}, E_{req}, F_{req}, G_{req}, H_{req}, I_{req}, J_{req}. Ces requêtes sont classées en trois ensembles :
  - $A_{req}, B_{req}, C_{req} \subseteq R_{ii}$: ensemble des injections invalides
  - $D_{req}, E_{req}, F_{req}, G_{req} \subseteq R_{iv}$: ensemble des injections valides
  - $H_{req}, I_{req}, J_{req} \subseteq R_a$: ensemble des injections aléatoires

reponses du serveur	A_{resp}	B_{resp}	C_{resp}	D_{resp}	E_{resp}	F_{resp}	G_{resp}	H_{resp}	I_{resp}	J_{resp}
A_{resp}	0.0	0.004	0.005	0.973	0.973	0.973	0.973	0.004	0.005	0.005
B_{resp}	0.004	0.0	0.0	0.952	0.952	0.953	0.952	0.0	0.0	0.004
C_{resp}	0.005	0.0	0.0	0.952	0.952	0.953	0.952	0.0	0.0	0.008
D_{resp}	0.973	0.952	0.952	0.0	0.003	0.007	0.002	0.952	0.952	0.952
E_{resp}	0.973	0.952	0.952	0.003	0.0	0.007	0.004	0.952	0.952	0.952
F_{resp}	0.973	0.953	0.953	0.007	0.007	0.0	0.006	0.953	0.953	0.952
G_{resp}	0.973	0.952	0.952	0.002	0.004	0.006	0.0	0.952	0.952	0.952
H_{resp}	0.004	0.0	0.0	0.952	0.952	0.953	0.952	0.0	0.0	0.003
I_{resp}	0.005	0.0	0.0	0.952	0.952	0.953	0.952	0.0	0.0	0.006
J_{resp}	0.005	0.004	0.008	0.952	0.952	0.952	0.952	0.003	0.006	0.0

TABLE 3.1 – Calcul de distance entre les réponses

- Le serveur répond respectivement par les 10 réponses suivantes : A_{resp}, B_{resp}, C_{resp}, D_{resp}, E_{resp}, F_{resp}, G_{resp}, H_{resp}, I_{resp}, J_{resp}.
- Les grappes sont créées par le calcul des distances 2 à 2 entre toutes les réponses, en rappelant que la distance entre A et B est calculée selon la formule indiquée dans la figure 3.2. Les distances ainsi obtenues sont présentées dans le tableau 3.1.

Deux pages HTML sont identiques quand la distance entre elles est égale à 0.0. Dans ce tableau, nous avons arrondi toutes les valeurs à trois décimales. Les grappes sont créées lorsque la distance entre les pages est au-dessus d'un seuil fixé à 0,01.
Nous avons utilisé le logiciel R pour générer une représentation visuelle des grappes obtenues, appelée "Dendrogramme", qui est illustrée sur la figure 3.4.

Dans la figure 3.4, `height` est le seuil de classification. Ce nombre doit être adapté à chaque application Web. Par exemple, avec une valeur à 0.2, nous obtenons deux clusters (représentés par le nombre de lignes verticales en face de la valeur du seuil), mais une valeur de seuil à 0,99 entraîne un seul cluster. Le choix du seuil est important car s'il est trop bas, autant de grappes que de requêtes vont être créées et beaucoup de requêtes vont être considérées à tort comme des attaques réussies. S'il est trop élevé, toutes les requêtes vont faire partie d'une même et unique grappe et aucune requête ne sera considérée comme une attaque réussie. Ces deux cas extrêmes sont problématiques car ils génèrent trop de faux positifs pour l'un et trop de faux négatifs pour l'autre. Il n'y a pas de bonne valeur de seuil dans l'absolu, il est à déterminer en fonction du site, des pages qu'il contient. Néanmoins, comme notre objectif est tout d'abord de ne pas générer de faux positifs, un seuil proche de 1 est meilleur. C'est pourquoi le seuil que nous avons proposé dans notre approche (cf. section 3.2.3) nous semble un bon

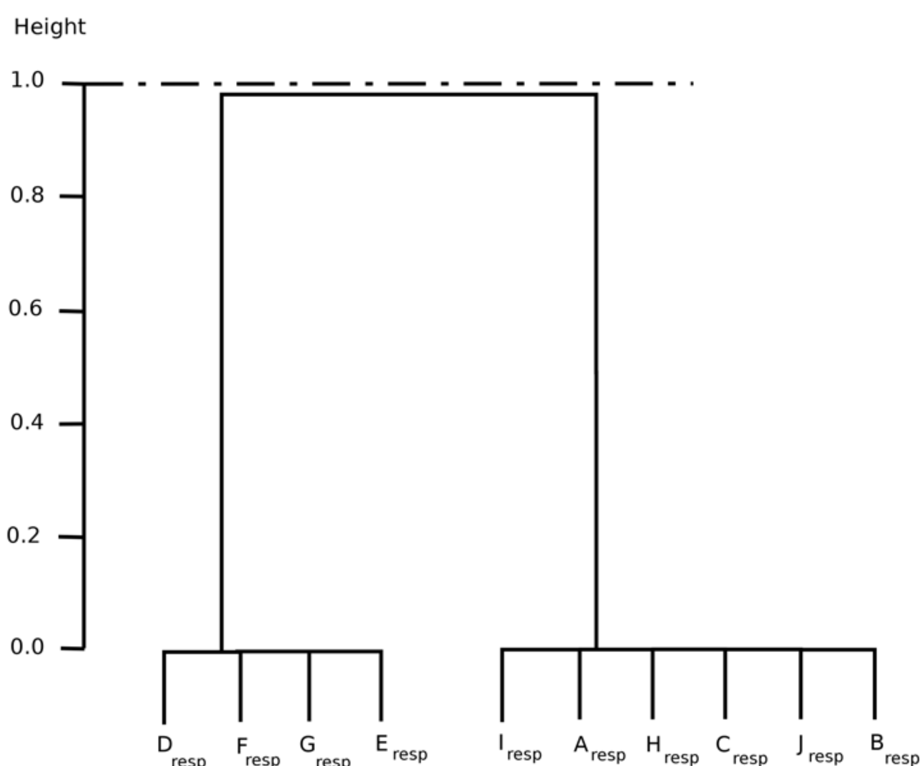

FIGURE 3.4 – Dendrogramme de grappes

compromis dans la mesure où il suit cette logique et permet de choisir le seuil le plus proche possible de 1 qui ne génère pas de faux positifs.

Dans l'exemple fourni ici, la page que nous testons est une page d'authentification, pour laquelle les réponses correspondant aux pages de rejet et aux pages d'exécution sont très différentes en terme de contenu. Par conséquent, même un seuil assez faible est satisfaisant pour obtenir la grappe qui fournit les requêtes d'injection réussie. Ainsi, pour un seuil situé entre 0.01 et 0.99, nous obtenons deux grappes.
La première grappe, constituée des requêtes D_{req}, E_{req}, F_{req}, G_{req}, ne contient que des requêtes syntaxiquement valides (R_{iv}). La seconde grappe, quant à elle, contient des requêtes provenant des 3 ensembles de requêtes R_a, R_{iv} et R_{ii}. Par conséquent, la première grappe nous permet donc d'identifier les requêtes d'injection réussie et la seconde grappe contient les autres requêtes, qui n'ont pas réussi à exploiter la vulnérabilité (qu'elles soient aléatoires, syntaxiquement valides ou invalides).

3.3 Génération des requêtes : utilisation des grammaires

Un objectif important de l'algorithme proposé est d'identifier la présence d'une vulnérabilité dans un point d'injection en fonction des réponses multiples générées à partir de ce point d'injection.

Pour améliorer la précision des résultats, il est nécessaire de générer un nombre suffisamment grand de requêtes permettant d'assurer une couverture élevée des différentes réponses pouvant être renvoyées par le serveur Web pour chaque point d'injection considéré. Notons que d'autres approches génèrent un nombre réduit de requêtes (par exemple, 3 pour *Skipfish*), ce qui peut donner lieu à des faux positifs, ce que nous voulons éviter dans notre approche.

Une manière possible de générer un nombre significatif de réponses (et les requêtes associées) est d'enregistrer dans un fichier statique, toutes les requêtes que l'on souhaite envoyer. En particulier, pour les requêtes de la catégorie R_{iv}, on peut se baser sur les connaissances provenant d'experts en sécurité ou disponibles sur Internet (par exemple, utiliser les "SQL Sheets" [KIEZUN ET AL. 09]). Cette approche présente l'avantage de pouvoir générer des requêtes particulièrement subtiles puisqu'elles sont produites par de réels experts du domaine. Son inconvénient majeur est qu'elle est statique et fastidieuse puisqu'il faut saisir une à une toutes les requêtes que l'on veut envoyer. En particulier, si l'on veut pouvoir envoyer des variantes d'un certain type de requêtes, il est nécessaire de toutes les imaginer et les écrire. Il en est de même pour les requêtes syntaxiquement invalides et les requêtes aléatoires.
Une approche plus flexible consiste à définir une grammaire ou un modèle pour automatiser ce processus. Cette approche comporte elle aussi des avantages et des inconvénients. Un des inconvénients est que l'écriture d'une grammaire ou d'un modèle est une opération pouvant s'avérer complexe, coûteuse, et qui nécessite de disposer d'une spécification du langage concerné (SQL par exemple dans notre cas). Ce modèle ou cette grammaire peut de plus s'avérer inadapté si l'implémentation du langage ne respecte pas scrupuleusement les spécifications ou si l'application testée n'utilise qu'une partie de ce langage (par exemple, une application PHP qui n'utilise que les requêtes SELECT, qui ne représentent qu'une petite partie du langage SQL). D'un autre côté, cette approche présente des avantages certains. Elle permet de générer de façon complètement automatique un très grand nombre de requêtes. Dans le cas du langage SQL qui nous intéresse ici en particulier, la grammaire nous permet de générer une multitude de variantes d'un certain type d'injection. Ceci est très important dans notre approche car l'efficacité de l'algorithme de clustering repose sur le fait que l'on soit capable de générer un grand nombre de requêtes. Une grammaire peut également facilement être mise à jour si on envisage un nouveau type d'injection que l'on n'avait pas considéré jusqu'à présent. L'avantage de la grammaire est qu'il suffit d'ajouter une règle pour pouvoir automatiquement générer une multitude de variantes de ce nouveau type d'injection. Nous en donnons des exemples dans les sous-sections suivantes.

3.3.1 Requêtes syntaxiquement valides : R_{iv}

Du côté du serveur Web, la plupart du temps, une requête SQL est forgée par la concaténation de termes SQL et des paramètres envoyés par le client. Par exemple, le script PHP suivant traite l'authentification d'un utilisateur en donnant le nom d'utilisateur et mot de passe envoyés par le client :

```
$query  = "SELECT id FROM users WHERE name='$name' AND pass='$pass'";
```

Étant donné un couple correct (identifiant, mot de passe), la requête SQL forgée est considérée syntaxiquement correcte. En outre, la requête forgée associée au couple (identifiant, mot de passe) basée

sur une attaque par dictionnaire est considérée comme syntaxiquement correcte, même si l'authentification a échoué.
A partir de cette d'observation, une injection SQL est définie comme une chaîne de caractère qui peut changer la sémantique de la requête SQL générée. Dans de nombreuses situations, une injection SQL est intéressante si elle conduit à une tautologie dans le clause WHERE de la requête SQL forgée. Dans l'exemple précédent, on peut considérer l'injection SQL suivante :

```
name="' OR 1=1 OR string='"
```

Par conséquent, la grammaire de l'injection SQL est juste une partie de la grammaire de la requête SQL. L'avantage de cette grammaire est qu'elle permet de générer facilement le nombre d'injections SQL dont nous avons besoin. Nous pouvons appliquer le même raisonnement à l'ensemble des mots générés aléatoirement R_a et à l'ensemble R_{ii} des injections SQL qui sont syntaxiquement invalides pour le point d'injection donné.
Voici, à titre d'exemple, une première version d'une grammaire pour l'ensemble de requêtes R_{iv}, exprimée en utilisant la notation *BNF* [3] :

```
I0 := String "' " I1  Field Operator   " '" String %d10 %d13

I1 := ")" I1 "(" / I2

I2 := AndOr / AndOr I3 AndOr

I3 := Not "(" I3 ")" / I4

I4 := Field Operator Field / Field Operator Field I1 Field Operator Field

String := *%d48-95

Field := " '" String "' "

Operator := "=" / "<" / ">"

AndOr := " or " / " and "

Not := " not " / ""
```

Cette grammaire génère différentes variantes d'attaques par injection SQL dont le principe consiste à insérer une tautologie dans une expression évaluée par une clause WHERE, de telle manière que cette expression devienne une tautologie elle-même.
Sans perte de généralité, nous supposons que l'injection est réalisée à droite d'un opérateur (pour généraliser la grammaire, il suffit de prendre le symétrique). Pour réaliser une injection sur une zone entourée d'apostrophe, il faut forcer la fermeture de l'apostrophe, injecter la tautologie et ré-ouvrir l'apostrophe pour que la requête SQL générée soit syntaxiquement correcte (règle I0). Notons que dans cette règle, %d10 et %d13 représentent les caractères *Carriage Return* et *Line Feed*. Pour la création de la tautologie – à la manière de la stratégie adoptée pour l'apostrophe – nous pouvons avoir besoin de forcer la

3. BNF : Backus normal Form : une notation pour écrire la grammaire

fermeture des parenthèses (règle I1). A chaque fermeture doit également correspondre une réouverture pour les mêmes raisons que précédemment. L'expression à injecter peut être composée d'une séquence de AND et de OU d'expressions parenthèses (règles I2, I3 et I4). De manière générale, ces règles ont été construites en s'assurant de l'équilibre dans le nombre de parenthèses ouvrantes et fermantes et d'apostrophes.
Cette grammaire permet de générer par exemple des injections de ce type :

```
    '  OR  '' = '
A' ) Or ( '' > '<K
' ) and ( '' < 'P0
:' ))))) aNd  '' > ''  And  '' > ''  oR  '' < 'E'  aND ((((( '^P[' > 'T
'  aNd  '>' < '
'  or  'I' > ';
<[C' ) aNd ( '' > '
' ) and  'I' > 'LUM'  AND ( 'PV' < 'JV
B' ) OR ( '' = '
'  OR  'K' > ''  And  '' < 'DU
```

Cette première version ne couvre pas toutes les injections listées dans les SQL Sheet traditionnellement proposées par les experts, mais elle peut être enrichie au fur et à mesure selon les besoins pour couvrir de plus en plus d'injections. Imaginons par exemple, qu'il soit nécessaire d'utiliser une clause `UNION` pour générer une injection SQL efficace pour une application Web particulière. Imaginons que cette injection ait la forme suivante :

```
' UNION SELECT 'klj','qdksfj','ma' WHERE 'mlqskdjf'='
```

Dans ce cas, il nous suffit simplement d'ajouter une règle permettant ce type d'injection dans notre grammaire, qui devient ainsi dans sa seconde version :

```
I0 := String "' " I1  Field Operator   " '" String %d10 %d13
I1 := ")" I1 "(" / I2
I2 := AndOr / AndOr I3 AndOr / "union select " Liste " where "
I3 := Not "(" I3 ")" / I4
I4 := Field Operator Field / Field Operator Field I1 Field Operator Field
Liste := "'" String "'" / "'" String "'," Liste
String := *%d48-95
```

```
Field := " '" String "' "

Operator := "=" / "<" / ">"

AndOr := " or " / " and "

Not := " not " / ""
```

Grâce à la modification de la troisième règle, cette nouvelle grammaire nous permet de générer cette nouvelle injection mais nous permet surtout de générer une multitude de variantes de ce type d'injection et c'est en ce sens que l'utilisation d'une grammaire est particulièrement utile.
Cette grammaire permet de générer par exemple ce type de requêtes :

```
' )))uniON SelEct  '67417UJ','W]' WHeRE  ((( 'RE' > '

K' ) OR ( ':' < '

' Union SElecT  '' whEre    '\^' < '1

YBDH'  AND  ';' = '

' ))) and  ( NOT ( '' < '1EF;U' )) or  ((( '' > ']

6'  ANd  '[U<' < '

N58DR' )UNion  selECt  '' whERe  ( '' < '9

'  and  ( ':WZFW' > '' ) ANd   '' = '

\' )uNIOn  selecT  '' whEre  ( '' = 'S1RXW

' )) or  (( '' > '
```

3.3.2 Requêtes aléatoires : R_a

La grammaire permettant de générer des requêtes aléatoires se limite à une seule règle :

```
STR = *%d48-95
```

Cette grammaire est très simple puisqu'elle ne comporte dans sa règle que des caractères alphanumériques, dont les codes ASCII sont compris entre 48 et 95. Elle génère donc des chaînes de caractères aléatoires composées de ces caractères.

3.3.3 Requêtes syntaxiquement invalides : R_{ii}

La grammaire permettant de générer des requêtes syntaxiquement invalides est constituée des règles suivantes :

```
I0 = STR I1 F O "'" STR / STR "'" I1 F O STR

I1 = I1 "(" / ")" I1 / I2
```

```
I2 = A / A I3 / I3 A

I3 = N "(" I3 / N I3 ")" / I4

I4 = F O F / F O F I1 F O F

STR = *%d48-95

F = " '" STR "' "

O = "=" / "<" / ">"

A = " or " / " and "

N = " not " / ""
```

Cette grammaire va systématiquement fournir des requêtes d'injection SQL invalides car, quelle que soit l'injection qu'elle génère, le nombre d'apostrophes ou de parenthèses est impair (il est donc incorrect). Par exemple, la première règle indique que l'injection peut être réalisée en fermant une expression entre apostrophes puis en injectant une expression booléenne, mais sans ajouter d'apostrophe à la fin. L'équilibre du nombre d'apostrophes n'est donc pas assuré par cette règle.
Voici, à titre d'exemple, quelques requêtes qui peuvent être générées par cette grammaire :

```
NoT (( 'MX' > '\LC'  aND  NOt  not ( nOT ( '' > '6>0HK' )) '' > '6I' ) aND ( ''>'

'T' > ''  aND  'H' ='

L7') 'V' > '0]^3'  Or ( '' <BV

) NOt ( noT  nOt ( nOt ( nOt  '' < '8' ) aNd ((( '>' > '' ))) Or (((( '' >'T

TRG?') or (( '' <H

)) 'L' > 'Y??' ))) noT ( '3' = ''  ANd ( '' = 'L'  Or ( 'RD' >'UX

' or  '' =5

<I\X' 'S' = 'Y'  Or  '' =M

'' = ''  aNd  'X' <'

D4') or  '' <J_
```

3.4 Extension à d'autres classes de vulnérabilités

Le principe de la détection de la vulnérabilité d'injection SQL peut être généralisé. En effet, de nombreuses attaques adoptent le même comportement : le client envoie une chaîne de caractères qui change la sémantique de la requête forgée. Selon le contexte, la requête forgée est envoyée à un composant spécifique côté serveur Web tel que le moteur XPATH, le système d'exploitation, etc. Les noms des attaques par injection correspondantes sont dérivés du nom de cette composante menant à l'injection : injection XPATH, Os Commanding, etc. Ainsi, l'algorithme de clustering que nous avons illustré en

prenant l'exemple de l'injection SQL dans la section précédente peut aussi être utilisé pour ces types de vulnérabilités.
Comme expliqué précédemment, l'algorithme utilise trois ensembles de requêtes : R_a (requêtes aléatoires), R_{ii} (injections syntaxiquement invalides) et R_{vi} (injections syntaxiquement valides). L'adaptation de l'algorithme à d'autres types de vulnérabilités ne nécessite que la définition de ces trois ensembles pour chaque type de vulnérabilité. Une fois que ces ensembles sont établis, l'algorithme procède de la même manière : envoyer ces requêtes, stocker les résultats correspondants et obtenir des grappes grâce à la distance que nous avons présentée dans la section 3.2.2.

Dans la suite nous analysons quelques exemples qui illustrent comment notre algorithme peut être appliqué pour la détection de vulnérabilité de type XPATH, Os Commanding et File Include. Pour chaque cas, nous expliquons brièvement le principe de l'exploitation et nous donnons un exemple de requête appartenant aux ensembles R_{iv} et R_{ii}.

3.4.1 Xpath

La vulnérabilité XPATH correspond, comme la vulnérabilité SQL, à l'entrée d'un formulaire HTML ou des paramètres d'URL non assainis, à la différence que cette vulnérabilité peut être exploitée pour exécuter des requêtes XPATH et non les requêtes SQL.
Un code source Python vulnérable peut ressembler à ce qui suit :

```
query="//user[name/text()=('"+login+"')and password/text()=('"+login+"')]/account/text()"

res = ctxt.xpathEval(query)
```

`xpathEval` est une fonction qui exécute une requête XPATH, notée `query` dans le code source. Cette fonction est le résultat de la concaténation de la variable `login`, la variable `password` et quelques autres chaînes de caractères. Comme ces variables ne sont pas assainies, le code est alors vulnérable. Ainsi soumettant des entrées intentionnellement mal formées au site Web, un attaquant peut savoir comment les données XML sont structurées, ou accéder à des données auxquelles il ne peut légitimement pas avoir accès. Il peut même être en mesure d'élever ses privilèges sur le site Web si les données XML sont utilisées pour l'authentification (par exemple un fichier utilisateur basé sur XML).

Deux requêtes syntaxiquement invalide et valide peuvent respectivement ressembler à :

```
Rii : fzx' or hew('pjq')='vrs

Riv : xsw' or string-length('sF')=2
```

La première injection est invalide parce que la fonction `hew` n'existe pas. La seconde est construite pour être correcte et est basée sur la tautologie : `string-length('sF')=2`, qui signifie que la longueur de la chaîne `SF` est égale à 2.

3.4.2 Os Commanding

Dans le cas de la vulnérabilité OS Commanding, la chaîne envoyée par le client est utilisée pour créer une commande exécutée par le système d'exploitation. Cette commande est exécutée sous l'identité du processus correspondant au serveur Web. L'exploitation de cette vulnérabilité permet à un attaquant d'exécuter des commandes du système arbitraires et peut permettre également l'accès en lecture ou/et en écriture à certains fichiers. Toutes ces commandes s'exécutant sous l'identité du processus correspondant au serveur Web, les droits d'accès seront évalués en fonction de cette identité. La plupart des langages de script permettent aux programmeurs d'exécuter des commandes du système d'exploitation, en utilisant diverses fonctions `exec`. Si l'application Web permet aux entrées fournies par l'utilisateur d'être utilisées en tant que paramètre d'un appel de fonction sans être assaini au préalable, il peut être possible pour un attaquant d'exécuter des commandes du système d'exploitation à distance.

Voici un premier exemple de code PHP vulnérable :

```
<?php
system("cat ".$_GET['cmd']);
?>
```

Ce code exécute la commande `cat` avec des arguments extraits du paramètre `cmd`. Comme ce paramètre n'est pas assaini, le code est vulnérable.
Un deuxième exemple, dans ce script PHP, liste le contenu d'un répertoire système (sur les systèmes Unix), à l'aide de l'exécution d'une commande shell :

```
exec ("ls-la \$ dir", \$ lignes, \$ rc);
```

La variable `dir` n'étant pas assainie, il suffit de lui donner comme valeur un point-virgule (;) suivi d'une commande du système d'exploitation. Cette dernière commande sera ainsi exécutée par l'application Web. Par exemple, l'injection :

```
http://example/directory.php?dir=\%3Bcat\%20/etc/passwd
```

permet d'afficher le contenu du fichier `/etc/passwd`. De même que pour les injections SQL et PATH, il est nécessaire de générer les ensembles d'injections valides et invalides pour appliquer notre algorithme. Deux exemples de requêtes syntaxiquement invalide et valide peuvent respectivement ressembler à :

```
Rii :.;. LeSS /OFr/BxEBHM . cAt /sCs/wJjqCg
Riv :| more /././etc/././passwd
```

La première requête contient des commandes Unix invalides. La seconde contient un pipe (|) vers une commande valide qui est censée afficher le contenu du fichier `/etc/passwd`, comme nous l'avons expliqué dans l'exemple précédent.

3.4.3 File Include

La vulnérabilité File Include fait référence à une instruction de divers langages de programmation qui permet d'inclure un fichier (include en PHP, d'où le nom de la vulnérabilité). L'inclusion de fichier est à priori destinée à provoquer l'inclusion d'un autre fichier PHP. Mais si cette inclusion est faite par l'intermédiaire d'une variable et que cette dernière n'est pas assainie, il est possible de provoquer l'inclusion d'un autre fichier. Ce dernier peut ensuite être retourné à l'attaquant dans la page HTML générée. Cette vulnérabilité est dangereuse parce qu'elle peut permettre à un attaquant de lire le contenu de certains fichiers système critiques tels que `/etc/passwd` (pour Unix OS).
Un code source vulnérable peut ressembler à :

```
<?php
include("include/".\$_GET['file'].".php");
?>
```

Ce code cherche à inclure un fichier dont le nom est défini dans le paramètre file. Comme ce paramètre n'est pas assaini, le code est vulnérable.
Deux requêtes syntaxiquement invalide et valide peuvent respectivement ressembler à :

```
Rii : /wrong\_directory/bad\_file
Rvi : ../../../../etc/././passwd
```

La première requête est invalide car elle contient un chemin de fichier qui, très probablement, n'existe pas. La deuxième requête est valide et est censée provoquer l'inclusion du fichier `/etc/passwd`.

3.5 Expérimentations

Cette section présente les expérimentations que nous avons réalisées afin de valider et d'évaluer notre algorithme de détection de vulnérabilités [DESSIATNIKOFF ET AL. 11A]. Nous avons considéré plusieurs applications en utilisant les scanners de vulnérabilités discutés dans le chapitre 2, *W3af 1.1*, *Skipfish 1.9.6b*, *Wapiti 2.2.1* et notre propre scanner de vulnérabilité *Wasapy* décrit dans la section 3.1. Notons qu'il n'y a pas eu de configuration particulière pour les différents outils utilisés sauf pour *Wasapy*, pour lequel nous avons défini le nombre de requêtes injectées par point d'injection à 30 pour chacune des classes de vulnérabilités testées.

Les expériences ont été exécutées sur une machine équipée du système *GNU/Linux* (noyau 2.6) exécutant plusieurs machines virtuelles grâce à l'utilitaire *VirtualBox*. Toutes les machines virtuelles exécutent le serveur web *Apache* 1.3.37 ou 2.2.8 avec *PHP* 4.0 ou 5,0 et *MySQL* 5 comme serveur de base de données.

