Quack1

Rapport de Stage de fin d'Études
Master 2 – Sécurité de l'Information et Cryptologie

Scénarios d'Attaques et Détection d'Intrusions

— du 04 Mars au 30 Août 2013 —

« Diffusion Publique »

Conix Security
34, rue Guynemer
92130 Issy Les Moulineaux
tel: 01 41 46 08 00

Tuteur :
Julien Iguchi-Cartigny

Maître de Stage :
Adrien Vernois

Résumé

Pour conclure le Master CRYPTIS (*Sécurité de l'Information et Cryptologie*) de Limoges, un stage de six mois est réalisé au sein d'une entreprise dans le domaine de la sécurité des systèmes d'information.

J'ai effectué mon stage dans la société Conix, au sein du service Conix Security, sous la tutelle d'Adrien Vernois et Julien Iguchi-Cartigny, du 4 Mars au 30 Août 2013. Cette entreprise, localisée à Paris, travaille exclusivement dans le domaine de l'informatique, mais son activité couvre plusieurs secteurs, dont la sécurité informatique.

Dans ce mémoire, je vais présenter le contenu de mon stage qui avait pour sujet la détection d'intrusions. Je présenterai la détection d'intrusions d'un point de vue théorique, afin de donner les clés de compréhension nécéssaires, puis je détaillerai les différents scénarios d'attaques que j'ai imaginé, ainsi que la façon de les détecter. Enfin, je terminerai par exposer les limites de ces techniques et je donnerai quelques pistes permettant de les dépasser.

Abstract

To conclude the Master's degree *CRYPTIS* (*IT Security and Cryptology*) from Limoges, a six month internship is realised in a company working in IT security.

I made my internship in the society Conix, into the Conix Security service, managed by Adrien Vernois and Julien Iguchi-Cartigny. This internship started on Month, 4th and finished on August, 30th 2013. This society, based in Paris, works exclusively on IT, but its activity covered several fields, like IT security.

In this report, I will present the content of my internship which has "intrusion detection" for subject. I will present theorically the intrusion detection field, to give keys to understand this report, then I will detail the differents attack scenarios I imagined and the way I detected them. Finally, I will expose the limits of these techniques and I will give some clues to surpass them.

Je tiens à remercier l'équipe pédagogique de la Faculté des Sciences de Limoges, ainsi que les intervenants professionnels de la formation CRYPTIS pour le haut niveau de leurs enseignements.

Je remercie également Adrien Vernois*, mon maître de stage à Conix, pour avoir partagé ses connaissances et son expérience avec moi au cours de ce stage. Ses recommandantions et conseils m'ont permis de mieux avancer dans les différents travaux que j'ai effectué au cours de mon stage.*

Je tiens également à le remercier pour m'avoir donné la possibilité de travailler sur des projets variés, ce qui m'a permis de parfaire et d'approfondir mes connaissances, de découvrir de nouvelles technologies, et de travailler sur des projets intéressants.

Je tiens également à remercier les membres de l'équipe SOC, et plus généralement l'ensemble de l'équipe Conix Security pour leurs nombreux conseils, ainsi que pour leur bonne humeur et leur sympathie au quotidien.

Enfin, je tiens à remercier mes parents qui m'ont soutenu et aidé tout au long de mes études, ainsi qu'Ingrid qui, plus que toute autre personne, m'a soutenu au cours de ce stage.

TABLE DES MATIÈRES

Résumé **i**

Remerciements **ii**

Table des matières **iii**

Introduction **v**

1 Contexte de Réalisation du Stage **1**
1.1 L'entreprise 1
1.2 Objectifs du Stage 4

2 La Détection d'Intrusions **5**
2.1 La Sécurité Informatique 6
2.1.1 Sécurité Informatique : La protection des données 6
2.1.2 Sécurité Informatique : La protection contre des attaques 7
2.2 Approche théorique à la détection d'intrusions 9
2.2.1 Caractéristiques des IDS 9
2.2.2 Propriétés attendues des IDS 12
2.3 Principaux IDS 14
2.3.1 Snort 14
2.3.2 OSSEC 17
2.3.3 Audit 20
2.3.4 Limitations de ces IDS 21
2.4 Améliorer la fiabilité de la détection : la corrélation d'évènements 25
2.5 OSSIM : *Open-Source SIEM* 28