Nous avons réalisé deux séries d'expériences. Dans la première série d'expériences, nous avons injecté volontairement et manuellement quelques vulnérabilités spécifiques dans cinq applications open source et analysé les capacités de détection de notre algorithme par rapport aux trois outils explorés précédemment. Dans la deuxième série d'expériences, nous avons examiné cinq autres applications vulnérables sans les modifier. Ces expériences nous ont permis d'illustrer les avantages potentiels de notre algorithme.

3.5.1 Notation

Les résultats de nos expériences sont présentés dans différents tableaux. Par souci de lisibilité, nous utilisons les notations et abréviations suivantes :

✓	La vulnérabilité a été détectée par le scanneur correspondant
✗	La vulnérabilité n'a pas été détectée par le scanneur correspondant
–	Le point d'injection n'a pas été testé par le scanner
SQL*i*	`Injection SQL`
XPa	`Injection XPATH`
OsC	`Os Commanding`
FIn	`File Include`
CVE	La référence CVE de la vulnérabilité considérée si elle existe
NR	La vulnérabilité correspondante n'a pas de référence CVE. Donc, soit elle a été détectée par un des outils testés, et pour vérifier qu'il ne s'agit pas d'un faux positif, nous l'avons testée manuellement, soit nous l'avons notée de la référence MOTH[4] sur laquelle nous nous sommes basés pour certaines expérimentations.

- Une vulnérabilité est considérée comme détectée si le scanner génère une alerte pour cette vulnérabilité, quelle que soit la méthode utilisée pour la détecter.
- Une vulnérabilité est considérée comme non détectée si le scanner teste le point d'injection correspondant sans envoyer une alerte.
- Une vulnérabilité est considérée comme ignorée par le scanner si le point d'injection correspondant n'a pas été testé par le scanner.

Dans la suite nous considérons deux types d'applications avec lesquelles nous avons réalisé des expérimentations : des applications modifiées et des applications non modifiées.

3.5.2 Expériences avec les applications modifiées

3.5.2.1 Présentation des applications modifiées

Les cinq applications choisies pour cette première série d'expériences sont décrites dans la suite :

phpBB-3 [5]
Cette application est un gestionnaire de forum écrit en *PHP* 4 et utilisant une base de données *MySQL* version 4.x.x. Nous avons modifié le formulaire d'authentification de l'application de sorte qu'il inclut la vulnérabilité (`v1`) qui peut être exploitée par une injection SQL. Cette vulnérabilité permet à un attaquant d'atteindre l'accès restreint à l'espace administrateur du forum.

Secure Page [6]
Cette application écrite en *PHP*, est conçue pour protéger l'accès d'un site Web grâce à l'authentification. Les couples valides pour cette authentification sont stockés dans une base de données *Mysql*. Une vulnérabilité (`v2`) similaire à (`v1`) a été volontairement injectée.

Hardware Store
Nous avons nous-mêmes développé cette application en *PHP* 5.0. Cette application permet à un utilisateur de faire l'inventaire d'équipements informatiques dans une base de données et d'interroger cette base de données. L'utilisateur doit d'abord s'authentifier.
Cinq vulnérabilités SQL ont été volontairement injectées dans cette application :

– La vulnérabilité (`v3`) : permet l'injection du code SQL dans un formulaire de recherche, et permet à un attaquant d'accéder à toute la base de données.
– La vulnérabilité (`v4`) : permet l'injection du code SQL dans le formulaire de l'authentification.
– La vulnérabilité (`v5`) : permet l'injection du code SQL dans un paramètre d'une requête HTML. Pour cette page HTML vulnérable, nous avons volontairement désactivé le message d'erreur renvoyé par le serveur, afin de comparer le comportement de W3af et Wapiti avec celui de *Wasapy* dans une telle situation.
– La vulnérabilité (`v6`) : est similaire à la vulnérabilité (`v4`), mais elle est utilisée dans un contexte différent : le message d'erreur renvoyé par le serveur est là-aussi, comme pour (`v5`), désactivé.
– La vulnérabilité (`v7`) : est particulière dans le sens où elle ne peut être exploitée qu'après l'exploitation réussie de la vulnérabilité (`v4`). En effet, cette vulnérabilité est incluse dans une page qui ne peut être consultée qu'après une authentification réussie sur l'application Web ou après un succès de contournement du mécanisme d'authentification (par l'exploitation de la vulnérabilité (`v4`)).

5. `http://www.phpbb.com`
6. `http://www.01php.com/fiche-scripts-126.html`

Les vulnérabilités de types XPATH, OS Commanding et File Include ont également été injectées dans cette application.

- La vulnérabilité (`v10`) : permet à un attaquant de contourner l'authentification grâce à une injection XPATH dans la page d'authentification.
- La vulnérabilité (`v11`) : est une vulnérabilité Os commanding qui ne peut être exploitée qu'après exploitation de la vulnérabilité (`v4`). En effet, cette vulnérabilité est incluse dans une page qui est uniquement accessible après authentification (ou contournement de l'authentification grâce au succès de l'exploitation de (`v4`)).
- La vulnérabilité (`v12`) : est une vulnérabilité File Include, elle est incluse dans la même page que (`v11`) et peut être exploitée dans les mêmes conditions que (`v11`).

Insecure
Cette application a été développée en Ruby on Rails dans le cadre du projet DALI. Il s'agit d'un site de commerce électronique où plusieurs vulnérabilités ont été injectées volontairement. Une vulnérabilité (`v8`), qui permet à un attaquant d'injecter du code SQL, a été délibérément incluse dans le formulaire d'authentification de l'application. Cette vulnérabilité, équivalente fonctionnellement à (`v4`), est différente parce que *Insecure* est implémentée en Ruby et les messages d'erreur sont différents de ceux d'Apache.

Damn Vulnerable Web Application (DVWA) [7]
Cette application est écrite en PHP et utilise un serveur *MySQL*. Une vulnérabilité (`v9`), similaire à (`v3`), a été volontairement introduite dans l'application.

3.5.2.2 Résultats

Le tableau 3.2 présente les résultats de détection des vulnérabilités considérées par les différents outils. On peut observer que les performances de *W3af* et *Wapiti* sont similaires en moyenne, même si les vulnérabilités détectées ne sont pas les mêmes (Wapiti détecte avec succès `v1` et `v2` alors que *W3af* ne les détecte pas ; d'autre part, *W3af* détecte `v4` et `v8` alors que Wapiti ne les détecte pas). Ce résultat est conforme avec le fait que les deux scanners utilisent un algorithme de reconnaissance de messages d'erreurs. Les variations observées sont liées à la génération de différentes requêtes par ces outils, tel que discuté dans le chapitre 2. *Wasapy*, quant à lui, permet de détecter toutes ces vulnérabilités. Cela confirme que l'algorithme de classification et de regroupement de pages Web pour la détection de vulnérabilités présente une meilleure couverture que les outils basés sur la reconnaissance de messages d'erreur pour ces classes de vulnérabilité.

En ce qui concerne les vulnérabilités `v1` et `v2`, nous avons vérifié manuellement les injections réalisées par Skipfish (' ", ' " et ' ") et stocké les réponses correspondantes (respectivement A, B et C). Skipfish considère que les pages A et C doivent être différentes de sorte que la vulnérabilité soit présente. Malheureusement, pour ces deux points d'injection, ce n'est pas le cas. Les messages

7. `http://www.dvwa.co.uk`

Vulnérabilités			Scanners			
Type	Application	*ID*	Skipfish	W3af	Wapiti	Wasapy
SQL*i*	`phpBB3`	v1	✗	✗	✓	✓
	`SecurePages`	v2	✗	✗	✓	✓
	`HardwareStore`	v3	✓	✓	✓	✓
		v4	✓	✓	✗	✓
		v5	✗	✗	✗	✓
		v6	✗	✗	✗	✓
		v7	–	–	–	✓
	`Insecure`	v8	✓	✓	✗	✓
	`DVWA`	v9	✓	✓	–	✓
XPa	`HardwareStore`	v10	✗	✗	✗	✓
OsC	`HardwareStore`	v11	–	–	–	✓
FIn	`HardwareStore`	v12	–	–	–	✓
Nombre de détections			5	4	3	12

TABLE 3.2 – Résultats de détection de vulnérabilités pour les applications modifiées

d'erreur SQL renvoyés par le serveur sont très similaires. Par conséquent, Skipfish ne peut pas détecter ces vulnérabilités.

En ce qui concerne les vulnérabilités `v5` et `v6`, elles sont incluses dans des pages PHP pour lesquelles nous avons volontairement désactivé la fonction de notification de message d'erreur dans le fichier de configuration de PHP5. Dans ce cas particulier, aucun des trois scanners (*Skipfish*, *W3af* et *Wapiti*) n'a été capable de détecter les vulnérabilités.

En ce qui concerne la vulnérabilité `v7`, *Wasapy* est le seul scanner qui est capable de la détecter. Par ailleurs, c'est le seul qui a été capable d'identifier le point d'injection correspondant. En effet, ce point d'injection est inclus dans une page HTML qui n'est accessible qu'après une authentification réussie ou après l'exploitation réussie de la vulnérabilité `v4`. *Wasapy* est le seul à pouvoir exploiter automatiquement `v4`, et d'accéder à la page comprenant la vulnérabilité `v7`. Pour les autres scanners, il est nécessaire d'effectuer une exploitation manuelle de `v4` afin de pouvoir accéder à cette page.

Les vulnérabilités `v11` et `v12` ont été identifiées seulement par notre outil pour les mêmes raisons : elles restent masquées jusqu'à ce que l'authentification soit contournée.

Le but de ces premiers tests était l'étalonnage de *Wasapy*. Ils nous ont notamment permis de tester les grammaires que nous avons élaborées pour la génération automatique de requêtes. Bien sûr, les vulnérabilités correspondantes ont été identifiées dans ce but. Ainsi, ces résultats ne sont pas destinés à être utilisés pour faire une comparaison absolue entre les scanners.
Une évaluation plus représentative des différents outils doit être réalisée sur des applications dans lesquelles les vulnérabilités n'ont pas été injectées par nous-mêmes. Ces expériences sont présentées dans la section suivante.

3.5.3 Expériences avec les applications non modifiées

Cette deuxième série d'expériences nous a permis d'avoir une idée plus précise de la couverture de notre algorithme de détection. À cette fin, nous l'avons comparé aux algorithmes de détection de *Skipfish*, *W3af* et *Wapiti* en considérant des applications Web vulnérables que nous n'avons pas modifiées. Pour certaines de ces applications, nous avons pu comparer notre algorithme avec certains scanners de vulnérabilités commerciaux, en considérant des résultats disponibles dans MOTH. Dans ce document, l'auteur présente les résultats de détection des vulnérabilités obtenus avec trois scanners commerciaux : *WebInspect* [8] de HP, *AppScan* [9] d'IBM et *Web Vulnerability Scanner*(WVS) [10] de Acunetix.

3.5.3.1 Présentation des applications non modifiées

Pour nos expériences, nous avons sélectionné cinq applications Web (la plupart d'entre elles testées dans MOTH, connues pour contenir des vulnérabilités). Ces applications couvrent différentes fonctionnalités et des contextes différents.
Nous avons installé ces applications, et effectué des tests de détection de vulnérabilité sans les modifier.

Cyphor [11]
Cette application implémente un forum de discussion, elle se base sur la notion de session dans PHP 4 pour authentifier les utilisateurs et sur une base de données MySQL pour stocker les données de l'utilisateur.
Seuls les utilisateurs enregistrés peuvent poster des messages. Ces utilisateurs sont répartis en utilisateurs normaux, les modérateurs et administrateurs. Les modérateurs et administrateurs peuvent supprimer les discussions, les administrateurs peuvent modifier les paramètres d'administration, créer de nouveaux forums, etc.

8. `http://www.web-inspect.com`
9. `http://www-01.ibm.com/software/awdtools/appscan/`
10. `http://www.acunetix.com/vulnerability-scanner/`
11. `http://webscripts.softpedia.com/script/Snippets/Cyphor-27985.html`

Seagull [12]

Cette application est un framework orienté objet (OOP : Object-oriented programming) pour développer des applications Web, des sites de commerce en ligne et des interfaces graphiques. Elle permet aux développeurs PHP d'intégrer et de gérer leur code source. Cette application nécessite la configuration suivante : *PHP* 4.3.0 ou plus récent, *MySQL* 4.0.x ou plus récent, `Apache` 1.3.x ou 2.x.

Fttss [13]

Cette application a été développée dans le cadre d'un projet de recherche qui porte sur la mise en œuvre d'un système Text-To-Speech permettant de transformer un texte écrit en un texte parlé. *PHP* (4.3 ou plus récent) et *MySQL* (4.1.2 ou plus récent) sont nécessaires pour l'exécution de cette application.

Riotpix [14]

Cette application est un forum de discussion en source libre où les internautes ont la possibilité de s'inscrire et poster des message. *PHP* (4.3 ou plus récent) et *MySQL* (4.1.2 ou plus récent) sont nécessaires.

Pligg [15]

Pligg est une application de gestion de contenu (CMS en anglais pour Content Management System) en source libre. Elle permet de créer un site Internet communautaire où les contenus sont créés et votés par les membres inscrits. Les nouveaux articles soumis par les utilisateurs sont ainsi notés par d'autres utilisateurs et affichés en page d'accueil s'ils remportent le succès nécessaire. Le *CMS Pligg* fournit un logiciel d'édition qui permet aux visiteurs de s'inscrire sur le site créé afin qu'ils puissent soumettre des contenus et communiquer avec d'autres utilisateurs. *PHP* (4.3 ou plus récent) et *MySQL* (4.1.2 ou plus récent) sont nécessaires.

3.5.3.2 Résultats

Nous avons inspecté manuellement toutes les vulnérabilités détectées par chaque scanner, afin d'avoir un contrôle précis sur les résultats de l'expérimentation et obtenir une plus grande confiance sur le nombre de vulnérabilités détectées et les faux positifs.

La figure 3.3 présente les résultats obtenus pour l'application *Cyphor*. Tous les scanners trouvent toutes les vulnérabilités parce que les messages d'erreur ne sont pas désactivés. Ainsi, il est facile d'identifier l'exploitation réussie des vulnérabilités des messages d'erreur. Nous pouvons remarquer que certains résultats ont été soulignés. Ils correspondent à des détections rendues possibles en fournissant le couple valide (login / mot de passe) afin que les scanners puissent réussir l'authentification à l'application. En d'autres termes, la vulnérabilité correspondante est visible uniquement lorsque l'utilisateur est connecté sur le site. Nous remarquons également qu'il y a un faux positif détecté par *Skipfish*.

L'application suivante testée est *Seagull*. Les résultats correspondants sont présentés dans le ta-

12. `http://seagullproject.org/`
13. `http://fttss.sourceforge.net`
14. `http://www.riotpix.com/`
15. `http://www.pligg.com/`

Vulnérabilités			Skipfish	W3af	Wapiti	Wasapy
Type	CVE	Emplacement				
SQL*i*	*NR*	search.php	✓	✓	✓	✓
	2005-3236	lostpwd.php	✓	✓	✓	✓
	2005-3236	newmsg.php	✓	✓	✓	✓
	2005-3575	show.php	✓	✓	✓	✓
Faux positif			1	0	0	0

TABLE 3.3 – Résultats de détection de vulnérabilités pour l'application *Cyphor*

bleau 3.4. *Wasapy* est le seul qui signale une vulnérabilité dans cette application. Aucune vulnérabilité n'a été détectée par les autres scanners car l'application désactive l'envoi des messages d'erreurs vers les clients. Concernant les vulnérabilités File Include, les points d'injection qui permettent leur exploitation ne sont pas directement accessibles par le client. Ainsi, le code source est nécessaire pour identifier ces vulnérabilités. C'est ce qui explique l'échec de tous les scanners.

Vulnérabilité			Skipfish	W3af	Wapiti	Wasapy
Type	CVE	Emplacement				
SQL*i*	*2010-3212*	index.php	✗	✗	✗	✓
FIn	*2010-3209*	container.php	✗	✗	✗	✗
	2010-3209	QuickForm.php	✗	✗	✗	✗
	2010-3209	NestedSet.php	✗	✗	✗	✗
	2010-3209	Output.php	✗	✗	✗	✗
Faux positif			0	0	0	0

TABLE 3.4 – Résultats de détection de vulnérabilités pour l'application *Seagull*

Fttss est une application qui a été testée dans MOTH. Ainsi, certains résultats associés aux trois scanners commerciaux considérés sont disponibles (cf. tableau 3.5). Les scanners commerciaux ne détectent pas la vulnérabilité Os Commanding, qui est la seule vulnérabilité connue dans cette application. En revanche, *W3af* et *Wasapy* sont en mesure d'identifier cette vulnérabilité. *Skipfish* et *Wapiti* ne la détectent pas parce que l'application ne signale aucun message d'erreur. Il est à noter qu'aucun des

scanners testés ne génère de faux positifs dans ce cas.

Vulnérabilité			Skipfish	W3af	Wapiti	Wasapy	AppScan	WebInspect	Acunetix
Type	CVE	Emplacement							
OsC	*NR*	index.php	✗	✓	✗	✓	✗	✗	✗
Faux positif			0	0	0	0	0	0	0

TABLE 3.5 – Résultats de détection de vulnérabilités pour l'application *Fttss*

En ce qui concerne *Riotpix* (cf. tableau 3.6), les résultats sont similaires à ceux de *Cyphor*. Les vulnérabilités ne sont accessibles qu'après l'authentification réussie des utilisateurs. Une vulnérabilité n'a été découverte par aucun scanner. Elle correspond à l'injection de code dans les variables qui ne sont pas accessibles par le client et donc qui ne peuvent pas être découvertes par les scanners (il serait nécessaire d'effectuer une analyse du code source, c'est-à-dire une approche en boîte blanche).

La dernière application que nous avons testée est *Pligg* (cf. tableau 3.7). Dans cette application, toutes les vulnérabilités, sauf les deux premières, ne sont pas directement accessibles car les points d'injection correspondants sont cachés. Le scanner doit être averti de la présence de ce point d'injection afin de tester la vulnérabilité. Ceci explique les faux négatifs. Pour les deux premières vulnérabilités, *Wasapy* les a découvertes, alors que les autres scanners trouvent seulement une de ces vulnérabilités. Cela est dû au fait que les messages d'erreur sont désactivés par l'application.

3.5.4 Synthèse

La synthèse de toutes nos expériences est résumée dans le tableau 3.8. Ces expériences montrent que :

– *Wasapy* est un scanner efficace, surtout dans des conditions particulières pour lesquelles il a été conçu : 1) il est plus efficace que les autres scanners en source libre testés lorsque les notifications d'erreur sont désactivés, 2) il est plus efficace que les autres scanners pour découvrir et exploiter les vulnérabilités qui sont incluses dans les pages qui ne sont pas directement accessibles (pages qui nécessitent l'exploitation réussie d'une vulnérabilité pour être consultées). En effet, notre scanner est le seul qui est capable d'exploiter la vulnérabilité et fournir les requêtes exactes pour les injections correspondantes.
– *Wasapy* est globalement aussi efficace que les autres scanners de vulnérabilités testés sur les applications web vulnérables non modifiées.
– Notre algorithme de "clustering" peut être facilement adapté à différents types de vulnérabilités. En effet, même s'il a été conçu pour les injections SQL, les résultats des expériences montrent que *Wasapy* détecte également les vulnérabilités XPATH, OS commanding et File Include et qu'il est au moins aussi efficace que les autres scanners de vulnérabilités.

Vulnérabilité			Skipfish	W3af	Wapiti	Wasapy	AppScan	WebInspect	Acunetix
Type	CVE	Emplacement							
SQL*i*	*NR*	edit_post.php	✗	✗	✗	✓	✗	✗	✗
	NR	edit_post_script.php	✗	✗	✗	✗	✗	✗	✗
	NR	index.php	✗	✗	✗	✓	✗	✗	✗
	NR	message.php	✗	✗	✗	✓	✗	✗	✗
	NR	reader.php	✓	✓	✗	✓	✗	✗	✗
Faux positif			0	0	0	0	0	0	0

TABLE 3.6 – Résultats de détection de vulnérabilités pour l'application *Riotpix*

3.6 Conclusion

Dans ce chapitre, nous avons présenté un scanner de vulnérabilité qui implémente un nouvel algorithme pour identifier la présence de certains types de vulnérabilités et fournir les requêtes permettant d'exploiter ces vulnérabilités suivant une approche boîte noire. La génération automatique des requêtes est basée sur des grammaires. L'identification des requêtes conduisant à une exploitation réussie des vulnérabilités est basée sur la classification automatique des réponses du serveur basée sur un algorithme de "clustering".

Nous avons utilisé un grand nombre d'applications avec et sans injection volontaire des vulnérabilités afin de valider notre approche et d'analyser son efficacité par rapport aux trois outils existants disponibles en source libre. Les résultats expérimentaux sont prometteurs. En particulier, nous avons montré que notre algorithme peut contribuer à améliorer la capacité de détection de certains outils existants et à automatiser le processus de détection de vulnérabilité (en permettant par exemple la détection réussie de vulnérabilités dans les pages qui ne sont pas directement accessibles, mais qui sont uniquement accessibles après le succès de l'exploitation d'une première vulnérabilité).

Nous nous sommes principalement intéressés, dans ce chapitre, aux injections SQL. Néanmoins, nous avons également montré que l'algorithme proposé peut aussi être appliqué avec succès à d'autres types de vulnérabilités. L'algorithme proposé a été mis en œuvre dans un nouvel outil *Wasapy*. Cet algorithme peut aussi être intégré dans les outils en source libre que nous avons étudiés tel que *W3af* et *Skipfish* pour profiter des autres fonctionnalités puissantes offertes par ces outils.

Notre algorithme permet de réaliser des attaques élémentaires et ne gère pas les dépendances entre les URLs et les attaques, et ceci est dû au fait qu'il utilise seulement une session pour naviguer sur le site cible. Autrement dit, ce modèle ne permet pas de déterminer s'il existe une relation d'ordre

Vulnérabilité			Skipfish	W3af	Wapiti	Wasapy	AppScan	WebInspect	Acunetix
Type	CVE	Emplacement							
SQL*i*	*2008-7091*	login.php	✗	✗	✗	✓	✗	✓	✗
	2008-7091	story.php	✓	✗	✓	✓	✓	✓	✓
	NR	userrss.php	✗	✗	✗	✗	✓	✓	✓
	2008-7091	out.php	✗	✗	✗	✗	✓	✗	✓
	2008-7091	trackback.php	✗	✗	✗	✗	✗	✗	✗
	2008-7091	cloud.php	✗	✗	✗	✗	✗	✗	✗
	2008-7091	cvote.php	✗	✗	✗	✗	✗	✗	✗
	2008-7091	recommend.php	✗	✗	✗	✗	✗	✗	✗
	2008-7091	submit.php	✗	✗	✗	✗	✗	✗	✗
	2008-7091	vote.php	✗	✗	✗	✗	✗	✗	✗
	2008-7091	edit.php	✗	✗	✗	✗	✗	✗	✗
False positive			0	0	0	2	1	1	0

TABLE 3.7 – Résultats de détection de vulnérabilités pour l'application *Pligg*

entre les URLs car nous n'avons pas assez de traces.
Par conséquent, nous avons pensé à améliorer cette approche et proposer une solution pour tenir compte de ces performances. Cette solution consiste principalement à générer le graphe de navigation du site cible et à élaborer des scénarios d'attaques possibles. Nous détaillons cette approche dans le chapitre suivant.

	Skipfish	W3af	Wapiti	Wasapy
True Positive	6	6	5	12
False Positive	1	0	0	3
False Negative	20	20	21	14

TABLE 3.8 – Résumé des résultats

4

Dépendance entre vulnérabilités et génération de scénarios d'attaque

Introduction

La contribution décrite dans ce chapitre complète celle du chapitre précédent pour la génération du trafic malveillant pour l'analyse et l'évaluation d'applications Web. L'objectif est de générer automatiquement des scénarios d'attaques prenant en compte explicitement les dépendances entre vulnérabilités. Ces scénarios sont élaborés à partir d'un graphe d'attaques qui est construit dynamiquement en identifiant de façon itérative les différentes possibilités pour naviguer au travers du site Web et des

vulnérabilités susceptibles d'être exploitées par un attaquant via les différents points d'entrée identifiés au cours de l'exploration du site. Pour ce faire, nous commençons par un parcours combinatoire du site qui a pour objectif d'identifier toutes les traces de navigation possibles à travers le site Web. Ainsi, un nombre important d'urls peuvent être identifiées, qui sont traitées par la suite dans le but de générer un graphe réduit du site parcouru que nous appelons *graphe de navigation*. Ensuite, à partir de ce graphe commence une étape d'identification et d'exploitation de vulnérabilités conformément à l'approche décrite dans le chapitre 3 en envoyant des requêtes spécifiques à travers les liens disponibles à partir de chaque nœud du graphe de navigation. L'exploitation de vulnérabilités permet de révéler de nouvelles urls qui n'étaient pas initialement accessibles. Ces deux étapes s'exécutent itérativement jusqu'à aboutir au graphe final modélisant le site complet. Ce graphe final permet enfin de définir l'ensemble des scénarios d'attaque incluant les différentes vulnérabilités identifiées en tenant compte de l'ordre dans lequel ces vulnérabilités peuvent être exploitées.
Ce chapitre est composé de six sections. La section 4.1 donne un aperçu de différents travaux liés aux graphes d'attaque dans la littérature. La section 4.2 décrit notre approche pour la génération du graphe de navigation. L'algorithme qui a été mis en œuvre pour la génération de ce graphe est présenté dans la section 4.3. L'approche proposée est illustrée sur un cas d'étude qui est présentée dans la section 4.4. Une étude de complexité de l'algorithme de génération du graphe d'attaque est effectuée dans la section 4.5. Enfin, la section 4.6 conclut ce chapitre.

4.1 Graphe d'attaque

La génération de scénarios d'attaques a fait l'objectif d'un grand nombre de travaux de recherche sur les *graphes d'attaque*.
Un *graphe d'attaque* est un formalisme qui permet de décrire les vulnérabilités dans un système et les scénarios d'attaque possibles (à travers des chemins dans le graphe) qui peuvent être utilisés par un intrus pour atteindre un objectif spécifique. Dans un graphe d'attaque, les nœuds modélisent les états du processus d'attaque et représentent généralement les privilèges possédés par l'attaquant et les connaissances dont il dispose sur l'état du système cible et de son environnement. Les arcs représentent les actions de l'attaquant et les vulnérabilités dont l'exploitation permet le passage de l'état source vers l'état destination.
Diverses formes de graphes d'attaque ont été proposées dans la littérature, nous citons par exemple [INGOLS ET AL. 06],[DACIER 94], [PHILLIPS ET SWILER 98], [ORTALO ET AL. 99],[SHEYNER ET AL. 02] et [NOEL ET AL. 10]. Ces travaux focalisent aussi sur l'optimisation de la construction des graphes d'attaque et la maîtrise de leur complexité. Dans le même contexte, nous pouvons citer également les arbres d'attaque [SCHNEIER 99], [MAUW ET OOSTDIJK 05] et les arbres d'attaque-défense [KORDY ET AL. 11]. Ces formalismes constituent une adaptation des arbres de fautes utilisés couramment pour représenter des scénarios de défaillance dans des études de sûreté de fonctionnement.
A notre connaissance, les graphes d'attaque et les arbres d'attaque n'ont pas été utilisés à ce stade

dans le contexte des applications Web. Cependant ils peuvent être très utiles pour mettre en évidence les dépendances entre vulnérabilités dans le sens où certaines ne peuvent être découvertes que suite à l'exploitation d'autres vulnérabilités. Ces informations sont utiles pour améliorer la sécurité des applications Web complexes et éliminer les vulnérabilités existantes qui ne peuvent pas être détectées facilement par les scanners Web traditionnels. En effet, l'identification automatique de scénarios d'attaques qui incluent les dépendances causales entre les vulnérabilités nécessite le développement de nouvelles techniques qui permettent l'exploitation des vulnérabilités durant l'exécution dynamique d'une application. L'approche présentée dans ce chapitre, qui est synthétisée dans la section suivante, vise à répondre à cet objectif.

4.2 Présentation de l'approche

Rappelons que l'objectif de notre approche [ALATA ET AL. 12] est de pouvoir construire automatiquement un graphe d'attaque qui représente l'ensemble des navigations possibles sur un site en tenant compte de ses vulnérabilités. Nous commençons cette section par la définition de certains termes qui sont utiles pour la présentation de notre approche. Par la suite nous détaillons les deux étapes de notre approche qui s'exécutent d'une manière itérative.

4.2.1 Définitions

Une *navigation* sur un site correspond à une séquence de requêtes envoyées au site par le client. Un *état de navigation* correspond au contenu du *cookie* stocké en local sur le navigateur du client ainsi qu'à la page affichée par ce navigateur. Un état dépend de l'historique de la navigation sur le site. Une requête permet donc de passer d'un état de navigation à un autre état de navigation. L'ensemble des navigations possibles peuvent être représentées sous la forme d'un *graphe de navigation*.
Chaque *nœud* du graphe correspond à un état de navigation. Un *arc* entre deux états correspond à la possibilité d'exécuter une requête permettant, depuis un état de départ, d'atteindre un autre état. Un arc peut correspondre à une requête normale ou à une requête liée à l'exploitation d'une vulnérabilité sur le site. Un *graphe de vulnérabilité* est un cas particulier de graphe de navigation contenant des arcs correspondant à des exploitations de vulnérabilités.

Un graphe de navigation est différent d'un graphe de pages HTML décrivant la structure du site Web. Dans un graphe de pages HTML, chaque nœud correspond à une page du site et un arc entre deux nœuds correspond à l'existence d'un lien HTML permettant de passer de la première page à la seconde. Ce type de graphe est utile pour représenter la structure d'un site Web. La différence entre un graphe de navigation et un graphe de pages HTML tient principalement du fait qu'un état de navigation ne dépend pas simplement de la page accédée. Effectivement, nous pouvons accéder à une même page d'un site tout en étant dans des états différents. Par exemple, dans un site marchand nous pouvons accéder à la page de paiement soit en ayant des produits dans le panier soit sans en avoir. Dans le premier cas, le cookie du navigateur contient les informations du panier et dans le second cas, nous pouvons imaginer

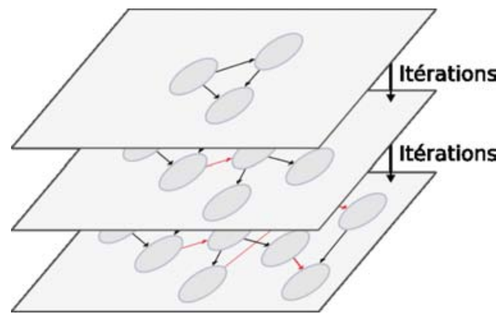

FIGURE 4.1 – Principe de l'algorithme

que le cookie est vide. Nous pouvons donc accéder à la page de paiement dans des états différents. Or, dans le premier cas l'utilisateur peut poursuivre son paiement, contrairement au second cas où le site Web avertit que le panier est vide. Notons toutefois que, dans certains cas de sites statiques (sans contenu variable), le graphe de navigation et le graphe des pages HTML se ressemblent beaucoup.

Nous nous plaçons dans la situation où nous disposons uniquement de l'adresse du site, qui constitue le premier paramètre de notre algorithme. La construction du graphe de navigation doit donc se faire de manière progressive, par identification des différentes navigations sur le site et par identification des vulnérabilités. De plus, l'exploitation d'une vulnérabilité ouvre de nouvelles possibilités de navigation. La construction doit donc également se faire de manière itérative. Notre approche est donc constituée d'une étape de navigation appelée *crawling* et d'une étape d'identification des vulnérabilités, exécutées itérativement que nous détaillons dans la sous-section suivante. La figure 4.1 décrit de façon schématique cette approche itérative. Les arcs représentés en rouge identifient des vulnérabilités dont l'exploitation permet de révéler de nouveaux états et arcs du graphe de navigation qui étaient initialement inaccessibles.

4.2.2 Crawling

La première étape correspond au *crawling* du site permettant d'identifier les différentes navigations du site et ainsi les différents états de navigation. Notre approche est basée sur une recherche exhaustive permettant d'obtenir toutes les navigations possibles du site. Pour ce faire, nous prenons soin de partir d'un état initial vide en supprimant les cookies côté client. Cette initialisation est importante pour que les différentes navigations soient indépendantes. En effet, sans suppression des cookies, un effet de mémoire peut rendre la seconde navigation dépendante de la première. Ensuite, nous naviguons sur le site en commençant par la requête initiale et en mémorisant les requêtes envoyées au site. Le choix de la requête à envoyer se fait en analysant le contenu de la page affichée. Si cette page contient plusieurs liens HTML, un de ces liens est choisi pour construire la requête suivante et les autres liens sont mémorisés pour les analyser ultérieurement. Sachant que la navigation à travers un site peut être infinie, nous bornons le nombre de requêtes envoyées. Cette borne représente la profondeur maximale

de navigation du site. Elle constitue un second paramètre de notre algorithme.

Lorsque nous aboutissons à une situation où la borne est atteinte ou l'état atteint ne permet plus d'envoyer de requêtes, autrement dit la page atteinte ne contient plus de liens HTML, alors la séquence de requêtes mémorisées constitue une navigation du site. Nous recommençons alors à zéro en essayant de nouvelles navigations, en se basant sur les choix mémorisés jusqu'à avoir tout essayé compte tenu de la borne fixée. Nous obtenons ainsi un ensemble de séquences de requêtes représentant l'ensemble de navigations du site. Cet ensemble est utilisé pour construire une première version du graphe de navigation dépourvu d'arcs correspondant à des vulnérabilités. L'objectif est d'obtenir un graphe minimal qui permet de représenter l'ensemble des navigations obtenues.

Inférence grammaticale

Le passage d'un ensemble de séquences de requêtes à un graphe peut être considéré comme un problème d'inférence grammaticale pour lequel nous cherchons à obtenir un automate minimal à partir d'exemples du langage, plus précisément à partir de séquences de symboles du langage. Dans cette analogie, l'automate correspond à notre graphe de navigation et les symboles correspondent à nos requêtes. La suite de ce paragraphe présente rapidement le principe de l'inférence grammaticale et nous présentons également l'algorithme de ce domaine que nous utilisons pour résoudre notre problème.

L'objectif de l'inférence grammaticale est d'obtenir un automate qui représente un langage, en s'appuyant sur des exemples de mots issus de ce langage. Un langage peut contenir un nombre infini de mots. Donc, les algorithmes doivent pouvoir fonctionner en s'appuyant uniquement sur un sous-ensemble de mots du langage étudié. Plus précisément, les algorithmes utilisent deux ensembles disjoints : les exemples positifs qui sont les séquences qui appartiennent au langage et les exemples négatifs qui sont les séquences qui n'appartiennent pas au langage. Des séquences peuvent être ni positives ni négatives et nous les nommerons les exemples neutres [1]. Ces algorithmes peuvent être séparés en deux catégories.

La première catégorie correspond aux algorithmes qui s'appuient exclusivement sur les exemples positifs. L'automate minimal construit par ces algorithmes doit reconnaître toutes ces séquences et rejeter le plus possibles les autres. Cet automate peut donc accepter un exemple neutre plus court que la plus longue séquence positive. Il s'agit donc de trouver un compromis entre une minimisation excessive qui reconnaît beaucoup d'exemples neutres et une minimisation nulle correspondant à un arbre de préfixe qui reconnaît exclusivement les séquences positives.

Un exemple d'inférence grammaticale avec des exemples positifs est présenté dans la figure 4.2. L'objectif dans cet exemple est de trouver un automate pour un langage L. Nous savons uniquement que les mots b, aa et aba sont des mots de L (ils sont notés $E+$) et nous ne savons pas si parmi les exemples neutres (par exemple : ab, bbb, etc.) certains correspondent en fait à des mots du langage. L'approche présentée peut engendrer l'automate de la figure. Cet automate permet effectivement de reconnaître les mots b, aa et aba. Il reconnaît également d'autres mots tels que $abba$ et $abbba$. Par contre, certains exemples neutres sont rejetés par l'automate (par exemple $abab$). Le langage correspondant à cet

1. Ce terme n'est pas un terme consacré du domaine mais ajouté dans ce chapitre pour simplifier les explications

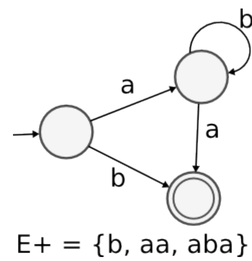

FIGURE 4.2 – Exemple avec des exemples positifs

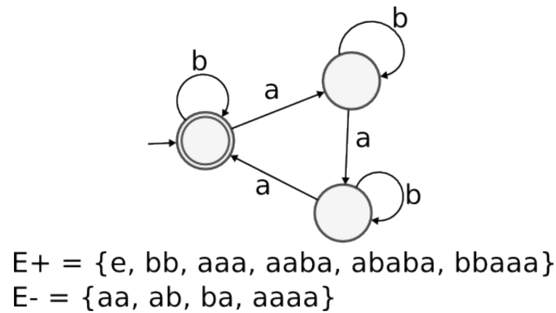

FIGURE 4.3 – Exemple avec des exemples positifs

et négatifs

automate est $b|ab^*a$.

La seconde catégorie correspond aux algorithmes qui s'appuient sur les deux ensembles d'exemples à la fois. Leur objectif est de construire un automate minimal acceptant tous les exemples positifs et rejetant tous les exemples négatifs. L'automate obtenu peut également accepter un exemple neutre tout comme il peut le rejeter. La figure 4.3 présente un exemple de résultat obtenu avec des algorithmes de cette catégorie. Les exemples positifs sont ϵ, bb, aaa, $aaba$, $ababa$ et $bbaaa$ et les exemples négatifs sont aa, ab, ba et $aaaa$. De la même manière que précédemment, nous pouvons vérifier que l'automate reconnait les mots de $E+$. De plus, cet automate rejette les mots de $E-$.

Dans notre situation, nous aurions tendance à employer un algorithme de la première catégorie, étant donné que nous disposons uniquement des navigations correspondant aux exemples positifs. Ce choix n'est pas pertinent car il aboutit à un automate susceptible d'accepter des séquences de requêtes plus courtes que la borne fixée et qui ne sont pas des navigations du site. Par contre, nous sommes capables de construire un ensemble d'exemples négatifs en considérant toutes les séquences de requêtes plus courtes que la borne et qui ne sont pas des navigations du site. Pour résoudre notre problème nous nous appuyons donc sur un algorithme de la seconde catégorie.
Nous utilisons en particulier l'algorithme RPNI (**R**egular **P**ositive **N**egative Inference)[DUPONT 96]

qui présente une complexité en temps polynomial, raisonnable par rapport aux autres algorithmes, tout en étant simple à implémenter. Son principe de fonctionnement repose sur la construction d'un arbre de préfixe reconnaissant les exemples positifs et des tentatives de fusion de nœuds. Les tentatives de fusion sont réalisées en considérant les nœuds deux à deux en partant du nœud racine. Une fusion est conservée si l'automate obtenu rejette toujours les exemples négatifs. A la fin de cette étape de crawling, nous notons que la seule possibilité d'enrichir le modèle est de considérer des attaques permettant de rajouter des arcs voire des nœuds. En effet, il n'existe pas de navigations composées uniquement de requêtes connues et de longueur inférieure à la borne permettant d'enrichir ce modèle. Si de telles navigations existent, alors nous pouvons considérer que le crawling était incomplet car il aurait du générer ces navigations. Par conséquent, l'automate obtenu à l'issu de cette étape correspond au graphe de vulnérabilités qui sera utilisé comme base pour l'étape suivante d'identification des vulnérabilités.

4.2.3 Identification des vulnérabilités

La seconde étape vise à chercher les vulnérabilités qui peuvent être exploitées sur le site. Le graphe de navigation est utilisé pour effectuer cette recherche. Il est ensuite enrichi avec les vulnérabilités identifiées et utilisé lors des itérations suivantes.

Etant donné le nombre important de requêtes envoyées pour obtenir ce graphe, mener une recherche de vulnérabilité sur chacune des requêtes envoyées serait excessivement long à réaliser. Afin de limiter cette phase de recherche, nous proposons dans notre approche de mener une recherche sur chacun des arcs du graphe de navigation uniquement. Le nombre d'arcs étant largement inférieur au nombre total de requêtes envoyées lors du crawling, la recherche sera largement plus rapide. Cette stratégie revient à poser une hypothèse forte : l'exploitation d'une vulnérabilité ne dépend que de l'état de navigation et est indépendante de la navigation ayant permis d'atteindre cet état. Aussi, il peut exister différentes navigations amenant dans un même état et permettant d'exploiter une même vulnérabilité. D'ailleurs, c'est en se focalisant sur un seul de ces chemins pour l'analyse de l'état que nous accélerons la recherche.

La recherche depuis un arc n'a de sens que si le navigateur utilisé pour la recherche est dans l'état correspondant à l'état de départ de l'arc. Toutes les requêtes du chemin choisi pour atteindre l'état doivent être exécutées dans l'ordre. Etant donné l'hypothèse précédente, nous choisissons le plus court chemin allant de l'état initial (avec des cookies vides) à cet état.

Rappelons toutefois que l'algorithme est itératif. Il faut donc éviter de tester deux fois les arcs d'un même état. Cela revient à tester uniquement les arcs en sortie des nouveaux états, par rapport au graphe de l'itération précédente. Pour chacun des arcs, la recherche est réalisée sur tous les paramètres de la requête. A ce niveau, nous donnons la main à l'algorithme d'identification de vulnérabilités basé sur l'outil *wasapy* présenté dans le chapitre 3.

Algorithm 1 $crawler(path, d_m)$

Require: $path, d_m$
Ensure: new_paths

```
1:  remain ← {path}
2:  traces ← ∅
3:  d ← |path|
4:  while remain ≠ ∅ ∧ d ≤ d_m do
5:    next ← ∅
6:    for trace ∈ remain do
7:      free_cookies()
8:      for i ∈ 1...|trace| do
9:        links ← response(trace_i)
10:     end for
11:     for link ∈ links do
12:       next ← next ∪ {trace ⊕ link}
13:       traces ← traces ∪ {trace ⊕ link}
14:     end for
15:   end for
16:   remain ← next
17:   d ← d + 1
18: end while
19: return traces_0
```

4.3 Algorithme de génération de graphe de navigation

Nous présentons dans cette section l'algorithme correspondant à notre approche, décrite dans la section précédente, et nous détaillons ses différentes fonctions. Elles sont présentées dans les algorithmes 3, 1 et 2. Dans ces algorithmes, une trace correspond à une navigation sur le site.

La fonction $crawler$ de l'algorithme 1 permet de découvrir une partie d'un site en partant d'une navigation fournie en paramètre. La variable $remain$ contient l'ensemble des navigations qui viennent d'être découvertes et qui doivent donc encore être analysées. Cet algorithme prend fin lorsqu'il n'y a plus de navigations à analyser, autrement dit lorsque l'ensemble $remain$ est vide, ou lorsque la borne d'exploration du site nommée d_m est atteinte. Avant l'analyse de chaque navigation, les cookies sont supprimés. Ensuite, la navigation est exécutée requête après requête. Le contenu de la réponse associée à la dernière requête est analysé : il determine les nouvelles requêtes exécutables. De nouvelles navigations, correspondant à la navigation en cours d'analyse enrichie de ces requêtes, devront à leur tour être analysées. Ces nouvelles navigations sont construites en utilisant l'opérateur $\oplus$ qui représente la concaténation d'une requête à une navigation. Elles sont ensuite stockées dans l'ensemble $remain$.

La fonction $search_vulns$ de l'algorithme 2 utilise une navigation comme paramètre. L'objectif

Algorithm 2 $search_vulns(path)$

Require: $path =$ navigation
Ensure: $vulns =$ ensemble de navigations
1: $vulns \leftarrow \emptyset$
2: **for** $class \in classes$ **do**
3: $\quad vulns \leftarrow vulns \cup wasapy(path)$
4: **end for**
5: **return** $vulns$

de cette fonction est d'analyser la dernière requête de cette navigation pour rechercher des vulnérabilités. L'analyse est réalisée avec la fonction $wasapy$, en considérant les différentes classes de vulnérabilités. Le retour de cette fonction est une liste de navigations avec, pour chacune et comme dernière requête, une des vulnérabilités identifiées.

La fonction $main$ de l'algorithme 3 permet d'enchaîner les deux fonctions précédentes. Lors de la première itération, les invocations de la fonction $crawler$ permettent de découvrir le site en ne considérant que les requêtes normales. Ensuite, les navigations obtenues sont condensées en un graphe avec l'algorithme RPNI. Chaque état de ce graphe fait alors l'objet d'une analyse de vulnérabilités avec la fonction $search_vulns$. A l'issue de la première itération, nous disposons de toutes les navigations contenant au plus une vulnérabilité et finissant par cette vulnérabilité. Si des vulnérabilités ont effectivement été identifiées, leur exploitation peut éventuellement permettre de découvrir de nouvelles parties du site. L'itération suivante peut alors débuter. Le début de chaque itération i de la fonction $main$ correspond à l'exploration d'une sous-partie du site située juste après l'exploitation de la $(i-1)$-ème vulnérabilité de la navigation. La fin de chaque itération i de la fonction $main$ correspond à la recherche de vulnérabilités en considérant des navigations contenant $i-1$ vulnérabilités et finissant pas une requête normale. A l'issue de la i-ième itération, nous disposons de toutes les navigations contenant au plus i vulnérabilités. La fonction prend fin lorsque nous atteignons la borne ou lorsque plus aucune vulnérabilité ne peut être identifiée.

Le nombre de boucles dans cet algorithme peut vite nous amener à nous poser la question de sa complexité. Cette étude fait l'objet de la section 4.5. Et pour illustrer notre approche, nous présentons dans la section suivante un exemple où nous décrivons l'exécution de notre algorithme étape par étape sur un site simple que nous avons développé.

4.4 Exemple

Afin d'illustrer notre approche, nous avons développé un site Web simple en utilisant le langage `php` et la base de données `mysql`. Il s'agit d'un site de commerce électronique qui permet l'achat de livres. Ce site n'est pas fait pour être représentatif de tous les sites disponibles sur l'Internet. Néanmoins, il est conçu pour illustrer les dépendances entre vulnérabilités. Le site Web est simple car il ne contient pas beaucoup de pages différentes. Cependant, le site inclut des pages dynamiques et implémente les

Algorithm 3 $main(urls)$

Require: $urls$
Ensure: $(G = (S, N, R), vulns)$
1: $G \leftarrow RPNI(urls)$
2: $ntraces \leftarrow urls$
3: $traces \leftarrow urls$
4: $vulns \leftarrow \emptyset$
5: **while** $|ntraces| \neq 0$ **do**
6:   **for** $nt \in ntraces$ **do**
7:     **if** $|nt| < d_m$ **then**
8:       $traces \leftarrow traces \cup crwl(nt, d_m)$
9:     **end if**
10:   **end for**
11:   $G' \leftarrow RPNI(traces)$
12:   $new_nodes \leftarrow N' \setminus N$
13:   $ntraces \leftarrow \emptyset$
14:   **for** $nn \in new_nodes$ **do**
15:     $ptnn \leftarrow shortest_path(R', S', nn)$
16:     **if** $|ptnn| < d_m$ **then**
17:       $nptv \leftarrow search_vulns(ptnn)$
18:       **for** $np \in nptv$ **do**
19:         $new_vuln \leftarrow np_{|np|}$
20:         $vulns \leftarrow vulns \cup \{new_vuln\}$
21:       **end for**
22:       $ntraces \leftarrow ntraces \cup nptv$
23:     **end if**
24:   **end for**
25:   $G \leftarrow G'$
26: **end while**
27: **return** $(G, vulns)$

principales fonctionnalités qui sont couramment disponibles sur des sites sur Internet : connexion et authentification, recherche, description des livres disponibles, paiement, etc. Le site intègre aussi différentes vulnérabilités. Certaines vulnérabilités ne peuvent être atteintes qu'après l'exploitation réussie d'autres vulnérabilités.

La figure 4.4 donne une vue d'ensemble du site complet. Une page est représentée comme une icône de fichier. La page principale est `index.html`. Si une page contient un lien vers une autre page, une flèche est ajoutée entre les icônes correspondantes. La page `display.php` est associée à un lien réflexif car elle contient un lien vers elle-même, utilisé pour mettre à jour le modèle de la recherche. Dans cet exemple, nous considérons trois vulnérabilités. Les pages qui contiennent ces vulnérabilités

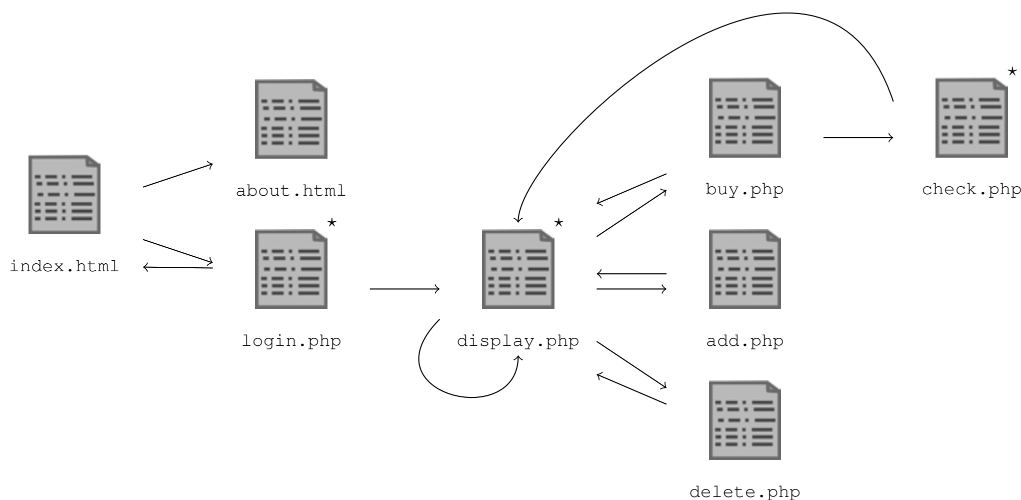

FIGURE 4.4 – Structure du site Web

sont marquées par une étoile en haut à droite de l'icône.

La première est associée à la page `login.php`. L'exploitation de cette vulnérabilité permet de contourner l'authentification par une injection SQL. La seconde est associée à la page `display.php` et permet à l'attaquant de faire un "dump" du contenu de la base de données. La dernière est associée à la page `check.php` et elle permet d'acheter le contenu du panier sans fournir un numéro de carte valide. Elle ne peut être exploitée que si le produit a été ajouté au panier.
Dans la sous-section suivante, nous illustrons l'exécution de l'algorithme en considérant une borne maximale pour l'exploration du graphe d'une valeur de 7.

4.4.1 Construction automatique du graphe de navigation

Tout d'abord, nous considérons le point de vue d'un utilisateur normal, qui ne dispose pas d'un compte sur le site. Les seules actions que cet utilisateur peut faire sont :

- accéder à la page `index.html`
- remplir le formulaire d'authentification avec des informations dans la page `login.php`
- obtenir des informations à partir de la page `about.html`

Ces actions sont présentées dans le graphe de navigation de la figure 4.5. Dans ce graphe, les arcs correspondent aux liens du site (page HTML, la page php, etc.). Un nœud correspond à un état de navigation. L'ensemble des arcs de sortie d'un nœud correspond à l'ensemble de liens accessibles de

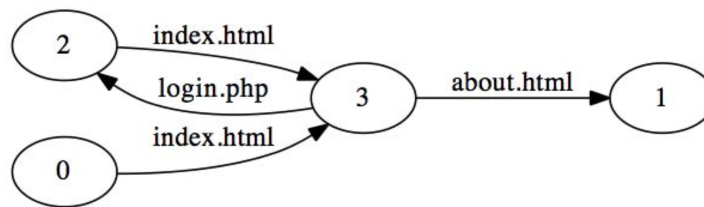

FIGURE 4.5 – Graphe de navigation d'un utilisateur non enregistré

P_1 : `index.html`
P_2 : `index.html` $\rightarrow$ `about.html`
P_3 : `index.html` $\rightarrow$ `login.php`
P_4 : `index.html` $\rightarrow$ `login.php` $\rightarrow$ `index.html`

FIGURE 4.6 – Liste des scénarios utilisés pour tester les vulnérabilités

l'état de navigation correspondant. Cet ensemble est indépendant des liens précédemment utilisés pour atteindre cet état de navigation.

Ce graphe de navigation est très simple parce que les actions possibles sans (`login` / `password`) valides sont limitées. Toutefois, si nous considérons le point de vue d'un attaquant et s'il est en mesure d'exploiter correctement les vulnérabilités du site, alors il est capable d'exécuter plus d'actions que l'utilisateur normal. Par conséquent, le graphe de navigation associé correspond à une version plus riche que le graphe 4.5. En effet, de nouveaux arcs et nouveaux nœuds peuvent être ajoutés.