3 Mise en pratique - Scénarios d'attaques **32**
3.1 Outils d'Audits Automatisés - Détection NIDS 33
3.1.1 Analyse réseau d'outils d'audit Web 33

3.1.2 Outils analysés . . . 35
3.2 Limites de la détection d'intrusions . . . 43
3.2.1 Contournement d'IDS . . . 43
3.2.2 Attaques par « force brute » . . . 45
3.2.3 Attaques Web génériques . . . 46
3.2.4 Le cas *nmap* . . . 49
3.3 Scénarios d'attaques évolués – Corrélation N/HIDS et SIEM . . . 52
3.3.1 Détection avancée d'une attaque de type LFI . . . 53
3.3.2 Détection avancée d'attaques lancées par des outils automatiques . . . 54
3.3.3 Détection d'une attaque « générique » . . . 57
3.3.4 Surveillance avancée de l'authentification . . . 59
3.4 La corrélation d'évènements : solution miracle ? . . . 60

4 *Security Operations Center* : La détection d'intrusions en pratique 62
4.1 *Security Operations Center* . . . 63
4.2 La détection des intrusions appliquée par les SOC . . . 63
4.3 Levée de doutes . . . 64
4.4 Activités connexes . . . 67
4.4.1 Configuration générique OSSEC . . . 67
4.4.2 Amélioration d'une solution SIEM éditeur . . . 67
4.4.3 Rédaction d'un article pour MISC Mag . . . 69

Conclusion 71

Glossaire 74

Bibliographie 86

A Règles Snort détaillées 87
A.1 Outils d'audit automatisés . . . 87
A.2 Attaques Web génériques . . . 94

B Parties sur lesquelles sont basées la détection d'outils d'audit 96

C Payloads fréquemment utilisés lors d'attaques Web 98
C.1 Injections SQL . . . 98
C.2 Cross-Site Scripting . . . 99
C.3 Local File Inclusion . . . 99
C.4 Burp *Scanner* . . . 103

D Architecture en couches du modèle TCP/IP 104

E Modèle de fiche d'incident « brute » 105

F Configuration par défaut OSSEC 107

INTRODUCTION

L'informatique est une science relativement jeune. Depuis ses débuts dans les années 40, avec notamment les travaux d'Alan Turing, en passant par les premiers ordinateurs personnels dans les années 70, nous sommes aujourd'hui arrivés à l'heure de la miniaturisation et de la mobilité, où chacun possède et utilise quotidiennement plusieurs périphériques différents afin d'accéder à des ressources informatisées : ordinateurs portables, smartphones, tablettes.

Dans le monde de l'entreprise, l'informatique est également de plus en plus présente. Elle est devenue, au fil des années, nécessaire à toutes les tâches administratives, remplaçant le papier et le crayon sur tous les bureaux, et permet de recenser toutes les données vitales des sociétés dans d'immenses bases de données. Elle a également infiltré toute la chaîne de production puisque aujourd'hui on retrouve une quantité incroyable d'installations techniques entièrement pilotées par ordinateur.

Les données de ces entreprises sont indispensables au bon fonctionnement des activités des salariés, y compris lors des déplacements de ces derniers. Les machines stockant les informations se retrouvent donc connectées entre elles au sein de grands réseaux internes, et généralement connectées à Internet.

Avant l'utilisation massive des ordinateurs, l'espionnage industriel était principalement basé sur des techniques physiques, comme fouiller les poubelles à la recherche d'informations griffonnées sur des feuilles de brouillon, ou en dérobant les attachés-cases des dirigeants au détour d'un coin de rue.

Avec l'arrivée de l'informatique, des réseaux, puis d'Internet, les vols d'informations physiques se sont peu à peu transformés en intrusions informatiques. Il devient donc essentiel que les données stockées sur des serveurs soient suffisamment protégées et sécurisées. Pour cela, de nombreuses techniques de sécurisation sont mises en place, afin d'assurer le respect de bonnes pratiques de sécurités (comme celles définies dans la série des normes ISO 27000 [1]).