Durant l'exécution du crawler pour la première itération de notre algorithme, la première trace obtenue correspond à une navigation régulière à travers les pages HTML sans exploitation de vulnérabilités. Le graphe correspondant obtenu est présenté dans la figure 4.5. Dans ce graphe, il y a quatre arcs à tester. Pour chacun de ces arcs, l'algorithme crée un scénario à partir du nœud initial jusqu'à cet arc (voir la figure 4.6). Un scénario est donc composé des actions élémentaires successives où chacune correspond à l'exécution d'un lien de la page accédée à l'état de la navigation courante.

Les scénarios de la figure 4.6 sont testés afin d'identifier les vulnérabilités. Le test est effectué par l'outil `WASAPY` : avant chaque test de vulnérabilité, le scénario est exécuté par `WASAPY` afin de parvenir à l'état de navigation correspondant. Cet outil est utile pour obtenir un requête valide pour réussir l'exploit de la vulnérabilité. L'exploitation d'une nouvelle vulnérabilité peut changer l'état de navigation. Ainsi, elle peut mener à l'insertion d'un nouveau nœud sur le graphe. A cette étape, la seule vulnérabilité accessible correspond à l'injection SQL pour l'authentification, i.e., scénario P_3. Le résultat obtenu est présenté dans la figure 4.7.

Au cours de la deuxième itération, le crawler identifie les pages qui peuvent être accessibles suite à l'exploitation de la vulnérabilité identifiée lors de l'itération précédente. Afin d'atteindre ces pages, il

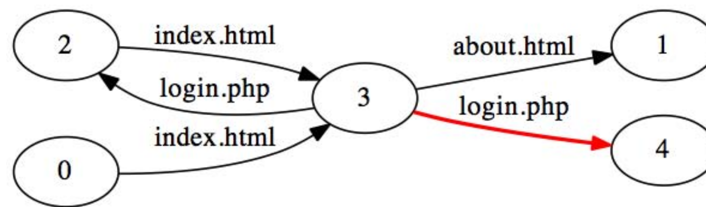

FIGURE 4.7 – Résultats de la première itération de l'algorithme

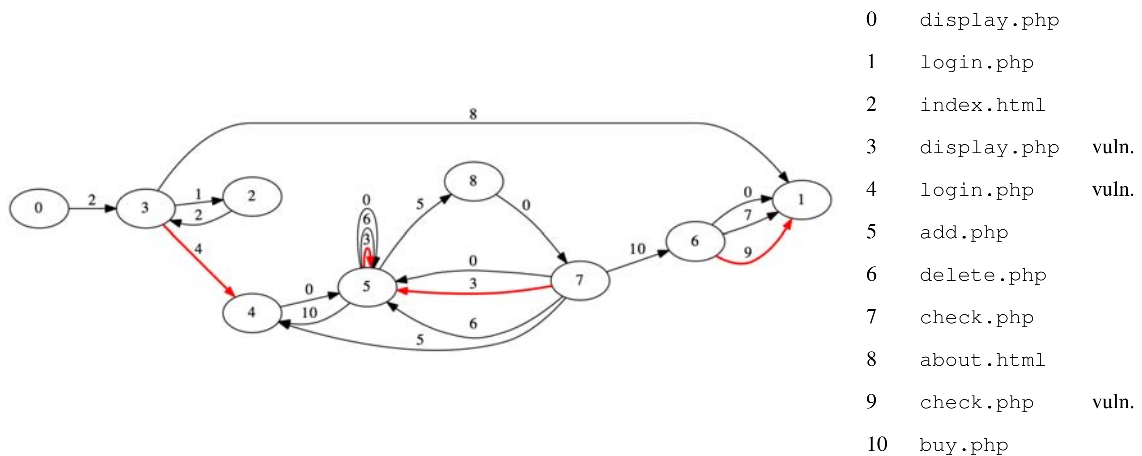

FIGURE 4.8 – Graphe de vulnérabilités final

est nécessaire de traverser les arcs `index.html` et `login.php` *.
L'ensemble des traces exécutées par le crawler pour cette deuxième itération contient 65 traces. Ces traces atteignent `display.php`, `add.php`, `delete.php`, `buy.php` ou `check.php`. Ensuite, la phase d'identification de vulnérabilité est exécutée encore une fois pour chaque nouvel arc de cette seconde itération. Le graphe obtenu à la fin de l'exécution de l'algorithme est présenté dans la figure 4.8. L'algorithme s'arrête après 6 itérations. Cela signifie qu'il n'y a plus de vulnérabilités découvertes au cours de la sixième itération.
Le graphe final contient 9 nœuds et 20 arcs. Néanmoins, il représente 241 scénarios. Le graphe initial généré par toutes les traces, sans l'aide de l'algorithme de réduction RPNI contient 353 arcs. Le graphe de vulnérabilités est plus compact que le graphe initial et conduit à une recherche de vulnérabilités plus efficace. En outre, ce graphe fournit suffisamment d'informations pour en déduire les dépendances causales entre les vulnérabilités. Par exemple, la vulnérabilité associée à l'arc 3 ne peut pas être identifiée et exploitée avant l'exploitation de la vulnérabilité associée à l'arc 4. Nous déduisons ainsi qu'il y a une dépendance causale entre ces deux vulnérabilités.

Basé sur le graphe présenté dans la figure 4.8, nous pouvons extraire différents scénarios d'attaque en fonction de certains critères spécifiques. Par exemple, si nous voulons que tous les scénarios d'attaque contiennent deux vulnérabilités, nous avons besoin d'extraire à partir du graphe l'ensemble

```
S1 :  index.html  →login.php*  →display.php  →display.php*
S2 :  index.html  →login.php*  →display.php  →add.php  →display.php  →display.php*
S3 :  index.html  →login.php*  →display.php  →add.php  →display.php  →buy.php  →check.php*
```

FIGURE 4.9 – Exemple des scénarios d'attaque extraits du graphe final

des chemins qui passent par les arcs 4 et 3, ou 4 et 9. Un exemple de ces scénarios est donné dans la figure 4.9.

A la lumière de ce qui précède, nous notons que le nombre de traces exécutées et utilisées pour obtenir ce graphe est important. En outre, chacune de ces traces comprend plusieurs requêtes. En conséquence, le nombre de requêtes est également important. Il est donc important d'étudier et d'analyser la complexité de notre approche et ce sera le sujet de la section suivante.

4.5 Discussion sur la complexité

Nous allons nous intéresser à la complexité de notre algorithme en nous focalisant sur l'opération élémentaire la plus exigeante en temps de calcul. Il s'agit de l'opération $response(trace_i)$ qui nécessite l'envoi d'une requête à travers le réseau et l'attente de la réponse. En particulier, nous nous intéressons au pire cas d'exécution de cet algorithme. Dans la suite, nous allons présenter ce pire cas. Ensuite nous présenterons le prédicat $T_{l,i}$ qui reprsénte le nombre de chemins de longueur l contenant i attaques. Pour finir, nous étudierons la compléxité de l'algorithme en nous appuyant sur le pire cas et en utilisant ce prédicat. Nous montrerons que la complexité de l'algorithme estimée par le nombre de requêtes N générées est :

$$N \quad = \quad d_m \times n^{d_m} + \sum_{l=0}^{d_m-1} ((l + (l+1) \times n_n \times r \times c) \times n^l)$$

Dans cette formule, d_m correspond à la borne d'exploration du site, n et n_n représentent la complexité du site et r et c correspondent aux paramètres de *wasapy*. Ces différentes notations seront décrites dans la suite.

4.5.1 Pire cas

Pour notre algorithme, le pire cas correspond à un site pour lequel nous arrêtons d'itérer uniquement lorsque nous atteignons la profondeur maximale d_m. Un arrêt prématuré correspond à une itération durant laquelle nous n'identifions plus de vulnérabilités. Le pire cas doit donc nous permettre d'itérer d_m fois dans la boucle principale de l'algorithme 3. Autrement dit, le site correspondant doit nous permettre d'enchaîner d_m attaques à la suite.

Afin de ne pas ménager les efforts, il ne faut pas que l'algorithme $RPNI$ puisse proposer la

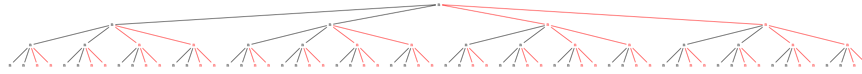

FIGURE 4.10 – Arbre de navigation associé au pire cas

moindre simplification dans la représentation sous forme de graphe du site. Le graphe de vulnérabilités du site ainsi obtenu durant les différentes itérations doit correspondre, pour toutes les itérations, à l'arbre des préfixes des navigations du site. Notons n_a le nombre maximum d'attaques qu'il est possible de suivre depuis chacun des états de navigation et n_n le nombre maximum de requêtes normales qu'il est également possible de suivre depuis chacun des états. Le pire cas est donc un arbre dans lequel chaque état possède n_a attaques et n_n requêtes normales. Plus précisément, il s'agit d'un arbre n-aire avec $n = n_a + n_n$.

Une représentation graphique de l'arbre associé au pire cas est fournie à la figure 4.10. Dans cet exemple, nous avons utilisé les valeurs $n_a = 2$ et $n_n = 2$. Les arcs associés aux attaques sont représentés en rouge et les autres en noir. Chaque état contient donc $n_a + n_n = n = 4$ fils. Nous avons limité la profondeur de l'arbre à 3 niveaux. Dans cet exemple, une navigation correspond à un chemin de l'état initial à l'un des autres états en suivant les arcs. Par exemple, la navigation la plus à gauche enchaîne trois requêtes normales, identifiées par des traits noirs. La navigation la plus à droite enchaîne trois attaques, identifiées par des traits rouges. Entre ces deux navigations, toutes les combinaisons de requêtes normales et d'attaques sont présentes. Leur nombre dépend des paramètres de l'algorithme et de la structure du site. Afin de les dénombrer, nous introduisons la notation $T_{l,i}$ à cet effet.

4.5.2 Comptage des chemins

Notons $T_{l,i}$ le nombre de chemins de longueur l contenant i requêtes associées à des vulnérabilités. Ce nombre correspond au nombre de chemins de l'état initial vers les états du niveau l et incluant exactement i requêtes associées à des vulnérabilités – les traits rouges. Il n'y a aucune restriction sur la position de ces i requêtes sur le chemin. Nous obtenons donc la formule suivante :

$$T_{l,i} = \binom{l}{i} \times n_n^{l-i} \times n_a^i$$

4.5.3 Complexité de $crawler$

La fonction $crawler$ dispose en paramètre d'une navigation de départ, de longueur l. Cette navigation correspond à un chemin de l'arbre, de l'état initial à un nœud de profondeur l. Lors de la première itération de la boucle de la ligne 4 de l'algorithme 1, cette navigation est exécutée et les requêtes normales qui peuvent ensuite être exécutées sont listées dans l'ensemble $links$. Elles sont au nombre de n_n car le $crawler$ ne peut pas identifier les n_a vulnérabilités. Ces requêtes sont utilisées dans les lignes

11 à 14 pour construire les n_n navigations de longueur $l+1$ à utiliser lors de l'itération suivante. Pour les itérations suivantes, ce nombre augmente géométriquement avec un facteur n_n. Le nombre de requêtes exécutées par le $crawler$, noté R_{l,d_m} est (d est le nombre de requêtes exécutées pour une trace de longueur d) :

$$R_l = \sum_{d=l}^{d_m} n_n^{d-l} \times d$$

4.5.4 Complexité de $search_vulns$

La fonction $search_vulns$ dispose en paramètre d'une navigation à analyser. En particulier, la dernière requête doit faire l'objet d'une analyse de vulnérabilités. L'algorithme utilisé est celui mis en œuvre dans l'outil $wasapy$. Pour chacune des c classes de vulnérabilités (SQL Injection, OS Commanding, etc.), r exécutions de la navigation sont réalisées pour la classification. Il s'agit du nombre total de requêtes des ensembles R_{ii}, R_{iv} et R_r, envoyées pour chaque point d'injection, cf. 3.2.1. Bien entendu, ces exécutions doivent permettre de dévoiler l'existence des n_a vulnérabilités. La complexité, notée S_l est donc :

$$S_l = l \times c \times r$$

4.5.5 Complexité de $main$

La fonction $main$ dispose en paramètre des $urls$ du site à analyser. Dans le pire cas, elles correspondent aux n_n requêtes permettant de sortir de l'état initial de l'arbre pour atteindre les états suivants.

Lors de la première itération de la boucle principale, la fonction $crawler$ est invoquée sur chacune des n_n urls fournies en paramètre. A l'issue de ces invocations, nous disposons d'une vision plus précise du site, qui correspond à l'ensemble des états de l'arbre atteignables sans exploiter de vulnérabilités. Ensuite, la fonction $search_vulns$ est invoquée sur chacun des arcs de ce modèle, à la recherche de vulnérabilités. Le nombre de requêtes I_1 envoyées lors de cette première itération est donc :

$$\begin{aligned} I_1 &= (n_n \times R_1) + \sum_{l=1}^{d_m} (T_{l,0} \times S_l) \\ &= R_0 + \sum_{l=1}^{d_m} (T_{l-1,0} \times n_n \times S_l) \end{aligned}$$

Lors des itérations suivantes, indicées par i, la fonction $crawler$ est invoquée sur tous les chemins contenant $i-1$ attaques et finissant par une attaque. Chacun de ces chemins correspond à une attaque découverte lors de l'itération précédente et le $crawler$ est utilisé pour découvrir la portion du site accessible après l'exploitation de cette vulnérabilité. Notons que le nombre de chemins de longueur

l, contenant $i-1$ vulnérabilités et finissant par une attaque est $T_{l,i-1} \times n_a$. Après la découverte de nouveaux liens du site, la fonction $search_vulns$ est à nouveau invoquée sur tous les états découverts, en considérant les plus courts chemins accédant à ces états. Le nombre de requêtes envoyées lors d'une itération $i > 1$ est donc :

$$I_i = \sum_{l=i-1}^{d_m-1} (T_{l-1,i-2} \times n_a \times R_l) + \sum_{l=i}^{d_m} (T_{l-1,i-1} \times n_n \times S_l)$$

Le nombre de requêtes envoyées au total est la somme des nombres de requêtes envoyées lors des différentes itérations :

$$I = I_1 + \sum_{i=2}^{d_m} I_i$$

Notons N_c le nombre de requêtes envoyées par la fonction $crawler$ et N_s le nombre de requêtes envoyées par la fonction $search_vulns$. Nous pouvons démontrer que :

$$\begin{aligned} N_c &= \sum_{l=0}^{d_m} (l \times n^l) \\ N_s &= n_n \times r \times c \times \sum_{l=0}^{d_m-1} ((l+1) \times n^l) \end{aligned}$$

En effet, pour N_c, nous avons :

$$\begin{aligned} N_c &= R_0 + \sum_{i=2}^{d_m} \sum_{l=i-1}^{d_m-1} (T_{l-1,i-2} \times n_a \times R_l) \\ &= R_0 + \sum_{l=1}^{d_m-1} \sum_{i=2}^{l+1} (T_{l-1,i-2} \times n_a \times R_l) \\ &= R_0 + \sum_{l=1}^{d_m-1} (n_a \times R_l \times \sum_{i=2}^{l+1} T_{l-1,i-2}) \\ &= R_0 + \sum_{l=1}^{d_m-1} (n_a \times R_l \times \sum_{i=0}^{l-1} T_{l-1,i}) \\ &= R_0 + \sum_{l=1}^{d_m-1} (n_a \times R_l \times \sum_{i=0}^{l-1} \binom{l-1}{i} \times n_n^{l-1-i} \times n_a^i) \\ &= R_0 + \sum_{l=1}^{d_m-1} (n_a \times R_l \times (n_n + n_a)^{l-1}) \\ &= R_0 + \sum_{l=1}^{d_m-1} (n_a \times R_l \times n^{l-1}) \end{aligned}$$

En montrant que $R_{l-1} = n_n \times R_l + (l-1)$, par récurrence, nous pouvons établir la relation suivante :

$$R_0 + \sum_{l=1}^{d_m-1} (n_a \times R_l \times n^{l-1}) = \sum_{l=0}^{i-1} (l \times n^l) + n^i \times R_i + \sum_{l=i+1}^{d_m-1} (n_a \times R_l \times n^{l-1})$$

Ainsi :

$$\begin{aligned} N_c &= (\sum_{l=0}^{d_m-3} (l \times n^l)) + n^{d_m-2} \times R_{d_m-2} + (n_a \times R_{d_m-1} \times n^{d_m-2}) \\ &= (\sum_{l=0}^{d_m-2} (l \times n^l)) + R_{d_m-1} \times n^{d_m-1} \\ &= \sum_{l=0}^{d_m} (l \times n^l) \end{aligned}$$

De même pour N_s :

$$\begin{aligned} N_s &= \sum_{i=1}^{d_m} \sum_{l=i}^{d_m} (T_{l-1,i-1} \times n_n \times S_l) \\ &= \sum_{l=1}^{d_m} \sum_{i=1}^{l} (T_{l-1,i-1} \times n_n \times S_l) \\ &= \sum_{l=1}^{d_m} (n_n \times S_l \times \sum_{i=1}^{l} (T_{l-1,i-1})) \\ &= \sum_{l=1}^{d_m} (n_n \times S_l \times \sum_{i=0}^{l-1} (T_{l-1,i})) \\ &= \sum_{l=1}^{d_m} (n_n \times S_l \times n^{l-1}) \\ &= \sum_{l=1}^{d_m} (n_n \times l \times r \times c \times n^{l-1}) \\ &= n_n \times r \times c \times \sum_{l=1}^{d_m} (l \times n^{l-1}) \\ &= n_n \times r \times c \times \sum_{l=0}^{d_m-1} ((l+1) \times n^l) \end{aligned}$$

Le nombre de requêtes envoyées est donc :

$$\begin{aligned} I &= \sum_{l=0}^{d_m} (l \times n^l) + n_n \times r \times c \times \sum_{l=0}^{d_m-1} ((l+1) \times n^l) \\ &= d_m \times n^{d_m} + \sum_{l=0}^{d_m-1} ((l + (l+1) \times n_n \times r \times c) \times n^l) \end{aligned}$$

n^{d_m}	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1
2	2	4	8	16	32	64	128	256	512
3	3	9	27	81	243	729	2187	6561	19683
4	4	16	64	256	1024	4096	16384	65536	262144
5	5	25	125	625	3125	15625	78125	390625	1953125
6	6	36	216	1296	7776	46656	279936	1679616	10077696
7	7	49	343	2401	16807	117649	823543	5764801	40353607
8	8	64	512	4096	32768	262144	2097152	16777216	134217728
9	9	81	729	6561	59049	531441	4782969	43046721	387420489

TABLE 4.1 – Tableau de n^{d_m}

Pour une profondeur limite donnée, la complexité de l'algorithme est polynomiale, de degré d_m. Bien que cette complexité soit considérée raisonnable, elle entraine vite un temps d'analyse important pour des valeurs de d_m importantes. Notons toutefois que n ne représente pas le nombre de nœuds dans l'arbre, mais le degré maximal, c'est-à-dire le nombre maximal d'arcs sortants. Le tableau 4.1 permet de se donner une idée des ordres de grandeur pour différentes valeurs de d_m et n. Il indique également qu'un paramètre à considérer lors des expériences et qui peut parfois être plus important est la rapidité avec laquelle le serveur traite les requêtes.

Très clairement, le nombre de requêtes envoyées est important, même plus important que celui des outils `skipfish` et `W3af`. Par contre, la quantité d'informations déduite de l'analyse du site est plus importante dans notre cas qu'avec ces outils. Effectivement, nous pouvons en déduire les dépendances entre les vulnérabilités et ce type d'informations n'est possible à obtenir qu'en augmentant les interactions avec le site pour identifier tous les cas de figure permettant d'exploiter la vulnérabilité ou en se fixant des hypothèses sur les dépendances entre classes de vulnérabilités.

Le nombre d_m correspond au nombre de liens contenu dans chaque page. Dans un site réel, ce nombre diffère d'une page à l'autre. Sur certains *blogs*, il peut être "faible" et sur certains *wiki*, il peut largement dépasser la centaine. Dans la pratique, les sites rencontrés s'éloignent fortement de ce pire cas mais localement un d_m élevé reste pénalisant. Dans la section suivante, nous présentons une approche pour limiter ce problème et, comme le montre la dernière section, les tests indiquent que nous pouvons obtenir des résultats intéressants dans des temps raisonnables.

4.5.6 Maîtrise de la taille du graphe – similarité des requêtes

Les difficultés rencontrées lors de la phase de crawling sont similaires à celles du problème du voyageur de commerce : une recherche exhaustive peut être très longue à réaliser. Pour illustrer ce problème dans notre cas, nous pouvons considérer un forum permettant aux utilisateurs d'ajouter des articles. A chaque fois qu'un article est ajouté, un nouveau lien est ajouté à une des pages pour accéder à cet article. Globalement, les requêtes permettant d'accéder aux différents articles peuvent se ressembler en se différenciant uniquement au niveau de l'identifiant de l'article. Par contre, leur nombre peut être infini. Donc, même si nous avons établi une borne pour limiter la profondeur de recherche, le nombre de requêtes à envoyer peut rester trop important. Dans ce cas précis, si une vulnérabilité est présente dans la fonctionnalité d'enregistrement de l'article, voire dans la fonctionnalité d'affichage, il est fortement probable qu'elle ne dépende pas du numéro de l'article. D'ailleurs, la plupart des exploits pour les vulnérabilités des différents sites correspondent à l'exécution d'un nombre réduit de requêtes.

Ces remarques nous permettent de conclure qu'il est inutile de poursuivre le crawling, depuis un état en considérant toutes les requêtes permettant d'accéder aux différents articles. Le crawling depuis cet état en considérant une seule de ces requêtes est suffisant. Il est donc important de pouvoir établir une notion de similarité entre les requêtes. Dans cet exemple simple, deux requêtes sont similaires si elles correspondent à des accès à des articles. Dans le cadre plus général, il est impossible d'établir cette notion sans un minimum de connaissances sur le site. Aussi, nous reportons le problème du côté de l'utilisateur de notre algorithme, qui aura pour tâche de naviguer sur le site et de définir lui même les critères d'équivalence entre requêtes.

L'utilisateur doit tout de même établir cette notion de similarité soigneusement. Si elle est trop restrictive, nous ne réduisons pas le problème lié à l'exhaustivité. Si elle est trop englobante, nous ne considérerons pas des navigations qui nous amèneraient dans de nouveaux états.

4.5.7 Expérimentation

Afin d'analyser la complexité étudiée, nous avons exécuté l'algorithme proposé sur notre exemple de site Web avec des valeurs de seuil de profondeur différentes. Nous avons paramétré *Wasapy* à 20 requêtes pour chacune des classes R_{iv}, R_{ii}, R_a afin de tester les injections SQL pour toutes les profondeurs considérées.
Les résultats sont présentés dans le tableau 4.2. Pour ces résultats, nous avons utilisé un ordinateur MacOSX (Core2 Duo 2Go de RAM) pour exécuter l'algorithme et un ordinateur de bureau Linux (processeur CPU Core2 Duo) pour le serveur sous test. Nous remarquons que plus on augmente la valeur du seuil plus la durée d'exécution augmente. Cependant, le graphe résultant réduit peut ne pas changer de façon significative (voir, par exemple, les seuils de 4, 5 et 6).

L'augmentation du seuil à $d + 1$ ne permet pas nécessairement d'atteindre de nouvelles pages ou de nouvelles vulnérabilités qui ne peuvent pas être atteintes avec un seuil d. Le choix du seuil dépend du site. L'expérience empirique montre qu'un seuil d'environ 8 est généralement suffisant et permet

Seuil de profondeur	2	3	4	5	6	7	8
Nœuds	2	5	5	6	6	9	9
Arcs	4	6	9	11	11	20	20
Vulnérabilités	1	1	2	2	2	4	4
Durée	7s	10s	15s	25s	57s	3m	10m37s

TABLE 4.2 – Résultats expérimentaux

d'accéder à toutes les pages de la plupart des sites Web.
La durée d'exécution de l'algorithme augmente selon le seuil. Pour notre exemple, le temps d'exécution reste raisonnable. L'augmentation de temps d'éxecution quand le nombre de noeuds et d'arcs ne change pas est dû à l'éxcecution de boucles dans le graphe.

Nous présentons dans les figures 4.11, 4.12, 4.13 et 4.14 les graphes finaux pour les 4 dernières profondeurs testées.

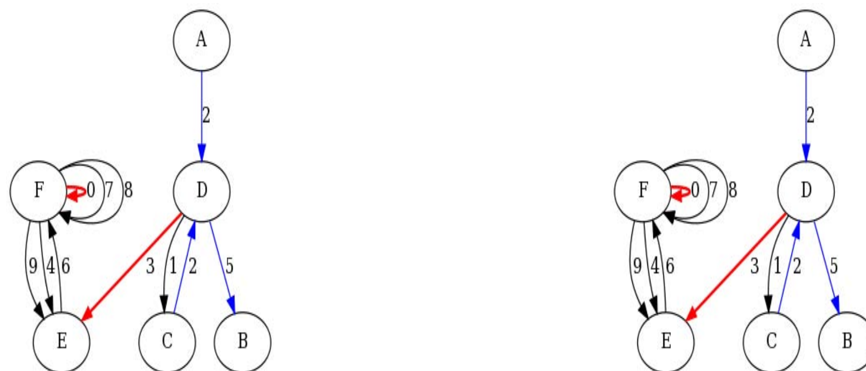

FIGURE 4.11 – Graphe final pour la profondeur 5

FIGURE 4.12 – Graphe final pour la profondeur 6

Nous avons également exécuté notre algorithme sur l'application *Riotpix* que nous avons présentée dans la section . Même si cette application reste relativement simple, elle est toutefois plus complexe que notre propre exemple et est bien représentative des applications que l'on rencontre sur le Web aujourd'hui. Le seuil de profondeur a été fixé à 6. Nous présentons les différents graphes résultant des itérations successives dans la figure 4.15.

Le test a duré 3 min. Nous remarquons que l'exécution de l'algorithme s'est arrêtée à la 3me itération car, entre le 2me et le 3me graphe, il n'y a pas eu de changement, ni au niveau des nœuds, ni au niveau des arcs. Il y a eu détection de 3 vulnérabilités d'injection SQL dans les pages `index.php`,

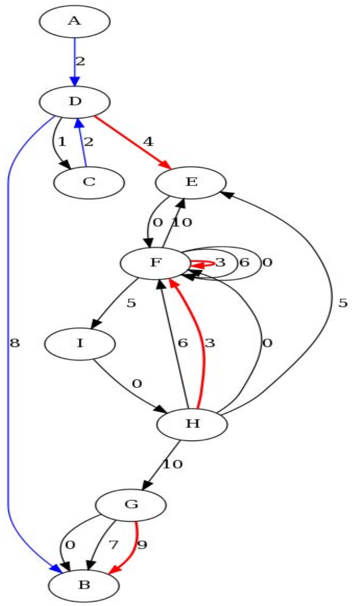

FIGURE 4.13 – Graphe final pour la profondeur 7

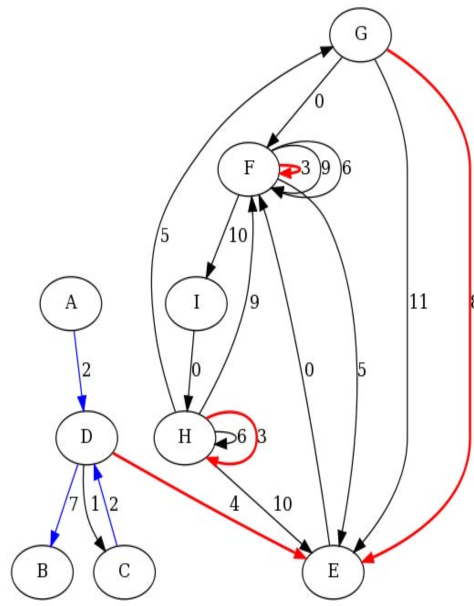

FIGURE 4.14 – Graphe final pour la profondeur 8

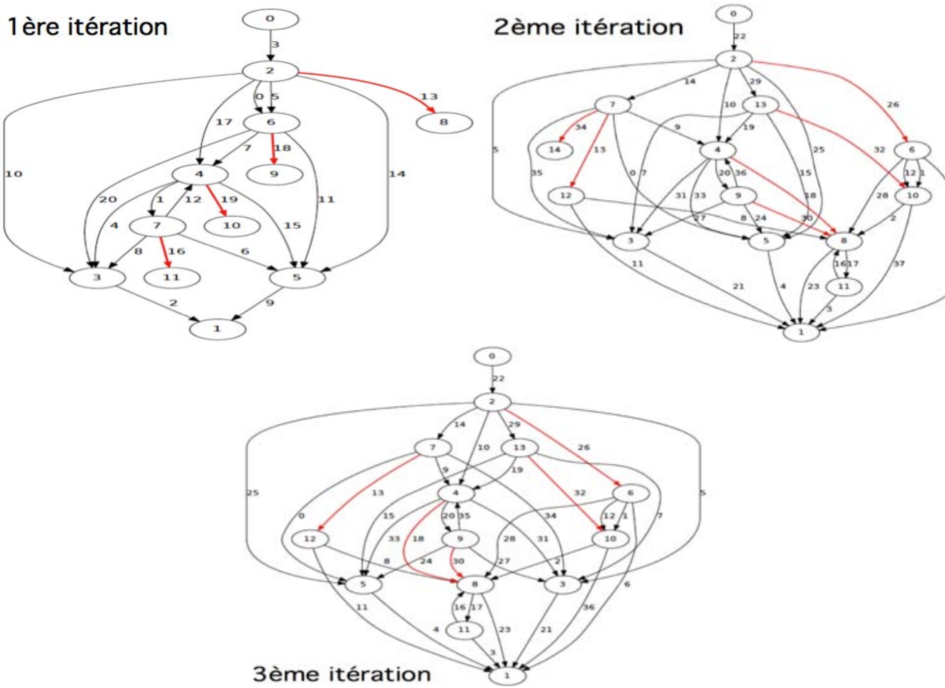

FIGURE 4.15 – Graphes de navigation des 3 itérations de l'application *Riotpix*

`message.php` et `read.php`. Il n'y a pas de de scénario d'attaque mettant en évidence des dépendances entre plusieurs vulnérabilités puisque ce site ne s'y prête pas. L'expérience est ici avant tout

destinèe à avoir un aperçu des performances de l'algorithme d'élaboration du graphe de navigation. En cela, le résultat est encourageant.

4.6 Conclusion

Dans ce chapitre, nous avons proposé une nouvelle méthodologie visant à identifier des scénarios d'attaque ciblant les applications Web basée sur l'analyse dynamique de l'application en suivant une approche boîte noire. Notre méthodologie permet de présenter automatiquement des dépendances causales entre les vulnérabilités et à identifier les scénarios d'attaque en exploitant d'une manière ordonnée ces vulnérabilités. Cette approche a été illustrée sur deux exemples d'applications Web. Nous avons également étudié la complexité de notre algorithme qui montre que les performances sont acceptables pour des sites de complexité raisonnable. Le nombre de requêtes générées peut néanmoins s'avérer élevé pour des sites très complexes. Nous avons aussi montré que la taille du graphe de navigation généré par notre algorithme peut être réduite en exploitant la similarité qui existe souvent entre différentes classes de requêtes. Cette piste mérite d'être explorée davantage pour améliorer les performances de notre algorithme.
Les scénarios d'attaques résultants de l'exécution de notre algorithme forment le trafic malveillant qui sera intégré par la suite dans l'ensemble de la plateforme d'évaluation que nous détaillons dans le chapitre suivant.

5

Plateforme d’évaluation

Introduction

Dans ce chapitre, nous présentons la conception et la mise en œuvre d’une plateforme d’évaluation des systèmes de détection d’intrusion. Dans la section 5.1 qui suit cette introduction, nous décrivons l’architecture de cette plateforme expérimentale, en précisant nos choix de conception. Puis nous détaillons le rôle des composants de l’architecture, et nous donnons quelques éléments concernant leur implémentation dans la section 5.2. L’intégration de ces différents éléments au sein de la plateforme est présentée dans la section 5.3. Afin d’évaluer les performances de la plateforme, nous avons réalisé un ensemble d’expérimentations, que nous décrivons dans la section 5.4. Enfin, la section 5.5 conclut ce chapitre.

5.1 Architecture de la plateforme d'évaluation

Afin de pouvoir réaliser une plateforme d'évaluation de systèmes de détection d'intrusion efficace, il est nécessaire qu'elle inclue des outils capables de générer du trafic d'attaque, du trafic sain, des outils capables de collecter les alertes émises par les systèmes de détection d'intrusion et des outils capables d'élaborer des rapports incluant un certain nombre de mesures (notamment les taux de faux positifs et de faux négatifs). C'est donc avec ces contraintes à l'esprit que nous avons élaboré notre plateforme d'évaluation [AKROUT ET AL. 12A].

L'architecture globale de notre plateforme d'évaluation est représentée dans la figure 5.1. Elle est composée de quatre machines :

- la machine cible sur laquelle s'exécute l'application Web et l'IDS associé.
- la machine d'attaque sur laquelle va se dérouler l'essentiel de notre méthodologie d'évaluation.
- la machine génératrice de trafic normal.
- la machine d'analyse.

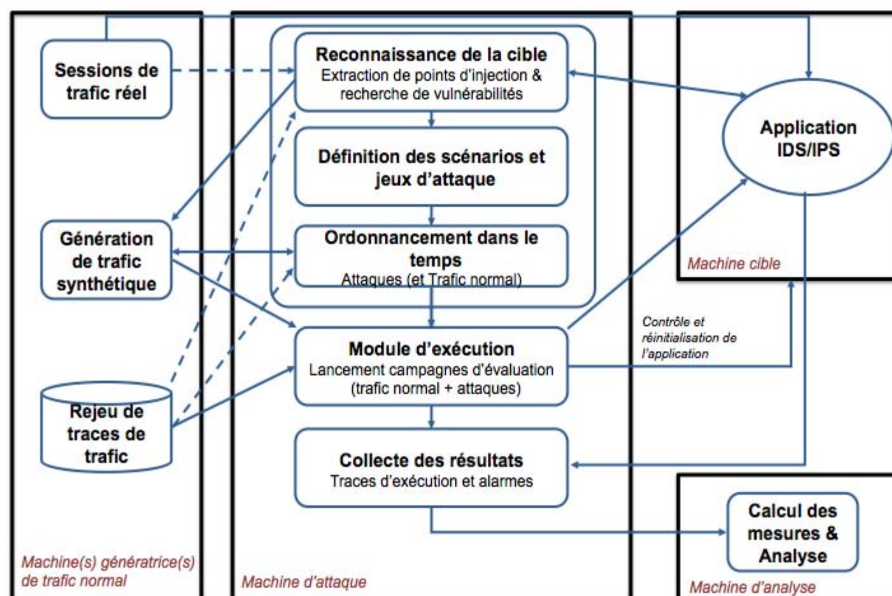

FIGURE 5.1 – Architecture initiale de la plateforme d'évaluation

Plusieurs grandes phases peuvent être identifiées lors de l'évaluation. La première phase est la phase de reconnaissance et elle consiste à explorer la cible et à en extraire les points d'injection. La deuxième phase est la phase d'identification de vulnérabilités. Cette phase a été décrite en détail dans le chapitre 3. Ensuite il convient d'identifier les relations de dépendance entre les différentes vulnérabilités existantes pour générer des scénarios d'attaque tels que décrits en détail dans le 4. Ces scénarios d'attaque constituent le trafic malveillant destiné à être envoyé à la cible. Cependant, il est nécessaire de le mélanger à du trafic normal de façon à reproduire aussi fidèlement que possible les conditions d'utili-

sation réelles d'un serveur Web. Pour la génération du trafic normal, différentes méthodes peuvent être envisagées :

- le rejeu de traces collectées lors d'utilisations "normales" de l'application ;
- le trafic synthétique généré à partir d'un modèle de l'application ;
- l'utilisation de l'application Web en temps réel.

Le module d'ordonnancement a pour objectif de définir les caractéristiques temporelles déterminant les instants et l'ordre d'envoi des requêtes faisant partie d'un trafic normal et d'un trafic d'attaques.

L'ensemble du trafic normal et du trafic malveillant est ensuite envoyé vers l'application vulnérable par la machine d'attaque conformément à la stratégie d'ordonnancement ainsi définie.

Après ces trois étapes, les traces d'exécution et les alertes des systèmes de détection d'intrusion sont collectées et envoyées à une machine d'analyse pour la dernière phase destinée à interpréter les résultats obtenus pour produire des mesures (par exemple, taux de faux positifs et faux négatifs) et présenter ces mesures.

La figure 5.2 décrit une implémentation détaillée basée sur l'architecture présentée dans la figure 5.1. Cette implémentation a été mise en œuvre dans le cadre de notre travail. Nous décrivons brièvement ici le rôle de chacun de ces éléments. Nous les présentons plus en détails dans la section suivante.

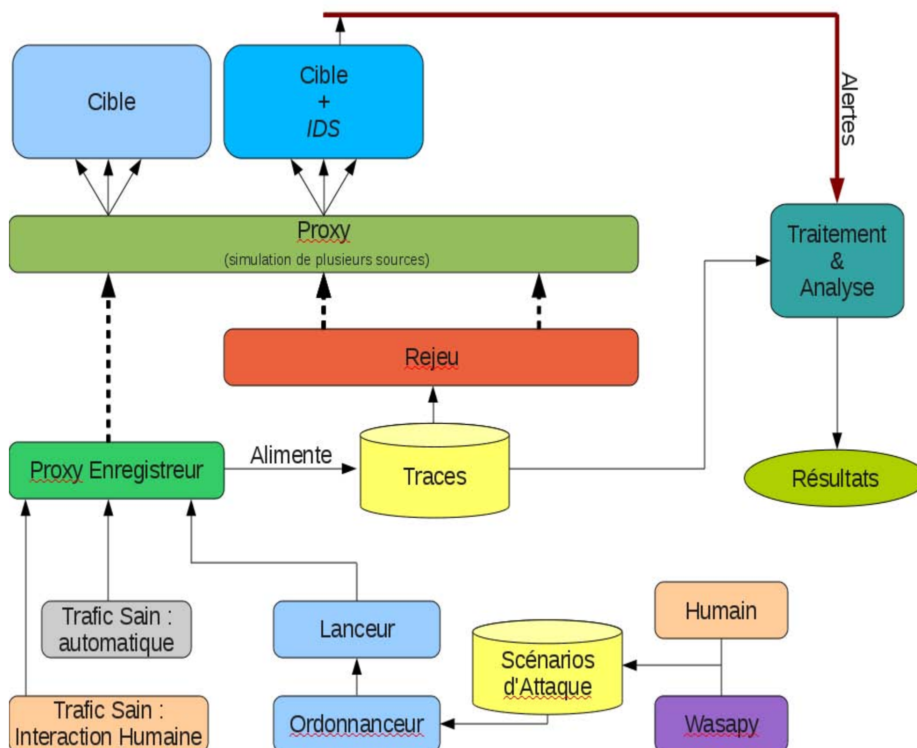

FIGURE 5.2 – Architecture détaillée de la plateforme d'évaluation

En pratique, une expérimentation se déroule en deux temps. Dans un premier temps, du trafic sain et malveillant sont envoyés à la cible sans activation des mécanismes de détection d'intrusion. Le trafic sain est généré soit de façon automatique, par rejeu de traces ou par synthése (*Trafic Sain : automatique*) ou manuellement (*Trafic Sain : interaction humaine*). Le trafic malveillant est généré à partir de l'élaboration de *scénarios d'attaque*, fournis automatiquement par *Wasapy* où construits manuellement (*Humain*). Ces scénarios d'attaque sont exécutés par le *lanceur*, selon une stratégie d'ordonnancement gérée par l'*ordonnanceur*.
L'ensemble du trafic sain et malveillant est enregistré par un *proxy enregistreur* dans un ensemble de *Traces*, ce qui nous permet de pouvoir rejouer la même campagne d'évaluation autant de fois que nous le souhaitons, grâce au module *Rejeu*.

Dans un second temps, nous rejouons donc l'ensemble de ce trafic sur la même cible, mais en présence de mécanismes de détection d'intrusion (*Cible + IDS*). En amont des cibles, un *Proxy* permet de simuler plusieurs sources de trafic.
Les alertes des différents systèmes de détection d'intrusion sont relevées et analysées (*Traitement & Analyse*). Grâce aux enregistrements, l'origine de chaque alerte peut être déterminée. Ceci nous permet de calculer différentes métriques telles que les faux positifs, faux négatifs, vrai positifs et vrai négatifs (*Résultats*). Finalement, une interface graphique permet de commander la plateforme.
Nous détaillons dans la section suivante le développement de chacun de ces éléments.

5.2 Présentation des différents composants de la plateforme

Dans cette partie nous présentons en détails les différentes composants de la plateforme d'évaluation présentés dans la figure 5.2. Pour chacun de ces composants, nous expliquons la démarche qui nous a amenés à décider son intégration dans la plateforme, ainsi que les choix d'implémentation que nous avons faits.

5.2.1 Génération de trafic normal

Pour évaluer des systèmes de détection et de prévention d'intrusion, il est évidemment important d'analyser leur comportement vis-à-vis d'un trafic malveillant, mais également vis-à-vis d'un trafic sain, pour lequel il ne devrait pas lever d'alerte. Ce trafic sert donc essentiellement à l'évaluation du taux de faux positifs. Nous présentons dans cette section les différents moyens qui assurent la génération du trafic normal.

5.2.1.1 Interaction humaine et crawler web

Il est difficile de définir manuellement toutes les différentes manipulations pouvant être effectuées sur un site. Nous avons donc décidé d'utiliser un robot capable de naviguer à travers un site Web. Il existe différents types d'outils réalisant cette tâche, notamment les crawlers Web. Ces crawlers découvrent les pages Web du site, les enregistrent et les indexent selon différents critères propres à chacun d'entre eux.

De nombreux crawlers Web existent aujourd'hui. Nous avons sélectionné plusieurs outils existants et nous les avons testés par rapport à nos besoins. Nous présentons dans ce qui suit deux exemples de ces outils.

JCrawler [1]

`JCrawler` est un outil développé spécialement pour tester la tenue en charge et simuler le plus possible des visites humaines sur un site. Sa configuration se fait très intuitivement par un fichier XML. Malheureusement, ses options sont très limitées. Mises à part quelques temporisations et les entêtes à utiliser, aucune option supplémentaire n'est implémentée dans l'outil. De plus il ne peut pas être configuré pour franchir avec succès des authentifications sur un site Web.

HTTrack [2]

`HTTrack` est un outil destiné à faire une copie locale d'un site Web distant. Pour cela, il construit récursivement tous les répertoires, en téléchargeant les fichiers HTML, les images et autres fichiers du serveur. `HTTrack` réorganise la structure des liens de façon relative ; il peut aussi mettre à jour un site existant, ou continuer un téléchargement interrompu. Le robot est entièrement configurable, avec un système d'aide intégré.

Comparé à `JCrawler`, `HTTrack` possède plus d'options de configuration et permet un meilleur apprentissage du site à crawler. Nous pouvons par exemple lui spécifier le "login" et le "mot de passe" de certaines pages afin qu'il puisse s'authentifier automatiquement. Cependant, ceci fonctionne uniquement sur des authentifications de type HTTP. Ce mécanisme d'authentification est cependant très peu utilisé sur les sites dits "Web 2.0".

L'analyse de ces deux crawlers nous a permis de conclure qu'ils ne répondent pas suffisamment à nos besoins. En effet, ces deux outils ne sont pas suffisamment paramétrables, notamment en ce qui concerne les pages d'authentification que ces applications ne pourront jamais franchir. Ainsi, aucun ne permet de parcourir concrètement des pages nécessitant une authentification, ni de remplir et de valider des formulaires.
Ces deux outils sont donnés à titre d'exemple mais leur analyse est représentative des défauts que nous avons identifiés sur l'ensemble des crawlers Web que nous avons pu rencontrer. C'est la raison pour laquelle nous avons donc décidé de développer nous-même ce générateur de trafic que nous détaillons dans la suite de cette section.

5.2.1.2 Crawler générateur automatique du trafic normal

Notre objectif est de parcourir automatiquement le site cible et d'en assurer une couverture maximale. Ceci implique que le crawler doit être capable de franchir des pages d'authentification. Il est préférable que l'outil ne s'arrête pas à une itération du site, et assure des visites aléatoires pendant une

1. `http://jcrawler.sourceforge.net/`
2. `http://www.httrack.com/`

durée définie sur le site. De plus, nous souhaitons que l'outil ait la possibilité d'offrir plusieurs accès concurrents.

Sans rentrer dans des détails techniques, l'outil fonctionne de la manière illustrée dans la figure 5.3. En partant de l'URL principale du site défini, le crawler charge la page correspondante et extrait tous les liens de type *a href* et tous les formulaires de type *form*. Puis, à partir de l'ensemble des liens récupérés, l'un d'entre eux est choisi aléatoirement. S'il s'agit d'un formulaire, le crawler extrait et remplit tous les champs présents dans le formulaire (remplissage de champ text, choix dans un menu défilant, cases à cocher, etc.). Enfin, il itère et il reprend à la phase du chargement de la page avec les nouvelles données calculées.

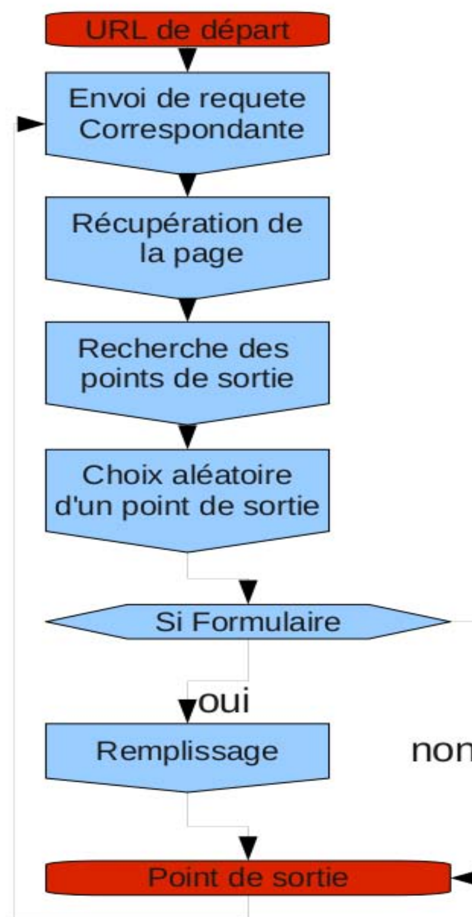

FIGURE 5.3 – Algorithme du crawler générateur du trafic normal

Notre outil est conçu pour remplir automatiquement des formulaires. Ceci lui permet de franchir, par exemple, des pages d'authentification. Ceci ne peut fonctionner que si l'outil a connaissance de certaines données. Pour cela, il est possible de spécifier les éléments suivants dans un fichier de configuration :

– Longueur moyenne et écart type des données à remplir (caractères aléatoires dans [a-zA-Z0-9])
– Champs, que nous identifions par leur nom, à remplir de manière identique (certaines inscriptions nécessitent de fournir deux fois le même mot de passe), ce comportement peut être activé, désactivé ou aléatoire.
– Champ à remplir comme une adresse mail
– Champ à remplir avec des chiffres uniquement
– Champ à remplir avec des lettres uniquement
– Des réponses prédéfinies : on peut identifier un formulaire par sa déclaration (contenu de la première balise), et spécifier les réponses à utiliser pour les différents champs. Ceci nous permettra entre autres de franchir les étapes d'authentification.

5.2.2 Trafic d'attaque : Scénarios d'attaque, Ordonnanceur, Lanceur

Ces trois éléments de la plateforme permettent de définir, d'échelonner et d'exécuter les scénarios d'attaque dont l'élaboration a été présentée dans le chapitre 4.

5.2.2.1 Scénario d'attaque

Les scénarios d'attaque sont constitutés de requêtes coordonnées, envoyées vers un site Web dans le but de réaliser une action illégitime. Il est important de signaler que parmi l'ensemble de ces requêtes, il est tout à fait possible que certaines ne contiennent pas d'exploitation de vulnérabilité et qu'elles constituent donc des requêtes saines.
Un scénario d'attaque peut donc contenir à la fois des requêtes saines et des requêtes d'attaques.
Par exemple, si un attaquant désire lire une information confidentielle accessible depuis la page privée de l'administrateur d'un site Web, il peut à l'aide d'une injection SQL (et donc d'une requête d'attaque) contourner l'authentification pour accéder à cette page. Ensuite, la requête consistant à accèder à un document confidentiel à partir de cette page n'est pas une requête d'attaque puisqu'elle s'exécute légitimement avec les privilèges administrateur. Ainsi le scénario d'attaque constituant à voler une information confidentielle est formée de deux requêtes, une requête d'attaque et une requête saine.
Un scénario sain, quant à lui, ne contient que des requêtes saines. La figure 5.4 illustre ces deux types de scénarios. Il est rappelè que l'èlaboration des requêtes permettant l'identification et l'exploitation de vulnèrabilitès dans l'application sous test est effectué avec notre outil *Wasapy*, conformément à la mèthodologie décrite dans le chapitre 3.

Ces scénarios, sains ou d'attaque, sont décrits dans un fichier au format XML présenté dans la figure 5.5. Ce formalisme a été conçu pour être le plus exhaustif possible.

Dans ce fichier de configuration de scénario, nous définissons le type du scénario (scénario d'attaque ou scénario sain). Nous listons également l'ensemble des requêtes envoyées en précisant leur méthode (POST/GET), leur url, et leur type (attaque/saine), leur adresse IP et l'ensemble des données passées en paramètre.
Pour mieux comprendre le format du scénario, la figure 5.6 présente un exemple d'un scénario rensei-

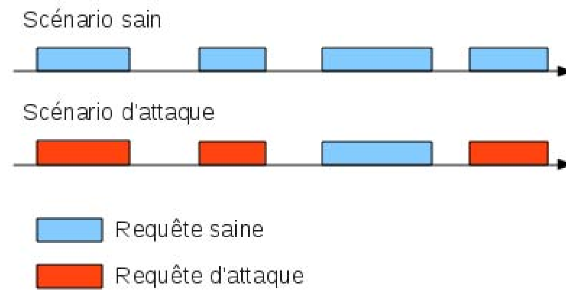

FIGURE 5.4 – Illustration des deux scénarios

```
<scenarios>
  <scenario type="x" id="xx">
    <descriptif></descriptif>
    <requete id="xx" methode="xx" url="xx" type="x" wait="x" ip="x">
      <data name="xx">valeur</data>
    </requete>
  </scenario>
</scenarios>
```

FIGURE 5.5 – Format des scénarios

gné que nous avons utilisé pour notre expérimentation. Dans ce fichier, nous disons qu'il s'agit d'un scénario d'attaque et nous présentons les différentes requêtes lancées ('id=0' et 'id=1') en précisant les données ('sarah' et 'ghost').

5.2.2.2 Ordonnanceur

L'ordonnanceur (représenté dans la figure 5.2) doit définir l'ordre d'exécution des différents scénarios d'attaque ainsi que des scénarios sains et les répartir sur les différentes sources. Il prend en entrée trois fichiers, un fichier de configuration, un fichier décrivant les scénarios d'attaque et un fichier décrivant les scénarios sains et il produit en sortie une suite d'instructions utilisables par le module d'exécution (*Lanceur* dans la figure 5.2).
La figure 5.7 représente les différentes durées pour une campagne d'évaluation. Les symboles utilisés dans cette figure sont précisés ci-dessous.

- **d**=[t_0,t_{final}] : la durée totale d'une campagne d'évaluation (de t_0 à t_{final}), elle représente également un critère d'arrêt.
- **a** : la loi et les paramètres du temps de démarrage (ex. loi gaussienne de moyenne x et écart 2).

```
<scenarios>
    <scenario type="A" id="01">
        <descriptif></descriptif>
        <requete id="01" methode="POST" url="http://localhost:3000/" type="A" wait="1" ip="2.2.2.2">
            <data name="login">sarah</data>
            <data name="password">ghost</data>
        </requete>
        <requete id="02" methode="GET" url="http://localhost:3000/" type="A" wait="2" ip="3.3.3.3">
            <data name="msg">30</data>
        </requete>
    </scenario>
</scenarios>
```

FIGURE 5.6 – Exemple de définition d'un scénario

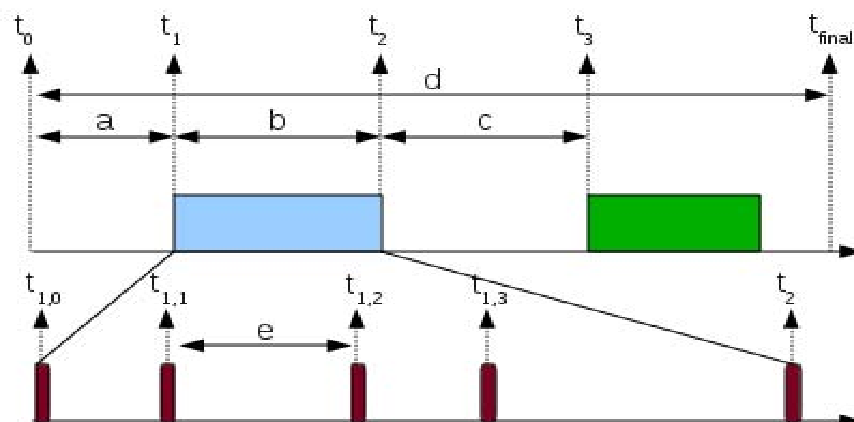

FIGURE 5.7 – Caractéristiques d'une campagne d'évaluation

– **b** : la durée moyenne d'un scénario avec écart.
– **c** : la loi et les paramètres du temps entre chaque scénario (ex. loi linéaire avec min et max).
– **e** : la loi et les paramètres du temps entre deux requêtes dans un scénario.