1. La suite ISO/CEI 27000 (ou « Normes ISO 27k ») contient les normes publiées par l'ISO (*Interna-*

Cependant, on peut faire deux observations quant à la sécurité des systèmes d'informations d'aujourd'hui :

La sécurité des SI [2] aujourd'hui ne fait pas partie des priorités des directions des systèmes d'informations, et les techniciens (développeurs, administrateurs systèmes), ni même les utilisateurs, ne sont pas suffisamment formés et sensibilisés à la sécurité informatique. Le postulat de Nicolas Ruff, « la sécurité est un échec », énoncé en 2009 au SSTIC ([Ruf09]) est, 4 ans plus tard, toujours d'actualité. Peu d'entreprises disposent d'un niveau de sécurité suffisant pour protéger convenablement leurs données.

De la même façon, on remarque qu'il est impossible pour un système d'information d'avoir une sécurité absolue et à toute épreuve. Il faut donc recourir à des techniques différentes que la sécurisation pure afin de s'assurer du niveau de fiabilité du système d'information (ou *SI*).

La première technique, l'audit de sécurité, permet de détecter les failles et les vulnérabilités en amont et d'obtenir un aperçu du niveau de sécurité du SI. La seconde technique est la détection d'intrusions. Elle permet, comme son nom l'indique, de détecter des intrusions à posteriori, à partir d'une liste de critères définissant les activités anormales et malicieuses qui pourraient survenir.

C'est dans ce contexte que se déroule mon stage de fin d'études dans la société Conix. L'objectif du stage est d'établir une liste d'attaques connues pouvant être lancées contre des systèmes d'information. Ces attaques peuvent être très simples (par exemple, l'utilisation basique d'un outil), ou plus complexes et distribuées sur plusieurs niveaux du SI.

Le second objectif du stage et de permettre à des systèmes de détection d'intrusions de détecter ces attaques.

Dans un premier temps, je présenterai Conix, l'entreprise dans laquelle j'ai effectué mon stage, ses différentes activités, ainsi que les objectifs de mon stage.

Dans un deuxième temps, je détaillerai les principes de fonctionnement des Systèmes de Détection d'Intrusions (ou *IDS* [3]), notamment les *Host-Based* IDS et *Network-Based* IDS.

Je ferai également un focus sur les principaux outils utilisés dans ce stage, en détaillant notamment les principes de base de configuration et d'écriture de règles de détection, nécessaires à la bonne compréhension de la suite du rapport.

tional Organization for Standardization) et l'IEC (*International Electrotechnical Commission*) au sujet de la sécurité informatique. Ces normes contiennent notamment les bonnes pratiques en matière de management de la sécurité, ainsi que pour la définition et la mise en place d'un système de gestion de la sécurité de l'information.

2. Un SI, ou Système d'Information désigne la totalité des ressources liées à la gestion de l'information (généralement informatisée) en entreprise.

3. Un IDS, ou Système de Détection d'Intrusions, est un logiciel permettant de détecter, au moyen de règles définies par l'utilisateur, des activités anormales ou suspectes sur un système cible, permettant ainsi de détecter des intrusions, réussies ou non.

Ensuite, je présenterai les différents types d'attaques que j'ai analysé afin de pouvoir les détecter de façon efficace.

Afin de me familiariser avec la manipulation des outils de détection utilisés dans l'entreprise, j'ai consacré le début de mon stage à la détection d'outils d'audits automatiques utilisés dans de nombreuses attaques.

Dans un second temps, j'ai travaillé sur des scénarios plus évolués et complexes, mettant en oeuvre plusieurs phases d'approche puis d'attaque à des niveaux différents du SI.

Je présenterai alors les SIEM [4], des outils qui permettent de corréler les informations de plusieurs IDS, ainsi que les avantages qu'ils apportent dans la détection de ce type d'attaques.