Ces durées peuvent être configurées à partir du fichier de configuration propre à l'ordonnanceur (cf. Figure 5.8). D'autres paramètres peuvent également être renseignés :

– Le nombre de scénarios que chaque “source” essaie d'exécuter (min et max après la balise `<repetition>`)
– Le ratio entre scénarios sains et scénarios d'attaque (balise `<ratio>`)
– La durée totale de la campagne d'évaluation (balise `<duree>`) ; elle correspond à la lettre **d** définie ci-dessus.
– La durée pour démarrer la campagne, définie par la balise (balise `<demarrage>`) ; elle cor-

respond à la lettre **a** définie ci-dessus.

- La durée entre deux scénarios successive (balise `<entrescen>`) ; elle correspond à la lettre **c** définie ci-dessus.
- La durée entre deux requêtes dans un même scénario (balise `<entrereq>`) ; elle correspond à la lettre **e** définie ci-dessus.
- Le nombre de "sources" (c'est-à-dire, les machines émettrices de trafic) et leurs adresses IP respectives (balise `<poolip>`)

```
<config>
        <repetitions min="1" max="1" />
        <ratio val="1" />
        <duree moy="15" ecart="3" />
        <plage val="180" />
        <demarrage moy="0" ecart="0" loi="gauss"/>
        <entrescen moy="5" ecart="2" loi="gauss"/>
        <entrereq ecart="2" loi="gauss"/>
        <poolip>
                <ip id="1">127.0.0.1</ip>
                <ip id="1">127.0.0.2</ip>
        </poolip>
</config>
```

FIGURE 5.8 – Fichier de configuration de l'ordonnanceur

En sortie l'ordonnanceur produit une liste de requêtes à exécuter. La figure 5.9 donne un exemple de sortie de l'ordonnanceur. Chaque élément de cette liste est construit dans cet ordre :

- le type de scénario : **A**ttaque ou **S**ain
- l'identifiant du scénario
- l'identifiant de la requête du scénario
- le type de requête : **A**ttaque ou **S**ain
- l'instant d'exécution de la requête
- l'adresse IP de la machine émettrice de la requête

Chaque ligne de sortie de l'ordonnanceur est unique et permet de reconnaître l'origine de chaque requête. Soulignons que nous avons décidé de ne pas intégrer directement l'ordonnancement dans le module d'exécution. Ceci permet tout d'abord d'obtenir des outils bien identifiables et éventuellement, en cas de problème, de mieux analyser la source du problème. Le module d'exécution que nous présentons dans la section suivante a donc toujours besoin de la description des scénarios pour fonctionner.

```
A_01_01_A_1_127.0.0.1
A_01_02_A_3_127.0.0.1
S_01_01_S_10_127.0.0.2
S_01_02_S_21_127.0.0.2
```

FIGURE 5.9 – Sortie de l'ordonnanceur

5.2.2.3 Le module d'exécution (*lanceur*)

Le *lanceur* doit simplement utiliser les sorties de l'ordonnanceur et exécuter les différentes requêtes. Afin de respecter l'indépendance des différentes sources, un thread est créé pour chacune d'entre elles. Chaque thread gère alors un accès indépendamment des autres au site Web avec notamment son propre gestionnaire de cookies.
Le module d'exécution prend en entrée deux ou trois fichiers : le fichier de sortie de l'ordonnanceur qui présente l'ordre d'exécution des différentes requêtes, les scénarios d'attaque et les scénarios sains.

Le formalisme de ces différents fichiers a été traité dans la section 5.2.2.2. A l'émission de chaque requête, le lanceur insère l'identifiant (défini dans le fichier de sortie de l'ordonnanceur) dans le paramètre "user-agent" de la requête. Ceci permet d'identifier chaque requête, mais permet également d'effectuer des modifications sur la requête (cf. 5.2.4).
Le trafic que nous utilisons pour nos évaluations doit avoir un comportement le plus fidèle possible à la réalité. Pour cela il est important que ces outils se comportent de la même manière qu'un explorateur Web normal. Il est nécessaire que les opérations telles que les redirections soient effectuées. Il est également important que l'outil soit capable de simuler plusieurs accès concurrents, afin d'accroître le réalisme de parcours du site. Pour répondre à ces besoins, nous avons développé des modules spécifiques que nous présentons dans la section suivante.

5.2.3 Proxy enregistreur, et rejeu de traces WEB

Il est important d'assurer que l'évaluation de chaque mécanisme de protection soit faite dans les mêmes conditions. Nous avons donc décidé d'effectuer la première évaluation sur la cible sans mécanisme de protection tout en l'enregistrant. Cet enregistrement nous permet par la suite d'évaluer plusieurs mécanismes de protection dans les mêmes conditions.
Un outil d'enregistrement se base sur l'interaction d'un utilisateur avec le site Web, de manière graphique ou pas. De plus, pour exécuter les traces, un enregistreur exécute réellement l'explorateur Web de notre choix (tels que Firefox, Internet Explorer ou Chrome).
L'enregistrement peut se faire par également le biais d'un plugin développé spécialement pour les explorateurs web. Le principe de l'enregistrement est assez simple pour tous les outils, il suffit de l'activer au début de l'enregistrement et de l'arrêter au moment voulu. Pendant ce temps, l'outil enregistre toutes les actions effectuées sur le site. Nous avons étudié des outils existants capables d'enregistrer, d'ordon-

nancer et d'exécuter les scénarios et nous présentons un aperçu sur chacun dans ce qui suit.

Selenium [3]

Selenium est une suite d'applications capables d'enregistrer et de rejouer des traces Web. Les traces utilisées par *Selenium* sont de haut niveau. Lors du rejeu, cet outil mène les actions telles qu'il les a enregistrées. Le rejeu peut se faire de plusieurs manières.

- La première consiste à utiliser le plugin de l'explorateur Web qui a servi à l'enregistrement des traces. Même si c'est la manière la plus évidente, elle est assez contraignante dans le cadre d'une automatisation.
- La deuxième est l'utilisation de *Selenium RC* (Remote Control). *Selenium RC* est un serveur capable d'ouvrir une session dans un explorateur Web et d'y rejouer des traces web. Cette solution est bien adaptée dans le cadre d'une automatisation du fait que des API pour différents langages existent et notamment pour Python.

Le besoin d'un explorateur Web et d'un environnement d'exécution JAVA fait de *Selenium* une application très lourde et peu configurable. De plus il n'est pas possible de spécifier, parmi les traces enregistrées, celles que l'on veut rejouer. Toutes sont par défaut rejouées.

JMeter [4]

C'est une application destinée à tester la performance de plusieurs types d'applications (Sites Web, Annuaires et Messageries). En ce qui concerne les tests sur des sites Web, *JMeter* utilise des traces de plus bas niveau que *Selenium*. Il s'agit de traces au niveau protocole HTTP.

JMeter nous permet de réaliser des scénarios de tests très paramétrables. Il est tout à fait possible de répéter de manière aléatoire et pondérée différents scénarios. L'outil est même conçu pour simuler différents utilisateurs, cette fonctionnalité présente malheureusement encore des bugs. En effet, lors de nos tests, nous avons cherché à exploiter cette fonctionnalité, mais on s'est rapidement aperçu que les différentes sessions n'étaient pas strictement isolées.

WebInject [5]

C'est un outil permettant uniquement le rejeu de scénarios définis statiquement. Il permet également de tester si le résultat obtenu est correct en testant le contenu de la page obtenue ou son code de retour.
WebInject précise dans sa documentation qu'elle ne supporte pas les redirections. Cette limitation ne nous permettra pas de franchir la majorité des authentifications. De plus cet outil n'est pas prévu pour répéter de manière aléatoire les mêmes scénarios sur plusieurs utilisateurs.

Nos tests montrent qu'aucun des outils testés ne répond totalement à nos besoins techniques.

---

3. `http://seleniumhq.org/`
4. `http://jakarta.apache.org/jmeter/`
5. `http://www.webinject.org/`

Par conséquent, nous avons opté pour un proxy intermédiaire enregistrant toutes les requêtes. Pour cela, nous avons décidé de reprendre un proxy développé par *SUZUKI Hisao* et modifié par *Mitko Haralanov* : *Tiny HTTP Proxy*[6]. Étant donné la simplicité du code il a été assez facile de l'adapter à nos besoins. Nous présentons ces adaptations dans le paragraphe suivant. Ces adaptations consistent essentiellement à adapter le fichier de sortie de ce logiciel à nos besoins.

Adaptation de *Tiny HTTP Proxy*

A chaque requête transmise, le proxy écrit les données utilisées dans un fichier, sans les mettre en forme. La Figure 5.10 en contient un extrait. Ce fichier liste l'ensemble des informations des requêtes émises sous cette forme :

#numéro de la requête#identifiant#url#données.

Un deuxième outil permet de transformer cette sortie dans un format proche de celui décrit pour l'ordonnanceur et le module d'exécution (cf. 5.2.2.2). La Figure 5.11 montre la transformation de l'exemple donné. Nous pouvons noter la ressemblance avec le format d'un scénario dans une campagne d'évaluation décrit dans la figure 5.5. Rappelons qu'il s'agit d'un fichier XML dont les balises et leurs paramètres servent à présenter les différentes informations de la requête (identifiant, méthode, url, données).

```
GET#1#1_1_1#/#localhost:3637#
GET#2#1_2_2#/products/filter#localhost:3637#filter%5Bcategory%5D=7
GET#3#1_3_3#/signup#localhost:3637#
POST#4#1_4_4#/signup#localhost:3637##user%5Bcolor%5D=TxyFS8ivYW&user%5Bname%5D=WYSN6D&user%5B
password%5D=r0EJvXjg1&user%5Blogin%5D=GPZS46R&user%5Bpassword_confirmation%5D=r0EJvXjg1&user%5B
email%5D=DnN7MY%40xqG2SHU.UP&authenticity_token=41PlgofX%2F0EDlL1VtZQ2v4XCHtJfAcJY%3D
GET#5#1_4_5#/user/home#localhost:3637#
```

FIGURE 5.10 – Sortie du proxy enregistreur non mis en forme

Après l'étape d'enregistrement, nous avons mis en place un moteur de rejeu capable de rejouer les traces enregistrées lorsque la campagne a été lancée sur la première cible. Il est important que cet outil respecte l'ordre d'exécution des différents requêtes.

5.2.4 Proxy : simulation de plusieurs sources

Afin de rendre le plus réaliste possible le trafic envoyé sur la cible, il doit provenir de différentes sources. Pour la cible, cela se traduit par la réception de paquets réseau provenant d'adresses IP différentes. Il n'y a donc pas besoin d'utiliser plusieurs machines. Il suffit d'utiliser plusieurs cartes réseau dans une machine, ou encore plusieurs adresses IP sur une même carte réseau de la même machine.

6. `http://www.voidtrance.net/2010/01/simple-python-http-proxy`

```
<requete id="1" methode="GET" url="http://localhost:3637/">
        <user_agent>1_1_1</user_agent>
</requete>
<requete id="2" methode="GET" url="http://localhost:3637/products/filter">
        <user_agent>1_2_2</user_agent>
    <data name="filter%5Bcategory%5D">7</data>
</requete>
<requete id="3" methode="GET" url="http://localhost:3637/signup">
        <user_agent>1_3_3</user_agent>
</requete>
<requete id="4" methode="POST" url="http://localhost:3637/signup?">
        <user_agent>1_4_4</user_agent>
    <data name="user%5Bcolor%5D">TxyFS8ivYW</data>
    <data name="user%5Bname%5D">WYSN6D</data>
    <data name="user%5Bpassword%5D">r0EJvXjg1</data>
    <data name="user%5Blogin%5D">GPZS46R</data>
    <data name="user%5Bpassword_confirmation%5D">r0EJvXjg1</data>
    <data name="user%5Bemail%5D">DnN7MY%40xqG2SHU.UP</data>
    <data name="authenticity_token">41PlgofX%2F0EDlL1VtZQ2v4XCHtJfAcJY%3D</data>
</requete>
<requete id="5" methode="GET" url="http://localhost:3637/user/home">
        <user_agent>1_4_5</user_agent>
</requete>
```

FIGURE 5.11 – Sortie du proxy enregistreur mis en forme

Nous identifions déjà chaque requête HTTP que nous envoyons en modifiant le champ "User-Agent". En fait, il s'agit d'un champ d'une requête HTTP permettant habituellement de spécifier le client utilisé. Dans le cadre de notre travail, nous détournons son utilisation afin d'identifier les requêtes que nous émettons. Dans cette identification est aussi spécifiée l'adresse IP Source de la requête. Un proxy intermédiaire utilise cette information pour faire croire à la cible que les requêtes ont des adresses IP Source différentes. Ce proxy simulateur de plusieurs sources est représenté sur la Figure 5.2.

Après avoir exploré certains outils qui peuvent répondre dans une certaine mesure à nos besoins,

nous avons décidé d'utiliser le logiciel *Squid*[7] qui est aujourd'hui dans le monde du logiciel libre le proxy le plus utilisé et surtout le plus paramétrable.

Chaque requête arrivant dans *Squid* se voit d'abord attribuer un identifiant. Cette attribution se fait selon des règles que *Squid* appelle ACL(Access Control List). Ces règles peuvent concerner de nombreux paramètres du paquet, dont par exemple l'adresse IP-Source ou encore le User-Agent. Dans notre cas (Figure 5.12) une ACL sera validée si le User-Agent (mot clef "browser") contient l'une des adresses IP spécifiées.

```
acl machine1 browser -i 192\.168\.0\.1\$
acl machine2 browser -i 192\.168\.0\.2\$
```

FIGURE 5.12 – ACL de Squid

De ce fait *Squid* associe l'identifiant "machine1" ou "machine2" à la connexion correspondant aux critères spécifiés : "User-Agent" terminant par 192.168.0.1 ou par 192.168.0.2.
Le paquet est alors traité par *Squid*. C'est en fonction de l'identifiant de la requête que *Squid* applique différents traitements. On peut par exemple décider d'autoriser ou non la requête. Dans notre cas nous modifions l'adresse IP-Source de la requête pour ainsi simuler différentes sources aux yeux de la Cible. La figure 5.13 montre l'association de l'adresse IP de sortie en fonction de l'identifiant.

```
tcp_outgoing_address 192.168.0.1 machine1
tcp_outgoing_address 192.168.0.2 machine2
```

FIGURE 5.13 – Configuration de sortie Squid

Dans cet exemple nous n'utilisons que deux adresses IP, mais il est tout a fait possible d'en utiliser un plus grand nombre. Ceci nécessite de modifier la configuration de *Squid* et d'ajouter ces adresses IP à une interface (virtuelle ou physique) réseau de la machine. Pour effectuer ces opérations, nous avons développé un script d'initialisation qui vient lire le contenu du fichier de configuration de l'ordonnanceur afin d'effectuer les différentes opérations nécessaires.

A ce stade, la sortie de ce proxy présente la campagne prête à être envoyée à la cible avec ou sans IDS associé pour l'évaluer.

5.2.5 Application Cible et IDS

Notre plateforme d'évaluation inclut bien évidemment l'application cible et le système de détection d'intrusions destiné à la protéger [ABGRALL ET AL. 10]. Ils sont notés *Cible* et *Cible + IDS* dans la figure 5.2 (puisque la soumission de trafic à la cible peut se faire avec ou sans activation de l'IDS

7. http://www.squid-cache.org/

associé). Dans notre étude de cas, nous avons utilisé l'application et les systèmes de détection d'intrusion développés par nos différents partenaires dans le cadre du projet ANR DALI [8]. L'application web se nomme "Insecure", nous l'avons déjà évoquée au chapitre 3 et nous la décrivons brièvement dans la suite. Les systèmes de détection d'intrusion ont été conçus et implémentés par Supélec Rennes et Télécom Bretagne. Même si la façon dont fonctionnent ces IDS n'est pas notre travail de recherche et ne constitue pas le cœur de cette thèse, il nous semble néanmoins important de donner un aperçu des principes associées. C'est la raison pour laquelle nous en donnons une brève description dans les paragraphes suivants.

Insecure

Insecure est une application Web vulnérable développée dans le cadre du projet ANR DALI hors-production, c'est-à-dire application fonctionnelle, mais non-accessible au grand public. Il s'agit d'un site web de commerce en ligne développé en utilisant Ruby on Rails. Les utilisateurs peuvent effectuer des opérations atomiques telles que s'authentifier sur le site, créer un compte, naviguer sur la liste des produits, soumettre un avis, ajouter un produit à un panier d'achat, payer, etc). Ce site contient plusieurs vulnérabilités dans le Top 10 de Owasp que nous avons présentées dans la section 1.2 du chapitre 1, notamment des vulnérabilités de type injection (SQL, Xpath, OsCommanding, FileUpload), Insufficient Transport Layer Protection et Insecure Direct Object References.

IDS basé sur les invariants

Ce système de détection d'intrusion[LUDINARD ET AL. 11] nommé *Rrabids* (**R**uby on **R**ails **A**nomaly **B**ased **I**ntrusion **D**etection **S**ystem) est basé sur l'approche "boîte blanche" que nous avons évoquée dans la section 2.2.2.3 du chapitre 2. Il se base sur un apprentissage automatique du comportement "normal" du site Web. Cet apprentissage permet de définir des invariants caractéristiques du système. La violation de ces invariants permet de détecter un comportement anormal du site Web, et donc des attaques. Ces invariants sont déterminés automatiquement grâce à une technique d'instrumentation du code source de l'application [LUDINARD ET AL. 12A].
La figure 5.14 présente, pour l'application Insecure, un exemple d'invariant obtenu lors de la phase d'apprentissage. Cet invariant a èté déterminé grâce à l'instrumentation du code source de l'application, représenté dans la figure 5.15. Cet invariant indique que la valeur de la variable **session[:user].password** à la ligne **21** doit être la même que celle de la variable **params[:user][:password]** à la ligne **19**. En effet il faut que le mot de passe contenu dans la session de l'utilisateur soit identique à celui retourné par la base de données lors de l'authentification.

```
21_session[:user].password != 19_params[:user][:password]
```

FIGURE 5.14 – Example d'invariant

IDS basé sur les contrats

8. http://dali.kereval.com/index.php

```
19 if session[:user] = User.authenticate(params[:user][:login], params[:user][:password])
20       flash[:notice] = You have been successfully logged in.
21       if session[:user].admin
```

FIGURE 5.15 – Code concerné

Cet IDS, nommé *Shield* [MOUELHI ET AL. 10], dont les principes ont été décrits dans la section 2.2.2.1 du chapitre 2, fonctionne à l'aide d'une connaissance a priori des spécifications de la plateforme à surveiller. Il s'agit de définir des contrats sur toutes les interactions que peuvent avoir les utilisateurs avec la plateforme. Le *Shield* est capable de détecter tous les points d'interaction utilisateur. Dans le cadre des plateformes Web, ces points d'interaction se limitent généralement aux formulaires. Les contrats de base sont définis en fonction des spécifications présentes dans la page Web. L'administrateur peut ensuite affiner ces contrats afin de définir avec précision le comportement autorisé des utilisateurs. La figure 5.16 présente un exemple de contrat associé à un champ d'adresse électronique. En particulier, ce contrat a été saisi par l'administrateur sous forme d'expression régulière que doivent satisfaire toutes les adresses électroniques.

```
[a-zA-Z0-9._-]*@[a-zA-Z0-9._-]+[.][a-zA-Z]{2,4}
```

FIGURE 5.16 – Exemple de contrat

Le *Shield* a deux modes de fonctionnement : il peut fonctionner en mode préventif où il va bloquer toute tentative d'attaque mais il peut aussi fonctionner en mode détection et se contenter de lever des alertes.
Le *Shield* n'est en aucun cas un système dit "intrusif", il est totalement transparent pour le client et pour le serveur. Il s'agit d'un module intermédiaire, sous forme de proxy, surveillant les flux client/serveur.

5.2.6 Traitement et Analyse des résultats

L'objectif de notre travail est d'évaluer des mécanismes de détection d'intrusion. Ces évaluations consistent à émettre, sous la présence de l'un de ces mécanismes, un nombre important de requêtes sur la cible. Ensuite nous récupérons auprès du mécanisme testé les alertes levées. L'analyse et le traitement des résultats sont en quelque sorte la finalité de l'évaluation. Dans un premier temps il s'agit essentiellement de chiffrer les différents taux de vrai-positifs, faux-positifs, vrai-négatifs et faux-négatifs. Par la suite, une étude plus approfondie peut être faite afin de déterminer la nature des attaques détectées ou non. Pour ce fait, un module permettant de traiter et d'analyser les alertes récupérées auprès des différents systèmes de détection d'intrusion s'avère nécessaire. Grâce à l'enregistrement de toutes les requêtes transmises ce module calcule les faux positifs, faux négatifs, vrai positifs et vrai négatifs.

Les différents taux mentionnés ci-dessus peuvent être calculés grâce à un croisement de différents fi-

chiers. Nous maîtrisons le format de certains de ces fichiers : l'enregistrement de toutes les requêtes exécutées et la sortie de *Wasapy*. Néanmoins, nous ne maîtrisons pas la sortie du système de détection d'intrusions que nous évaluons. La première étape du traitement consiste donc à récupérer ce fichier et à le transformer dans un format prédéfini.

La transformation du fichier vers un format que nous maîtrisons nécessite la connaissance du format de ces fichiers et l'écriture d'un fichier de transformation correspondant. Dans le cadre du projet DALI, notre étude de cas, les systèmes de détection d'intrusion que nous analysons utilisent chacun leur propre format XML. Nous les transformons donc par le biais de fichiers XSLT.

Les différents modules de notre plateforme envoient différents types de requêtes vers la cible testée. Nous distinguons 5 types de requêtes :

1. *Requête saine* : émise par le crawler
2. *Requête aléatoire* : ensemble des requêtes d'injection contenant des données d'authentification générées aléatoirement et émises par *Wasapy*. Il s'agit des des requêtes R_a dans la définition de *Wasapy*.
3. *Requête erronée* : ensemble des requêtes d'injections syntaxiquement invalides émises par *Wasapy*. Il s'agit des requêtes R_{ii} dans la définition de *Wasapy*.
4. *Requête d'attaque échouée* : ensemble des requêtes d'injections syntaxiquement valides, émises par *Wasapy*, mais qui ont échoué. Ces requêtes font partie des R_{iv} dans la définition de *Wasapy*.
5. *Requête d'attaque réussie* : ensemble des requêtes d'injections syntaxiquement valides, émises par *Wasapy*, qui ont réussi à exploiter une vulnérabilité. Ces requêtes font également partie des R_{iv} dans la définition de *Wasapy*.

Soulignons que les requêtes de type 4 et 5 proviennent du même ensemble du point de vue de *Wasapy*. Ceci s'explique par le fait que *Wasapy* ne peut pas prédire si une requête va réussir ou non. Les résultats finaux de *Wasapy* permettent de différencier ces deux types de requêtes.
Nous avons choisi de définir 4 différentes situations pour évaluer les différentes métriques que nous souhaitons calculer telles qu'elles sont présentées dans le tableau 5.1. Ces situations correspondent à différents comportements souhaités des systèmes de détection d'intrusion :

- Cas N°1 : Ne lever d'alerte que lorsque une attaque a réussi
- Cas N°2 : Lever une alerte sur toute attaque réussie ou non
- Cas N°3 : Lever une alerte sur toute attaque ou requête erronée
- Cas N°4 : Lever une alerte sur toute requête provenant de *Wasapy*

Dans le premier cas, nous considérons que l'IDS ne doit lever une alerte que lorsque une attaque a effectivement réussi. Nous considérons alors que toute autre requête, qu'elle soit d'origine malveillante ou non, ne doit pas provoquer de levée d'alertes, dans la mesure où elle ne réussit pas à exploiter une vulnérabilité. Dans le second cas nous souhaitons que l'IDS lève une alerte lorsqu'il détecte une requête d'attaque, qu'elle réussisse ou non à exploiter une vulérabilité. Dans le troisième cas, nous considérons

Valeur	Résultat	N°1	N°2	N°3	N°4
Vrai-Positifs	Alerte	5	4+5	3+4+5	2+3+4+5
Faux-Positifs		1+2+3+4	1+2+3	1+2	1
Vrai-Négatifs	Absence d'alerte	1+2+3+4	1+2+3	1+2	1
Faux-Négatifs		5	4+5	3+4+5	2+3+4+5

TABLE 5.1 – Présentation des résultats selon le type des requêtes

que l'IDS doit lever des alertes lorsqu'il détecte des requêtes d'attaques (réussies ou non) mais aussi toutes les requêtes syntaxiquement invalides. Enfin, dans le quatrième cas, nous considérons que l'IDS doit lever des alertes pour tout le trafic provenant de *Wasapy*, trafic contenant à la fois les requêtes aléatoires, les requêtes syntaxiquement invalides et les requêtes syntaxiquement valides, qu'elles échouent ou qu'elles réussissent. La figure 5.17 donne une aperçu de leur présentation dans l'interface de la plateforme.

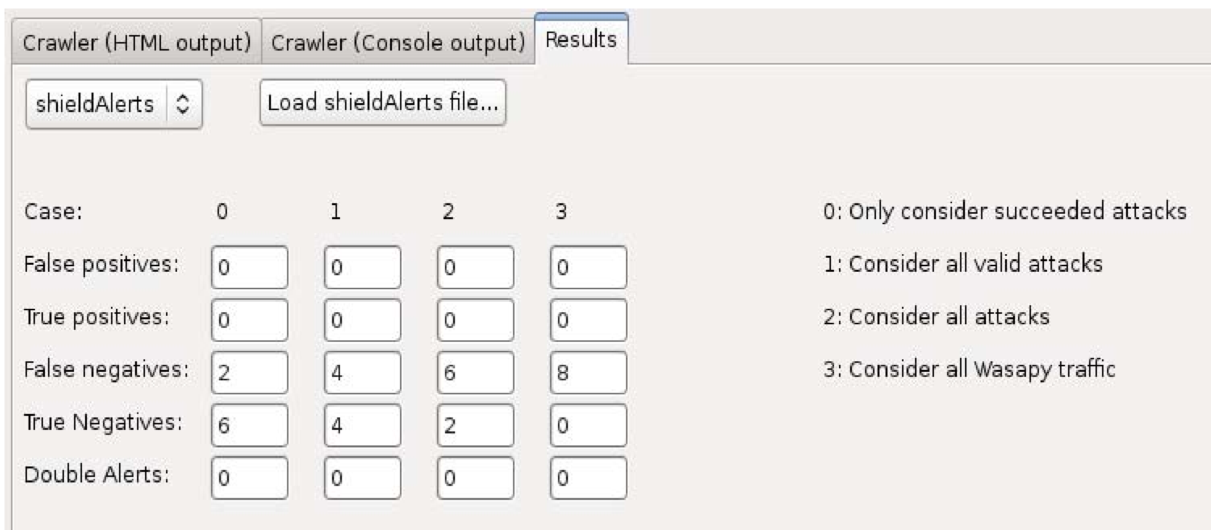

FIGURE 5.17 – Présentation des résultats

5.3 Intégration

La finalité du travail consiste au développement d'une plateforme d'évaluation complète. Dans la section 5.1 de ce chapitre, nous avons présenté l'architecture de cette plateforme ainsi que les différents

éléments qui la composent :

- Machine Cible avec IDS ou IPS
- Machine d'Attaque
- Machine de Trafic Normal
- Machine d'Analyse

Les différentes machines composant la plateforme finale ne sont pas, en réalité des machines physiques difffentes. Nous utilisons un système de machines virtuelles. Ceci a été décidé pour des raisons de coût, mais également pour la portabilité de la plateforme. Il existe différents systèmes de gestion de machines virtuelles. Nous avons décidé d'utiliser le format “VDI” qui est utilisé par VirtualBox [9]. D'autres systèmes tels que Qemu [10] peuvent également exécuter des machines au format VDI.

Nous avons également développé une interface graphique, présentée dans la figure 5.18, permettant de commander la plateforme qui rassemble l'ensemble des modules. Cette interface permet deux modes de fonctionnement. Le premier est un mode automatique. Dans ce mode, nous spécifions uniquement l'URL de la cible et la durée de l'évaluation (critère d'arrêt). Le second, manuel, permet d'intervenir sur les différents paramètres des deux sources de trafic (attaque et sain).

Grâce à cette plateforme, nous avons effectué des expérimentations, que nous présentons dans la section suivante.

5.4 Expérimentations

Pour évaluer les IDS, nous avons considéré comme étude de cas l'application *Insecure* et les IDS associés [LUDINARD ET AL. 12A], ainsi que nous l'avons présenté dans la section 5.2.5.
Nous avons réalisé deux ensembles d'expériences. Dans le premier ensemble, nous n'avons considéré que le trafic sain pour évaluer le taux de faux positifs dans des conditions normales d'utilisation. Ce trafic normal a été généré automatiquement en utilisant le crawler discuté dans la section précédente.
Dans le second ensemble d'expériences, nous avons utilisé *Wasapy* pour identifier les vulnérabilités de l'application sous test et générer automatiquement des attaques qui ont réussi à exploiter les vulnérabilités identifiées. Ces attaques sont complétées par des attaques spécialement conçues, qui sont exécutées manuellement. Au cours de cette deuxième série d'expériences, non seulement les faux négatifs (absence de détection) mais aussi les faux positifs ont été identifiés en présence de trafic malveillant.

Dans ce qui suit, nous détaillons les deux séries d'expériences ainsi que les résultats obtenus à partir de ces expériences effectuées sur les deux IDS [AKROUT ET AL. 12B].

Les expériences ont été réalisées sur une machine équippée d'un processeur Pentium 4, 3.6 GHz avec 6 Go de RAM. Le système d'exploitation est Gnu/Linux (version 2.6.32 du noyau Linux). Le serveur exécute Ruby (version 1.8) Rails (version 2.3.2) et utilise la bibliothèque bmsql-ruby 1.8.

9. `http://www.virtualbox.org/`
10. `http://www.qemu.org/`

FIGURE 5.18 – Interface de la plateforme

5.4.1 Expérimentation avec trafic sain uniquement

Pour cette série d'expérimentation, nous avons paramétré le crawler comme suit : il s'est exécuté durant 4 heures, émulant plusieurs utilisateurs Web. Sur la page d'authentification le crawler a utilisé dans 80% des cas un utilisateur normal, et dans les 20% des cas les identifiants de l'administrateur. Chaque requête a été taguée par un identifiant unique, qui est reportée au niveau des alertes qui sont levées, et qui nous permet donc de connaître la requête qui est à l'origine de chaque alerte.

5.4.1.1 IDS basé sur les invariants

Le tableau 5.2 récapitule les résultats de cette expérience effectuée en associant l'IDS basé sur les invariants à l'application cible.

L'expérience a levée 20 alertes qui correspondent à la violation de 4 invariants différents. Ils correspondent à des scénarios de trafic normal qui n'ont pas été exécutée lors de la phase d'apprentissage. L'un de ces invariants considérait que le mot de passe fourni lors de la vérification est toujours identique à celui fourni initialement, ce qui n'est pas vrai en cas d'erreur.

Nombre de requêtes	Nombre d'alertes levées	Nombre d'invariants violés
1623	20	4

TABLE 5.2 – Trafic normal uniquement avec l'IDS basé sur les invariants

5.4.1.2 IDS basé sur les contrats

Une deuxième série d'expérimentation a été réalisée en considérant le *Shield* (l'IDS basé sur les contrats). Nous distinguons deux types de contrat tels qu'ils sont décrits dans la section 5.2.5 :

- Contrat initial : défini à partir de la structure HTML de l'application, exemple pour certains champ du formulaire les valeurs saisies doivent répondre aux contraintes suivantes : Maxlength = x, Minlength = y, Single Value, Multiple Value
- Contrat ajouté manuellement pour l'application cible *Insecure* défini à travers les expressions régulières, exemple :
  - Login, password, name, addresse, etc :

    `[a-zA-Z0-9._\-&\/\\]*`
  - Email :

    `[a-zA-Z0-9._-]*@[a-zA-Z0-9._-]+[.][a-zA-Z]{2,4}`
  - card_security_code, card_number, etc :

    `[0-9]+`
  - Card_expiration :

    `[0-9]{4}`

Nous résumons dans le tableau 5.3 les résultats de cette série d'expérimentations.

Nombre de requêtes	Nombre d'alertes levées : Faux Positifs par type de contrat violé		
	Expression régulière	Valeur multiple	Longueur maximale
1614	33 : card_expiration, Password_confirmation	44	15

TABLE 5.3 – Trafic normal uniquement avec l'IDS basé sur les contrats

Nous constatons qu'une requête peut lever jusqu'à 4 alertes, autrement dit une même requête peut violer tous les contrats mis en places.

Cas	Trafic considéré comme attaque	Alertes levées	Faux positifs	Faux Négatifs
1 : Requêtes d'attaques réussies	11	11	0	0
2 : Requêtes d'attaques réussies ou pas	4440	11	0	4429
3 : Requêtes d'attaques ou erronées	8880	11	0	8869
4 : Requêtes de Wasapy	13320	11	0	13309

TABLE 5.4 – Résultats du trafic malveillant pour l'IDS basé sur les invariants

5.4.2 Expérimentation avec trafic malveillant

Pour cette série d'expérimentation, le trafic d'attaque a été généré par *Wasapy*. D'autres attaques ont été exécutées manuellement.

5.4.2.1 Trafic généré par Wasapy

L'algorithme de clustering de *Wasapy* a été paramétré pour que les trois classes de requêtes générées pour chaque point d'injection contiennent 30 requêtes chacune. Ainsi *Wasapy* envoie 90 requêtes par point d'injection et par type de vulnérabilité. Notre outil teste les vulnérabilités de type injection SQL, injection XPATH, FileInclude et OsCommanding. Ceci signifie que *Wasapy* envoie 360 (4x90) requêtes par point d'injection. Grâce à son algorithme de clustering, il est capable de déterminer les attaques qui ont réussi à exploiter les vulnérabilités identifiées. Ceci nous permet d'analyser la quantité de faux positifs. Nous traitons dans notre expérimentation les quatre cas que nous avons considérés dans la section 5.2.6, qui distinguent les situations correspondantes aux différents comportements souhaités des systèmes de détection d'intrusion.
Le tableau 5.4 récapitule les résultats de cette expérience pour l'IDS basé sur les invariants. *Wasapy* a détecté 37 points d'injections et, par conséquent, a envoyé 13320 (37x360) requêtes.

Les résultats de ces expériences montrent que le système de détection d'intrusion basé sur les invariants détecte toutes les requêtes qui ont réussi à exploiter les vulnérabilités. En effet, parmi les 90x37 requêtes envoyées par *Wasapy* pour détecter les vulnérabilités SQL, seulement 11 ont réussi à exploiter la vulnérabilité SQL. Les alertes levées correspondent exactement à ces 11 cas. Il n'y a pas de faux positifs, ni de faux négatifs. Les autres requêtes malveillantes ciblant chacun des points d'injection, et qui correspondent aux attaques XPATH, OsCommanding et le FileUpload, n'ont pas réussi parce ces classes de vulnérabilités n'existent pas dans l'application *Insecure*. Nous constatons un fait intéressant : l'IDS n'a pas levé l'alerte dans ce dernier cas.

Le tableau 5.5 récapitule les résultats de cette expérience pour l'IDS basé sur les contrats.

Les résultats de ces expériences montrent que le système de détection d'intrusion basé sur les contrats détecte toutes les requêtes qui violent les contrats définis dans la section 5.4.1.2. Nous remarquons le taux élevé de faux positifs, ceci est dû principalement au caractère " ' " que nous n'avons pas

Cas	Trafic considéré comme attaque	Alertes levées	Faux positifs	Faux Négatifs
1 : Requêtes d'attaques réussies	11	8880	8869	0
2 : Requêtes d'attaques réussies ou pas	4440	8880	4440	0
3 : Requêtes d'attaques ou erronées	8880	8880	0	0
4 : Requêtes de Wasapy	13320	8880	0	4440

TABLE 5.5 – Résultat du Trafic malveillant pour l'IDS basé sur les contrats

exclu dans nos contrats et il est émis pratiquement dans toutes les requêtes d'attaques de *Wasapy*.

5.4.2.2 Attaques manuelles

Trois scénarios d'attaque supplémentaires ont été exécutés manuellement pour exploiter d'autres vulnérabilités :

- La modification manuelle du cookie de session (permettant à un utilisateur de se connecter à l'application sans fournir aucun (login / mot de passe) valide, en utilisant le cookie de session d'un utilisateur déjà authentifié). Cette modification est possible à l'aide du plugin "Tamper Data" du navigateur Firefox qui permet d'intercepter la requête lors de son envoi et d'altérer ses données.
- La modification manuelle de paramètres de la requête qui permet de modifier le prix d'un produit au cours d'une session d'achat. Cette modification est également faisable à l'aide du plugin "Tamper Data".
- Le chargement d'un fichier Javascript malveillant au lieu d'un fichier image.

Résultats de détection par l'IDS basé sur les invariants

Pour l'IDS basé sur les invariants, la modification manuelle du prix d'un produit au cours d'une session d'achat est correctement détectée comme elle correspond à la violation d'un invariant identifié au cours de la phase d'apprentissage. Cependant, les autres attaques ne sont pas détectées.
En effet, le chargement d'un fichier JavaScript ne peut pas être détecté avec l'IDS proposé parce que la cible de l'attaque n'est pas l'application web elle-même, mais le navigateur (il s'agit d'une forme d'attaque XSS). En conséquence, il semble difficile d'être associé à un invariant. Une observation similaire peut être constatée sur les attaques basées sur la modification des cookies de session. Il semble assez difficile d'identifier un invariant qui pourrait être pertinent pour ce genre d'attaque.

Résultats de détection par l'IDS basé sur les contrats

L'IDS basé sur les contrats ne détecte que les injections de code dans les champs de formulaires dont la syntaxe viole les contrats définis. Par conséquent, nous notons la non détection des attaques

manuelles décrites ci dessus.

5.4.3 Discussion

Les expériences décrites dans 5.4.2 qui concernent l'IDS basé sur les invariants montrent que le système de détection d'intrusion est particulièrement efficace pour détecter les attaques réussies sur les applications Web, à savoir les requêtes qui mènent à la violation l'intégrité de l'état de l'application. Nous remarquons qu'il y avait ni de faux positifs ni de faux négatifs qui étaient générés lorsque le trafic malveillant a été envoyé à l'application. Nous soulignons que notre algorithme de détection de vulnérabilités, en envoyant un grand nombre de requêtes par point d'injection, est particulièrement adapté pour évaluer le nombre de faux positifs. Enfin, nos expériences avec le trafic normal illustrent l'importance de la phase d'apprentissage pour le système de détection d'intrusion. Ces expériences soulèvent quelques faux positifs, qui peuvent être associés à quatre invariants invalides. Ces invariants invalides étaient dus à la phase d'apprentissage, qui était incomplète.

En ce qui concerne l'IDS basé sur les contrats nous remarquons que pour les résultats de trafic malveillant il y a eu détection de toute injection du code dont la syntaxe viole les contrats définis ce qui prouve les limites de l'analyse syntaxique et un taux de faux positifs élevé.

5.5 Conclusion

L'objectif de notre étude est d'évaluer des systèmes de détection d'intrusion pour des applications Web. Pour cela nous avons proposé une méthodologie pour l'évaluation et le test des IDS. Pour mettre en œuvre cette méthodologie, nous avons d'éveloppé une plateforme constituée de différents modules facilement composables selon les besoins et qui intègrent les principales fonctionnalités nécessaires pour l'évaluation d'IDS. Outre la génération de trafic d'attaque avec l'outil Wasapy, la plateforme intègre des modules permettant la génération de trafic sain, le traitement des alertes pour le calcul des mesures caractérisant l'efficacité des IDS ainsi que d'autres outils par exemple pour le rejeu de trace afin de pouvoir mener des campagnes d'expérimentation ciblant différents IDS en considérant les mêmes conditions. Nous avons également développé une interface graphique qui intègre l'ensemble des modules de la plateforme pour gérer le processus d'évaluation et afficher les résultats.

Afin d'illustrer l'efficacité de notre plateforme, nous avons réalisé un ensemble d'expérimentations en prenant comme cas d'études les IDS développés dans le cadre du projet DALI. L'objectif principal de ces expériences est de caractériser la capacité des IDS à faire face à des attaques potentielles, et plus particulièrement d'évaluer leur efficacité à détecter ces attaques. Une telle efficacité est généralement caractérisée par l'évaluation du taux de vrais positifs, le taux de faux négatifs, et par le taux de faux positifs. Les expérimentations ont permis de confirmer que les deux types d'IDS développés dans le cadre du projet sont efficaces pour détecter les attaques pour lesquelles ils étaient conçus. Il s'agit des attaques susceptibles d'altérer l'état interne de l'application dans le cas de l'IDS basé sur des invariants, et des attaques qui violent les contraintes spécifiées dans le cas de l'IDS basés sur des

contrats. Dans le cas de l'IDS basé sur les invariants, la phase d'apprentissage est très importante afin d'éviter de générer des faux positif. Le problème des faux positifs se pose aussi dans le cas des IDS basés sur des contrats pour lesquels l'efficacité de détection est intrinsèquement liée aux contraintes définies dans le contrat. Le challenge est de trouver le meilleur compromis permettant de garantir une efficacité de détection d'attaques suffisamment élevée, sans trop contraindre la facilité d'utilisation de l'application.

Conclusion Générale

De nos jours, les applications Web sont à la fois beaucoup plus répandues et beaucoup plus complexes que celles des années 2000. Le développement du Web dynamique et la richesse fonctionnelle qu'offrent les nouvelles technologies du Web permettent de répondre à un grand nombre de besoins. Néanmoins, cette richesse fonctionnelle s'accompagne d'une complexité grandissante, et cette progression va de pair avec une multiplication de nombre de vulnérabilités informatiques publiées, offrant une surface d'attaque conséquente. De ce fait, aucun serveur Web n'est sûr à 100% et les applications Web sont devenues de plus en plus vulnérables et exposées à des attaques malveillantes pouvant porter atteinte à des propriétés essentielles telles que la confidentialité, l'intégrité ou la disponibilité des informations. Pour faire face à ces menaces, il est primordial de développer des techniques efficaces permettant d'aider les développeurs et les administrateurs des applications Web à identifier les vulnérabilités résiduelles dans ces applications et aussi à évaluer l'efficacité des outils de détection d'intrusions qui sont conçus pour faire face à des attaques visant à exploiter ces vulnérabilités. Les travaux effectués dans le cadre de cette thèse visent à atteindre cet objectif.

Les principales contributions de cette thèse peuvent se résumer comme suit :

- le développement d'une méthodologie originale permettant d'identifier automatiquement les vulnérabilités résiduelles d'une application Web à partir de l'analyse selon une approche boîte noire de l'application cible. L'approche proposée permet d'identifier et d'exploiter automatiquement des vulnérabilités élémentaires. Elle permet aussi de mettre en évidence différents scénarios d'attaque potentiels incluant l'exploitation de plusieurs vulnérabilités successives qui ne sont pas nécessairement indépendantes. L'identification de ces scénarios est basée sur un parcours dynamique de l'application cible qui se traduit par l'élaboration d'un graphe de navigation qui représente de façon explicite les dépendances existantes entre les vulnérabilités identifiées du site et par la suite les différents scénarios d'attaques.
- le développement d'une plateforme d'évaluation mettant en œuvre cette approche et permettant d'une part d'évaluer le comportement de l'application en présence de ces attaques et d'autre part d'évaluer l'efficacité des mécanismes de protection des applications cibles.

– la validation de la méthode proposée et de la plateforme d'évaluation en considérant différentes applications vulnérables et des techniques de détection d'intrusions développées en particulier dans le cadre du projet DALI.

Les résultats que nous avons obtenus sont encourageants. Ils montrent en particulier que l'approche proposée peut effectivement contribuer à l'amélioration de l'efficacité des scanners web existants pour les classes de vulnérabilités que nous avons considérées, en offrant la possibilité d'automatiser davantage les campagnes d'évaluation.

Nous pouvons envisager différentes perspectives à ces travaux de recherche, qui concernent à la fois notre approche de détection de vulnérabilités et notre plateforme d'évaluation.

Tout d'abord, en ce qui concerne l'approche proposée pour la détection de vulnérabilités et la génération de scénarios d'attaques basée sur l'élaboration du graphe de navigation d'un site Web, des optimisations peuvent s'avérer nécessaires afin de maîtriser la taille de ce graphe, en particulier quand il s'agit d'appliquer cette approche à des sites Web complexes. Nous avons mentionné dans le Chapitre 4 que le graphe de navigation pourrait être encore réduit en "factorisant" certains nœuds du graphe, notamment les nœuds dont nous savons pertinemment qu'ils ne peuvent pas donner lieu à une exploitation de vulnérabilités. Une réflexion plus approfondie est nécessaire pour établir les critères d'équivalence qui peuvent être considérés pour regrouper des requêtes similaires ou agréger des nœuds du graphe. Il nous reste également à mener des campagnes de tests plus significatives sur différents sites Web de complexité différente, de façon à tester en conditions réelles la montée en charge de notre scanner de vulnérabilité. Nous avons évalué théoriquement la complexité des algorithmes mis en œuvre dans *Wasapy*, en considérant des cas extrêmes qui ne sont pas nécessairement représentatifs de la plupart des sites qui existent sur Internet. Nous pensons que la complexité de ces sites est généralement plus faible que celle correspondant au pire cas. L'objectif est de valider expérimentalement cette hypothèse.

Une autre perspective intéressante consiste à enrichir les grammaires de génération des requêtes de façon à être capable de générer la plus grande variété possible d'attaques pour les injections déjà étudiées. Nous avons commencé à travailler sur ce point, notamment en ce qui concerne les injections SQL. Une piste possible consiste à définir le format général d'une injection SQL et déterminer de façon exhaustive toutes les possibilités d'injection directement à partir de la grammaire SQL elle-même. Cette idée est actuellement en cours d'investigation. Dans la même perspective, il est intéressant d'étudier plus précisément les autres types de vulnérabilités pour lesquelles notre scanner est également adapté (XPATH, OS Commanding, File Include). Nous en avons abordé le principe et proposé quelques grammaires élémentaires dans ce manuscrit mais nous n'avons pas fait d'expérimentation poussée sur ces autres classes de vulnérabilités. Pour cela il est nécessaire d'avoir à disposition des applications vulnérables contenant ce type de vulnérabilité ou bien de développer des applications dédiées à ce type d'expérimentations. Une autre possibilité serait d'utiliser des techniques d'injection de vulnérabilités que l'ont peut ensuite appliquer à des applications disponibles en source libre.

En ce qui concerne la plateforme d'évaluation proposée, les tests qui ont pu être réalisés à ce jour l'ont été uniquement sur les applications et systèmes de détection d'intrusions développés dans le cadre

du projet DALI. Il nous semble important de pouvoir tester cette plateforme dans d'autres conditions, en particulier d'autres cibles et d'autres systèmes de détections d'intrusions. Cela nécessite un effort, notamment car cela requiert la traduction des alertes émises par les systèmes de détection d'intrusions dans un format qui nous permet ensuite de les analyser. Cependant, cela ne doit pas consister un frein à l'élaboration d'autres expérimentations, qui nous permettront de bien valider notamment l'automatisation complète de notre plateforme.

Enfin, et cette perspective est à plus long terme, il nous semble important d'étudier les vulnérabilités pour lesquelles notre scanner n'est aujourd'hui pas adapté, comme les vulnérabilités de type XSS par exemple. Ce type de vulnérabilités est particulier puisqu'il vise non pas l'état du serveur Web, mais le navigateur du client lui-même (au travers d'une vulnérabilité de l'application Web toutefois). Le diagnostic de la réussite ou non de l'exploitation de la vulnérabilité doit se baser sur l'interprétation par un navigateur de la réponse renvoyée par le serveur. En cela, la technique de clustering que nous avons proposée dans le cadre de notre approche pour déduire si l'attaque a fonctionné côté serveur n'est pas adaptée. Il est donc nécessaire d'explorer d'autres solutions nous permettant de tester efficacement ces vulnérabilités sur des applications afin de pouvoir à leur tour les inclure dans notre plateforme d'évaluation. Une piste envisagée pourrait être l'intégration d'un navigateur dans cette plateforme.

Bibliographie

[ABGRALL ET AL. 10] E.Abgrall, R.Akrout, E.Alata, A.Dessiatnikoff, Y.Deswarte, S.Gombault, M.Kaâniche, K.Kanoun, I.Kemgoum, L.LE Henaff, Y.LE Traon, R.Ludinard, T.Mouelhi, V.Nicomette, A.Ribault, F.Sorin, E.Totel, F.Tronel, H.Waeselynck, "Etat de l'art et perspectives". Projet DALI, ANR. Projet DALI, 17p. Rapport LAAS N : 10822, , Août 2010.

[AKROUT ET DESSIATNIKOFF 10] R.Akrout, A.Dessiatnikoff, "An attack-goal driven approach for web applications security assessment", *European Dependable Computing Conference* (EDCC 2010), pp.47-48, Valence, Espagne, 28-30 Avril 2010.

[AKROUT ET AL. 11] R.Akrout, M.Kaâniche, V.Nicomette, "Identification de vulnérabilités et évaluation de systèmes de détection d'intrusion pour les applications web", *Congrès des Doctorants EDSYS 2011*,6p, Toulouse, France, 10-11 Mai 2011.

[AKROUT ET AL. 12A] R.Akrout, E.Alata, A.Dessiatnikoff, M.Kaâniche, V.Nicomette, "Evaluation d'IDS : Méthodologie. Projet DALI. D2.3",ANR. Projet DALI, 9p, Rapport LAAS N : 12268, Mars 2012.

[AKROUT ET AL. 12B] R.Akrout, E.Alata, Y.BACHY, M.Kaâniche, V.Nicomette, "Expérimentations et Résultats. Projet DALI. D3.2" ANR. Projet DALI, 23p, Rapport LAAS N : 12269, Mai 2012.

[ALATA ET AL. 12] E. Alata, M. Kaâniche, V. Nicomette, R. Akrout, "A Vulnerability-Based Approach to Build Attack Scenarios for Web Applications", 14 pages *The 6th International Conference on Network and Systems Security* (NSS).

[ALMGREN ET LINDQVIST 01] M.Almgren, Ulf Lindqvist. "Application-integrated data collection for security monitoring". *Proceedings of the fourth International Symposium on Recent Advances in Intrusion Detection* (RAID 2001), pp. 22-36, Davis, CA, October 2001.

[ANDERSON 80] J.P. Anderson,"Computer Security threat Monitoring and surveillance", Techni-

cal Report. PA : James P Anderson Company, Fort Washington, Pennsylvania, USA, April 1980.

[ARLAT ET AL. 93] J.Arlat, A.Costes, Y.Crouzet, J.-C.Laprie, D.Powell, ”Fault injection and dependability evaluation of faulttolerant systems”, *IEEE Trans. on Computers*,42(8), pp. 913-923, August, 1993.

[ATHANASIADES ET AL. 03] N.Athanasiades, A.Abler, J.Levine, H.Owen and G.Riley, “Intrusion detection testing and benchmarking Methodologies”, *In Proceedings of First IEEE International Workshop on Information Assurance*, pp. 63-72, 2003.

[AXELSSON 00] S.Axelsson, "Intrusion Detection Systems : A Survey and Taxonomy", Technical Report 99-15, Department of Computer Engineering, Chalmers University, Goteborg, Sweden, 2000.

[BARNETT ET B.RECTANUS 09] R.Barnett et B.Rectanus “WAF Virtual Patching Workshop : Securing WebGoat with ModSecurity”, Breach Security, 2009.

[BAU ET AL. 10] J. Bau, E. Bursztein, D. Gupta, and J. Mitchell, “State of the art : Automate black-box web application vulnerability testing”, *Proc. 2010 IEEE Symposium on Security and Privacy*, Oakland, USA, 2010.

[BEUGNARD ET AL. 99] A.Beugnard, J.-M.Jézéquel, N.Plouzeau, D.Watkins. “Making components contrat aware”, *IEEE Computer, 23(7)*, Los Alamitos, CA, USA, July 1999.