En parallèle des travaux effectués sur mon sujet de stage, j'ai également contribué au travail d'une équipe de Conix Sécurity qui met en place et gère des systèmes de détection d'intrusions.

Je présenterai donc les objectifs et travaux effectués par cette équipe, puis je détaillerai les missions auxquelles j'ai participé durant mon stage, ainsi que les différents travaux effectués dans ce contexte.

Enfin, je terminerai ce rapport par un bilan de ces six mois de stage en entreprise, ainsi que par une présentation des bénéfices du stage, pour moi d'une part, et pour l'entreprise d'autre part.

4. Un SIEM, ou *Security Information Management System*, ou Système de Gestion de la Sécurité du Système d'Information. Un SIEM permet de centraliser les informations relatives à la sécurité du SI, notamment en collectant les *logs* de différentes sources (services système, IDS, etc...), puis en offrant la possibilité de les corréler entre eux afin de lever des alertes de sécurité plus ciblées. Enfin, les solutions SIEM sont souvent équipées de systèmes d'archivage à haute sécurité et haute disponibilité permettant d'utiliser les informations stockées lors de procédures judiciaires.

1

CONTEXTE DE RÉALISATION DU STAGE

1.1 L'entreprise

Conix Services est une Société de Services en Ingénierie Informatique (ou *SSII*[1]) créé en 1997, aux multiples activités. Elle est notamment spécialisée dans le conseil, la conception et la réalisation d'architectures informatiques et l'élaboration et la sécurisation de systèmes d'information. Ces différentes activités sont actuellement réparties dans 3 filiales :

- Conix Technologies et Services
- Conix Consulting
- R2M

Conix Consulting est une filiale créée en 2003, dont l'activité principale est le conseil concernant des problématiques de gouvernance, de conformité et d'industrialisation du SI.

Conix Technologies et Services est une SSII à part entière, spécialisée dans la réalisation et la sécurisation de Systèmes d'Information d'entreprise.

Enfin, R2M, qui a rejoint le groupe Conix en 2012, est un cabinet de conseil sur le risque, la conformité et d'autres approches organisationnelles.

Grâce à ses filiales offrant un champ d'intervention plus important, Conix possède un socle fort et une base de clients solide. Ceux-ci sont principalement des grands comptes, dans des domaines d'activités variés tels que la banque, la finance, l'industrie ou encore les télécoms et l'audio-visuel. Ceci lui permet de réaliser un chiffre d'affaires de près de 20 millions d'euros en 2012.

1. Une SSII, ou Société de Services en Ingénierie Informatique, est une entreprise spécialisée dans la réalisation de missions de services liées à l'informatique.

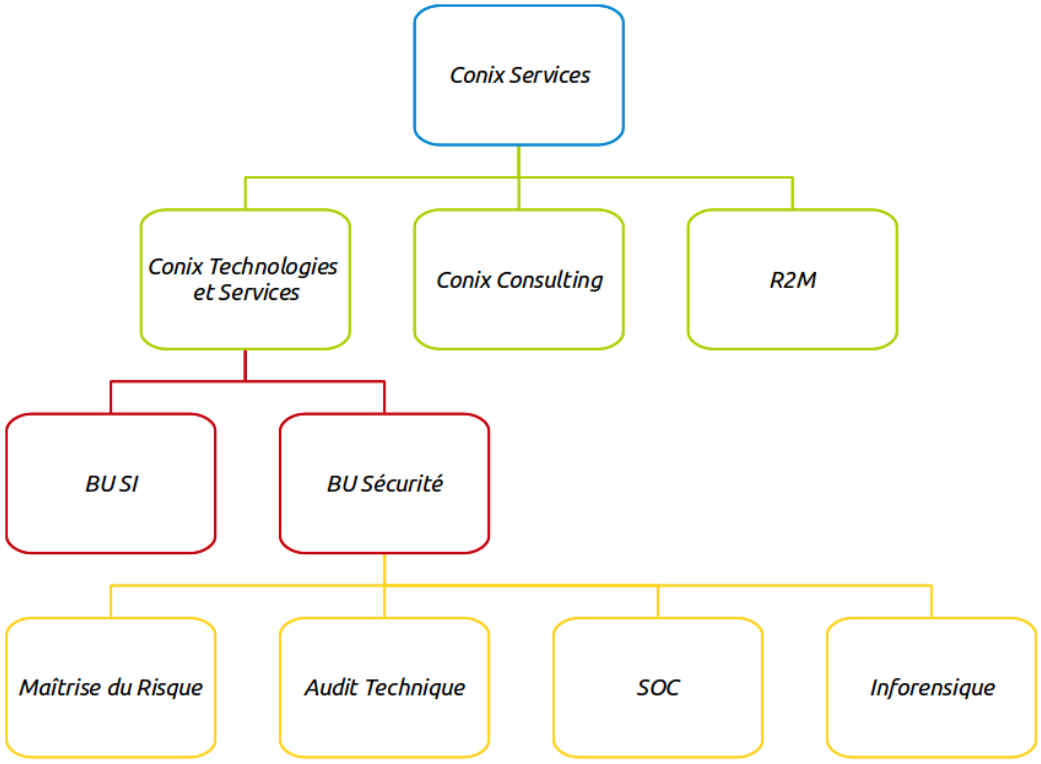


FIGURE 1 – Organigramme du groupe Conix

Conix intervient sur de nombreux projets sur lesquels elle a une forte expérience, afin de pouvoir garantir à ses clients une très bonne réactivité et une réelle valeur ajoutée. De plus, des contrats menés sur le long terme permettent de favoriser les échanges et augmentent la proximité des équipes avec les clients. Conix emploie ainsi près de 150 personnes réparties principalement en Île-de-France, à Pau et à Lyon.

La filiale Conix Technologies et Services s'articule autour de 2 *Business Unit* (ou BU) :

- Une *Business Unit* « Services Informatiques », qui effectue des missions de conseil en nouvelles technologies, développement et intégration d'applications métier, développement de projets Java, .Net et PHP et conception et mise en place d'architectures virtualisées ou d'environnements de travail collaboratif.
- Une *Business Unit* « Sécurité », qui est spécialisée en sécurité des systèmes d'information, analyse et gestion du risque, définition des politiques et objectifs de sécurité, pilotage SSI (*Sécurité du Système d'Information*), audit de sécurité et mise en oeuvre et exploitation d'infrastructures de sécurité.

Bien que la société Conix soit déjà fragmentée en différentes filiales et services, la *Business Unit* « Sécurité » possède elle-même plusieurs équipes, travaillant sur des sujets différents.

Une équipe participe à des missions de conseil en sécurité organisationnelle, notamment en évaluation et management du risque, au niveau de la réduction des risques liés à la sécurité du SI, ou au pilotage des solutions SSI (conseil aux RSSI [2] par exemple).

Une autre équipe intervient sur des missions d'audit de sécurité, permettant ainsi de contrôler et d'apporter des conseils afin de maximiser la sécurité des applications et installations évaluées. À l'heure actuelle, la majorité des activités consistent à auditer des applications Web ou mobiles.

Une nouvelle activité d'inforensique et de réponse à incident a récemment vu le jour au cours de l'année 2012 et propose ses services afin d'accompagner les clients dans la résolution et l'investigation suite à des intrusions ou à des attaques.

Enfin, le pôle sécurité propose également une offre de Management de la Sécurité du Système d'Information. Cette offre repose sur la conception, le déploiement et l'intégration de solutions de Supervision et de Management de la Sécurité du SI (notamment au travers d'un SIEM), ainsi que la mise en place d'un « SOC [3] » (*Security Operating Center*), qui permet au client d'externaliser la gestion de ses alertes de sécurité aux équipes de Conix.

C'est au sein de cette équipe que j'ai effectué mon stage, sous la direction d'Adrien Vernois, dont les objectifs sont présentés dans la partie suivante.

2. Dans une entreprise, le RSSI est le Responsable de la Sécurité des Systèmes d'Informations. Généralement rattaché à la Direction des Systèmes d'Information, il est chargé de toutes les missions en rapport avec la sécurité du SI et des données qu'il contient.