[BRADLEY 97] A.Bradley, “The use of the area under the roc curve in evaluation of machine learning algorithms”, Pattern Recognition, 12, pp.1145-1159, 1997.

[CASWELL ET AL. 03] B.Caswell, J.Beale, J-C. Foster, and Jeremy Faircloth, “Snort 2.0 Intrusion Detection” http ://2020ok.com/books/30/snort-2-0-intrusion-detection-41230.htm#ixzz1e4E1St9c.

[CHEN ET AL. 05] S. Chen, J. Xu, E. Sezer, P. Gauriar, R. Iyer. “Non control data attacks are realistic threats”, *Usenix Security Symposium*, pp.177-192, 2005.

[CHESWIK ET BELLOVIN 94] W. R. Cheswik, S. M. Bellovin, “Firewalls and Internet Security”, Addison-Wesley, 1994.

[COVA ET AL 07] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. “Swaddler : An Approach for the Anomaly-based Detection of State Violations in Web Applications”, *In Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, pp. 63-86, Gold Coast, Australia, September 2007.

[CHRISTMANSSON ET CHILLAREGE 96] J.Christmansson, R.Chillarege, “Generation of an Error Set that Emulates Software Faults”, *IEEE Fault Tolerant Computing Symp*, FCTS-26, 1996.

[CROUZET ET AL. 98] Y.Crouzet, P.Thevenod-Fosse, H.Waeselynck, “ Validation du test du logiciel par injection de fautes : l’outil SESAME”, *11ème Colloque National de Fiabilité et Maintenabilité*, pp.551-559, Arcachon, France, 1998.

[DACIER 94] M.Dacier, “ Vers une évaluation quantitative de la sécurité informatique”, Thèse de Doctorat de l’Institut National Polytechnique de Toulouse, 1994.

[DEBAR ET AL. 00] H.Debar, M.Dacier, A.Wespi, “A revised taxonomy for intrusion detection systems” , *Annales des Télecommunications*, vol. 55, pp. 361-378, 2000.

[DEBAR ET MORIN 02] H.Debar, B.Morin, “Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems”, *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*,(RAID), 2002.

[DEBAR ET TOMBINI 05] H.Debar, E.Tombini. “Webanalyzer : Accurate and fast detection of http attack traces in web server logs”, *In Proceedings of EICAR 2005*, Malta, 2005.

[DENNING 87] D.Denning, “An intrusion detection model”, *IEEE Transactions on Software Engineering*, vol. 13, pp. 222-232, 1987

[DESASILVA ET AL. 07] L.DeSaSilva, A.C.F.dosSantos, T.D.Mancilha, J.D.daSilvaSimoes, and A.Montes, “Detecting attack signatures in the real network withannida”, ElsevierLtd, 2007.

[DESSIATNIKOFF ET AL. 10] A.Dessiatnikoff, R.Akrout, E.Alata, V.Nicomette, M.Kaâniche, “Amélioration de la détection de vulnérabilitÃ©s Web par classification automatique des réponses”, Computer and Electronics Security Applications Rendez-vous (Cesar 2010), pp.116-130, Rennes, France, 22-24 Novembre 2010.

[DESSIATNIKOFF ET AL. 11A] A.Dessiatnikoff, R.Akrout, E.Alata, M.Kaâniche, V.Nicomette, “ HTML pages clustering algorithm for web security scanners”, Rapport LAAS N : 11053, Février 2011, 12p.

[DESSIATNIKOFF ET AL. 11B] A.Dessiatnikoff, R.Akrout, E.Alata, M.Kaâniche, V.Nicomette, “A clustering approach for web vulnerabilities detection”, *IEEE Pacific Rim International Symposium on Dependable Computing* (PRDC 2011), Pasadena, USA, 12-14 Décembre 2011, 10p.

[DESWARTE ET AL. 91] Y.Deswarte, L. Blain et J.C. Fabre : “Intrusion Tolerance in Distributed Computing Systems”. *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, pp. 110-121, Oakland, CA, USA, 1991.

[DESWARTE 03A] Y.Deswarte, “Chapitre 1 : La sécurité des systèmes d’information et de communication “, in Sécurité des réseaux et des systèmes répartis, dir. Yves Deswarte, Ludovic Mé, Traité IC2, Hermès, ISBN 2-7462-0770-2, pp 15-65, octobre 2003.

[DESWARTE 03B] Y. Deswarte : Comment peut-on tolérer les intrusions sur internet ? : Les systèmes critiques face aux malveillances. La Revue de l’Electricité et de l’Electronique, vol. 8, pp. 83-90, 2003.

[DESWARTE ET POWELL 06] Y. Deswarte et D. Powell, “Internet security : An intrusion-tolerance approach”, *Proceedings of the IEEE*, vol.94, num.2, pp.432-441, 2006.

[DOUPE ET AL. 10] A. Doupé, M. Cova, and G. Vigna, “Why Johnny can’t pentest : An analysis of black-box web vulnerability scanners”, *Proc. DIMVA*, 2010.

[DUPONT 96] P.Dupont, “Incremental regular inference”, pp 222-237, 1996.

[DURAES ET MADEIRA 06] J.Duraes, H.Madeira, ”Emulation of Software Faults : A Field Data Study and a Practical Approach”, *IEEE Trans. on Software Engineering*, Vol.32, No.11, November 2006.

[ESPOSITO ET AL. 05] M.Esposito, C. Mazzariello, F. Oliviero, S. Romano, et C. Sansone. “Evaluating pattern recognition techniques in intrusion detection systems”, *In Proceedings of the 5th International Workshop on Pattern Recognition in Information Systems (PRIS)*, May 2005, pp. 144-153.

[EVERITT 80] B.S.Everitt, Cluster Analysis, Second Edition, London : Heineman Educational Books Ltd. , 1980.

[FAWCETT 06] T.Fawcett, “An introduction to roc analysis. Pattern recognition Letters”, 27 pp. 861-874, 2006.

[FIELDING ET AL. 99] R.Fielding, J.Gettys, J.Mogul, H.Frystyk, L.Masinter, P.Leach, T.Berners-Lee, “Hypertext transfer protocol - HTTP/1.1”, RFC 2616, June 1999.

[FOGLA ET LEE 06] P.Fogla, and W. Lee, “Evading network anomaly detection systems : formal reasoning and practical techniques”, *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 59-68, 2006.

[FONSECA ET AL. 07A] J.Fonseca, M.Vieira et H.Madeira,”Detecting Malicious SQL”, *In Proceedings of TrustBus*, pp 259-268, 2007.

[FONSECA ET AL. 07B] J. fonseca, M. Vieira, and H. Madeira, “Testing and Comparing Web vulnerability scanning tools for SQL injections and XSS attacks”, *Proc.2007 IEEE Symposium Pacific Rim Dependable Computing (PRDC 2007)*, Victoria, Australia, pp. 330-337, USA, 2007.

[FONSECA ET VIEIRA 08A] J.Fonseca, M.Vieira, “Mapping Software Faults with Web Security Vulnerabilities”, *IEEE/IFIP Int. Conference on Dependable Systems and Networks*, June 2008.

[FONSECA ET AL. 08B] J.Fonseca, M.Vieira, H.Madeira, “Training Security Assurance Teams using Vulnerability Injection”, *IEEE Pacific Rim Dependable Computing conference*, December 2008.

[FONSECA ET AL. 09] J.Fonseca, M.Vieira, H.Madeira, “Vulnerability and attack injection for web applications”, *The 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 93-102, Lisbon, Portugal, 2009.

[FONSECA 10] J.Fonseca, “Evaluating the [In]security of Web applications”, PhD Thesis, University of Coimbra, 2010.

[FORREST ET AL. 96] S.Forrest, S.-A. Hofmeyr, A.Somayaji, T.-A. Longstaff, “A Sense of Self for Unix Processes”, *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy, IEEE Computer Society Press*, pp. 120-128, May 1996.

[GADELRAB 08] M.S.Gadelrab, "Évaluation des Systèmes de Détection d'Intrusion", Thèse de l'Université Toulouse III-Paul Sabatier, Toulouse, France, 2008.

[GAO ET AL. 04] D.Gao, M.-K. Reiter, D.Song, "Gray-box extraction of execution graphs for anomaly detection". *In Proceedings of the 11th ACM conference on Computer and communications security*, pp.318-329, 2004.

[GHOSH ET AL. 00] A.-K. Ghosh, Ch.Michael, M.Schatz. "A realtime intrusion detection system based on learning program behavior", *In Proceedings of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, pp. 93-109, October 2000.

[GIACINTO ET AL. 06] G.Giacinto, R. Perdisci, M. D. Ri, F. Roli, "Intrusion detection in computer networks by a modular ensemble of one-class classifier", 2006.

[GILHAM ET AL. 92] F. Gilham R. Jagannathan P.G. Neumann H.S. Javitz A. Valdes T.D. Garvey T.F. Lunt, A. Tamaru. "A real-time intrusion detection expert system (ides)". 1992.

[GRAHAM 72] R.-L. Graham,"An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set", *Information Processing Letter*, Vol. 1, Num. 4, pp. 132-133, USA, January 1972.

[GUPTA ET AL. 08] P.Gupta, Ch.Raissi, G.Dray, P.Poncelet, J.Brissaud,"Détection d'intrusions : de l'utilisation de signatures statistiques", *Actes du 5ieme Atelier Fouille de Données Complexes (FDC 08)*, pp. 105-116, , Sophia Antipolis, France, Janvier 2008.

[GUTESMAN 09] E. Gutesman, "gFuzz : An Instrumented Web Application Fuzzing Environment", *Hack.Lu '08*, Luxembourg, 2008.

[HAINES ET AL. 01] J.H.Lee, L.M.Rossey, R.K.Cunningham, "Extending the DARPA Off-Line Intrusion Detection Evaluations", *In the Proceedings of DISCEX 2001*, Anaheim, CA, June 2001.

[HALFOND 06] W.G.J.Halfond, J.Viegas, A.Orso, "A Classification of SQL Injection Attacks and Countermeasures", *Proc. of the International Symposium on Secure Software Engineering*, 2006.

[HEBERLEIN ET AL. 90] L. Todd Heberlein, G.Dias, K.N.Levitt, B.Mukherjee, J.Wood, D.Wolber. "A network security monitor", *In Proceedings ot the 1990 IEEE Computer Society Symposiumon Research in Security and Privacy*, pp. 296-304, Oakland, CA, May 1990.

[HOFMEYR 98] S.-A. Hofmeyr, S.Forrest, A.Somayaji, "Intrusion Detection Using Sequences of System Calls", *Journal of Computer Security*, 1998.

[HUANG ET AL. 03] Huang, Yao-Wen and Huang, Shih-Kun and Lin, Tsung-Po and Tsai, Chung-Hung, "Web application security assessment by fault injection and behavior monitoring", *WWW '03 : Proceedings of the 12th international conference on World Wide Web*, pp. 148-159, 2003.

[HUANG ET AL. 04] Y.-W Huang, F. Yu, C. Huang, C.-H.Tsai, D.-T.Lee, and S.-Y Kuo, "Securing Web Application code by static analysis and runtime protection", *Proc. 13th International Conference on World Wide Web (WWW'04)*, pp. 40-52, ACM, New York, USA.

[HUNT ET MCILROY 76] J. W. Hunt and M. D. McIlroy, "An Algorithm for Differential File Comparison", Tech. Report CSTR 41, Bell Laboratories, Murray Hill, NJ, 1976.

[ILGUN 95] K.Ilgun, R.A. Kemmerer, P.A. Porras, "State Transition Analysis : A Rule-Based Intrusion Detection Approach", Software Engineering, Vol. 21, pp. 181-199, 1995.

[INGHAM ET AL. 07] Kenneth L. Ingham and Hajime Inoue. "Comparing anomaly detection techniques for http". *In Proceeding of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID'2007)*, Springer, pp. 42-62, Queensland, Australia, September 2007.

[INGOLS ET AL. 06] K.Ingols, R.Lippmann, K.Piwowarski, "Practical attack graph generation for network defense", *In : 22nd Annual Computer Security Applications Conference (ACSAC)*, Miami Beach, Florida, December 2006.

[IYER 95] R.Iyer, "Experimental Evaluation", *Special Issue FTCS-25 Silver Jubilee, IEEE Symp. on Fault Tolerant Computing*, pp. 115-132, 1995.

[JOHNSON 67] S. C. Johnson, "Hierarchical Clustering Schemes", *in Psychometrika Journal*, pp. 241-254, Vol.2, 1967.

[JOVANOVIC ET AL. 06] N. Jovanovic, C. Kruegel and E. Kirda, "Pixy : A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)", *Security and Privacy, IEEE Symposium on Los Alamitos*, CA, USA, 2006.

[JOVANOVIC ET AL. 10] N. Jovanovic, C. Kruegel, and E. Kirda, "Static analysis for detecting taint-style vulnerabilities in web applications", *Journal of Computer Security*, 18, pp. 861-907, 2010.

[KAMADA 89] T.Kamada, S.Kawai, "An algorithm for drawing general undirected graphs", *Information Processing Letter*, Vol. 31, Num.1, pp. 7-15, issn=0020-0190, Elsevier North-Holland, Inc., 1989.

[KIEZUN ET AL. 09] A.Kiezun, P. J. Guo, K.Jayaraman and M. D. Ernst, "Automatic creation of SQL Injection and cross-site scripting attacks", *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on Vancouver*, BC, 2009.

[KIRDA ET AL. 06] E.Kirda, C.Kruegel, G.Vigna and N.Jovanovic, "Noxes : a client-side solution for mitigating cross-site scripting attacks", *SAC '06 : Proceedings of the 2006 ACM symposium on Applied computing*, pp.330-337, New York, USA, 2006.

[KO ET LEVITT 94] C. Ko, G. Fink and K.N Levitt, "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring", *Proceedings of the 10th Annual*

Computer Security Applications Conference (ACSAC'94), pp 134-144, Orlando, FL, 1994.

[KO ET LEVITT 97] C. Ko, M.Ruschitzka and K.N Levitt, "Execution monitoring of security-critical programs in distributed systems : A specification-based approach", *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pp 175-187, Oakland, CA, 1997.

[KORDY ET AL. 11] B.Kordy, S.Mauw, S.Radomorovic, P.Schweitzer, "Foundations of Attack-Defense Trees", *Proc. of Formal Aspects of Security and Trust (FAST 2010)*, LNCS Vol. 6561, pp. 80-95.

[KOSUGA ET AL. 07] Y.Kosuga, K.Kono, M.Hanaoka, "Sania : Syntactic and Semantic Analysis for Automated Testing against SQL Injection", 23rd Annual Computer Security Applications Conference (ACSAC2007), pp. 10-14, Miami Beach, Florida, USA, 2007.

[KRUEGEL ET AL. 03] C.Kruegel, D.Mutz, F.Valeur, G.Vigna, "On the detection of anomalous system call arguments", In 8th European Symposium on Research in Computer Security (ESORICS 2003), pp. 326-343, Gj vik, Norway, October 2003. October 2003.

[LAM ET AL. 08] M. S. Lam, M. Martin, B. Livshits, and J. Whaley, "Securing Web Applications with static and dynamic information flow tracking", *ProcÂ . of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation (PEPM'08)*, pp. 3-12, New York, NY, USA, 2008.

[LAPRIE ET AL. 96] J.-C. Laprie, J.Arlat, J.-P. Blanquart, A.Costes, Y.Croust, Y.Deswarte, J.-C.Fabre, H.Guillermain, M.Kaâniche, C.Mazet, D.Powell, C.Rabéjac, P.Thévenod, Guide de la sûreté de fonctionnement, Cépadués Editions, 1996, 370p, ISBN 2.85428.382.1, seconde édition.

[LEITNER ET AL. 07] A.Leitner, I.Ciupa, M.Oriol, B.Meyer, A.Fiva, "Contract driven development = test driven development - writing test cases", Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Dubrovnik, Croatia, September 2007.

[LE TRAON ET AL.06] Y.Le Traon, B.Baudry, and J-M Jézéquel,"Design by Contract to improve Software Vigilance",. IEEE Transactions on Software Engineering, 32(8). August 2006.

[LEVENSHTEIN 66] V.Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals", Soviet Physics Doklady, pp. 707-710, 1966.

[LI 05] X.-B.Li, "A scalable decision tree system and its application in pattern recognition and intrusion detection". Decision Support Systems 41(1), pp. 112-130, 2005.

[LIPPMANN ET AL. 00] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba and K. Das, “Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation”, in 3rd symposium on Recent Advances in Intrusion Detection (RAID 2000), pp.162-182, Springer Verlag, 2000.

[LIVSHITS ET LAM 05] V.B. Livshits and M. S. Lam, “Finding security errors in Java program with static analysis”, *Proc. 14th Usenix Security Symposium*, Baltimore, MD, USA, 2005.

[LUDINARD ET AL. 12A] R.Ludinard, E.Totel, F.Tronel, V.Nicomette, M.Kaaniche, E.Alata, R.Akrout, Y.Bachy, “Detecting attacks against data in web applications”, Rapport LAAS NÂ°12013, Janvier 2012, 12p.

[LUDINARD ET AL. 12B] R.Ludinard, E.Totel, F.Tronel, V.Nicomette, M.Kaaniche, E.Alata, R.Akrout, Y.Bachy, “Detecting Attacks against Data in Web Applications”, *The 7th International Conference on Risks and Security of Internet and Systems* (CRISIS), 8 pages.

[MADEIRA ET AL. 00] Madeira, H. Vieira, M., Costa, D. “On the Emulation of Software Faults by Software Fault Injection.”, IEEE/IFIP Int. Conf. on Dependable System and Networks, 2000.

[MAFTIA ET AL. 03] D.Powel, R.STrout et al., “Conceptual Model and Architecture of MAFTIA”, 2003. http ://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/maftia/deliverables/full.htm#1

[MARTIN ET LAM 08] M.Martin et M.-S. Lam, “Automatic generation of XSS and SQL injection attacks with goal-directed model checking” In USENIX Security, 2008.

[MASSICOTTE ET AL. 06] F.Massicotte,F. Gagnon, Y.Labiche, L.C. Briand and Mathieu Couture, “Automatic Evaluation of Intrusion Detection Systems”, 22nd Annual Computer Security Applications Conference (ACSAC2006), pp 361-370, Miami Beach, Florida, USA, ,December 2006.

[MARTY 02] R. Marty. “Thor : A Tool to Test Intrusion Detection Systems by Variations of Attacks”. ETH Zurich Diploma Thesis, March 2002.

[MAUW ET OOSTDIJK 05] Mauw, S., Oostdijk, M., “Foundations of attack trees”, Information Security and Cryptology-ICISC 2005, LNCS Volume 3935, pp. 186-198.

[MCHUGH 00] J. McHugh, “Testing Intrusion Detection Systems : A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory”, ACM Trans. on Inf. and System Security, 3 (4), pp. 262-294, 2000.

[MEYER 92] B.Meyer, “Object-oriented software construction”, Prentice Hall, 1992.

[MINAMIDE 05] Y. Minamide, “Static approximation of dynamically generated web pages”, *Proc. 14th International Conference on World Wide Web (WWW’05)*, Chiba, Japan.

[MOUELHI ET AL. 10] T.Mouelhi et Y.Le Traon et E.Abgrall et B.Baudry et S.Gombault, “Tailored Shielding and Bypass Testing of Web Applications”, *International Conference on Software Testing, Verification and Validation (ICST’11)*, Berlin Germany, 2011.

[MUKHERJEE ET AL. 94] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. “Network intrusion detection”. IEEE Network, 8(3) :26-41, May-June 1994.

[MUTZ ET AL. 03] D.Mutz, G.Vigna , R.Kemmerer, “An experience developping an IDS stimulator for the black box testing of network intrusion detection system”, In Annual Computer Security Applications Conference, Las Vegas, NV, pp 374-383, 2003.

[MUNZ ET AL. 07] G. Munz, S. Li and G. Carle. “Traffic Anomaly Detection Using k-means Clustering”. In GI/ITG Workshop MMBnet, 2007.

[MUTZ ET AL. 07] D.Mutz, W.Robertson, G.Vigna, R.Kemmerer. “Exploiting Execution Context for the Detection of Anomalous System Calls”, Proceeding of the 10th International Symposium on Recent Advances in Intrusion Detection (RAID’2007), 2007.

[NEVES ET AL. 06] Neves, N., Antunes, J., Correia, M., VerÃssimo, P., Neves R., “Using Attack Injection to Discover New Vulnerabilities”, IEEE/IFIP International Conference on Dependable Systems and Networks, 2006.

[NEWSOME ET AL. 05] J.Newsome, B. Karp, et D. Song, “Polygraph : Automatically generating signatures for polymor- phic worms”. Proceedings of the 2005 IEEE Symposium on Security and Privacy, pp. 226-241, 2005.

[NGUYEN ET AL. 05] A. Nguyen-Tuong, S. Guarnieri, D. Greene, J. Shirley and D. Evans. *Automatically Hardening Web Applications Using Precise Tainting* . In IFIP Security 2005, 2005.

[NOEL ET AL. 10] Noel, S., Jajodia, S., and Singhal, A., “Measuring Security Risk of Networks Using Attack Graphs”, *International Journal of Next-Generation Computing*, **1 (1)**, pp. 135-147.

[NOSEEVICH ET PETUKHOV 11] G.Noseevich, A.Petukhov, “Detecting Insufficient Access Control in Web Applications”, *In SysSec Workshop*, pp. 11-18, Los Alamitos, CA, USA, 2011.

[NSS 03] The NSS Group : Intrusion Detection Systems Group Test(Edition 4) (2003).

[ORTALO ET AL. 99] R.Ortalo, Y. Deswarte, M. Kaaniche, “Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security”, *IEEE Transactions on Software Engineering*, vol. 25, pp. 633-650, 1999.

[OWASP] Open Web Application Security Project, http ://www.owasp.org

[PAXON 99] V. Paxson, "Bro : A System for Detecting Network Intruders in Real-Time", *Proceedings of the 7th USENIX Security Symposium*, San Antonio, Texas, January 1998.

[PHILLIPS ET SWILER 98] C.Phillips, L.P.Swiler, “A graph-based system for network-vulnerability analysis”, *In : NSPW 1998 : Proc. of the 1998 workshop on New security paradigms*, pp. 71-79. ACM Press, New York, 1998.

[PIETRAZEK ET BERGHE 05] T. Pietraszek and C. V. Berghe. “Defending Against Injection Attacks Through Context-Sensitive String Evaluation”. *In Recent Advances in Intrusion Detection 2005* (RAID), 2005.

[PROCTOR 01] P.Proctor, “Practical Intrusion Detection Handbook”. *Upper Saddle River, NJ : Prentice-Hall*, 2001.

[PUKETZA ET AL. 96] Puketza, Nicholas J. ; Zhang, Kui ; Chung, Mandy ; Mukherjee, Biswanath and Olsson, Ronald A., “A methodology for Testing Intrusion Detection Systems”, IEEE Transactions on Software Engineering, 22, pp. 719-729, 1996.

[ROBERTSON ET AL. 06] W.Robertson, G.Vigna, C.Kruegel, and R. A. Kemmerer, “Using generalization and characterization techniques in the anomaly-based detection of web attacks”, *In Proceedings of the Network and Distributed System Security Symposium (NDSS 2006)*, San Diego, CA, February 2006.

[ROESCH 99] M.Roesch. “Snort - lightweight intrusion detection for networks”, *In Proceedings of the USENIX LISA’99 conference*, pp. 229-238, Seattle, WA, November 1999.

[SANIEE ET AL. 07] M.Saniee, J. Habibi, Z. Barzegar, et M. Sergi, “A parallel genetic local search algorithm for intrusion detection in computer networks”. *CSICC*, 2007.

[SCHNEIER 99] B., Schneier, “Modeling security threats”, *Dr Dobb’s journal*, December 1999.

[SEKAR ET AL. 02] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang and S. Zhou, “Specification-based Anomaly Detection : A New Approach for Detecting Network Intrusions”, *Proceedings of the 9th ACM conference on Computer and communications security*, pp 265-274, Washington, DC, 2002.

[SHEYNER ET AL. 02] Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M., “Automated generation and analysis of attack graphs”. *In : Proc. of the 2002 IEEE Symposium on Security and Privacy*, pp. 254-265. 2002.

[STEFAN ET AL. 06] K.Stefan, E. Kirda, C. Kruegel and N. Jovanovic, “SecuBat : a web vulnerability scanner”, *Proceedings of the 15th international conference on World Wide Web (WWW ’06)*, Edinburgh, Scotland, 2006.

[LUDINARD ET AL. 11] R.Ludinard, E.Totel, F.Tronel, V.Nicomette, M.Kaâniche, E.Alata, R.Akrout et Y.Bachy, “RRABIDS : Ruby on Rails Anomaly Based Intrusion Detection System”, *Rapport Laas N11294*, 2011.

[TOMBINI ET AL. 04] Elvis Tombini, Hervé Debar, Ludovic Mé, and Mireille Ducassé. “A serial combination of anomaly and misuse IDSes applied to HTTP traffic”. *In Proceedings of ACSAC’2004*, pp. 428-437, Tucson, AZ, December 2004.

[VALDES ET AL. 95] A. Valdes D. Anderson, T. Frivold, “Next-generation intrusion detection expert system (nides) : A summary”, 1995.

[VIGNA ET AL. 03] G.Vigna, W.Robertson, V.Kher, and R. A. Kemmerer, “A stateful intrusion detection system for world-wide web servers”, *In Proceedings of the Annual Computer Security Applications Conference* (ACSAC 2003), pp. 34-43, Las Vegas, NV, December 2003.

[VIGNA ET AL. 04] G.Vigna, W.Robertson, D.Balzarotti, “Testing network based intrusion detection signatures using mutant exploits” *Proc. ACM conference on Computer and communications security*, pp 21-30, 2004.

[WARRENDER ET AL. 99] C.Warrender, S.Forrest, B.Pearlmutter. “Detecting intrusions using system calls : Alternative data models”. In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pp. 133-145, Oakland, CA, May 1999.

[WASSERMANN ET SU 07] G. Wassermann and Z. Su, “Sound and precise analysis of web applications for injection vulnerabilities”, *SIGPLAN Notices*, vol 42, n06, pp. 32-41,2007.

[XIE ET AIKEN 06] Y. Xie, A. Aiken, “Static detection of vulnerabilities in scripting languages”, *Proc. 15th USENIX Security Symposium*, pp. 179-192, 2006.

[YEUNG ET DING 03] D-Y.Yeung, and Y. Ding, “Host-based intrusion detection using dynamic and static behavioral models”. *Pattern Recognition 36(1)*, pp. 229-243, 2003.