3. Un SOC, ou *Security Operations Center*, est dans une entreprise le service dédié à la gestion de la sécurité, notamment à la sécurité de l'information. Généralement, le SOC est responsable de la gestion des évènements de sécurité généré par les IDS.

1.2 Objectifs du Stage

Depuis plusieurs années, Conix Security a développé une expertise forte sur des solutions de supervision de l'état de sécurité d'un SI. Cette supervision s'effectue notamment au moyen de Systèmes de Détection d'Intrusions (ou IDS) placés à plusieurs endroits du SI.

Ces IDS agissent et détectent les activités malveillantes à partir d'une base de signatures des comportements suspects (ou, à l'inverse, à partir d'une liste des comportements autorisés). Ces signatures doivent donc être maintenues à jour afin de garantir un bon niveau de protection. Ces signatures sont généralement spécifiques à des outils d'attaques, ou à des scénarii d'attaques plus évoluées lancées contre la cible.

Le premier objectif de ce stage a été de déterminer une liste d'outils fréquemment utilisés dans des audits ou des attaques. J'ai ensuite cherché à analyser de façon précise le fonctionnement de ces outils ainsi que les traces qu'ils peuvent laisser sur le réseau ou sur le système afin de pouvoir les détecter.

Ensuite, j'ai cherché à atteindre un niveau supérieur en détectant des attaques plus complètes, en construisant des scénarii complets. Ces scénarii se composent de plusieurs phases, pouvant utiliser plusieurs outils. Ici aussi, la détection complète de ces scénarii est nécéssaire afin d'assurer un niveau de détection le plus avancé possible en étant au plus proche de l'état de l'art actuel. J'ai donc cherché à proposer des scénarii réalistes, utilisés « dans la nature » contre des systèmes d'information d'entreprises.

Enfin, ce stage devait également me permettre de monter en compétences sur plusieurs sujets, en particulier la détection d'intrusions, réalisée à plusieurs niveaux dans l'infrastructure d'un SI. De plus, il est possible d'obtenir un niveau de supervision plus élevé en utilisant les IDS ensemble et en corrélant les informations de chacun. L'apprentissage des solutions de type SIEM a donc été une part importante du stage. Je m'attarderai donc sur l'implémentation de ces scénarii au niveau des configuration des différents outils et au niveau des règles de détection associées.

2

LA DÉTECTION D'INTRUSIONS

Pour débuter ce rapport, je vais présenter de façon générale la détection d'intrusions afin de poser les bases nécéssaires à la bonne compréhension des techniques présentées plus après.

Je commencerai cette partie par une introduction à la sécurité informatique, afin de donner les notions et le vocabulaire récurrents liés à la détection d'intrusions.

Je poursuivrai ce rapport en décrivant la détection d'intrusions d'un point de vue théorique. J'aborderai notamment les caractéristiques des IDS, leur fonctionnement et les résultats attendus de ceux-ci.

Enfin, je terminerai cette partie par la présentation des différents outils utilisés en entreprise, et plus particulièrement par ceux utilisés lors de mon stage.

2.1 La Sécurité Informatique

Avant de pouvoir entrer dans le vif du sujet, il est important de poser quelques notions de base sur l'informatique, et notamment sur la sécurité informatique, puisque les travaux concernant la détection d'intrusions sont une partie des sciences informatiques.

2.1.1 Sécurité Informatique : La protection des données

Selon Wikipedia [Wik07], on peut définir la sécurité informatique comme suit :

> « La **sécurité informatique** est l'ensemble des moyens techniques, organisationnels, juridiques et humains nécessaires et mis en place pour **conserver, rétablir, et garantir** la **sécurité des systèmes informatiques**. Elle est intrinsèquement liée à la **sécurité de l'information** et des **systèmes d'information**. »

Bien que cette définition soit complète, elle utilise deux notions qu'il convient également de définir. Je vais donc définir la sécurité, et plus précisément la sécurité informatique, puis je détaillerai ce que l'on entend par *informatique* dans cette définition.

De manière générale, la sécurité informatique consiste à mettre en place et maintenir des techniques permettant de garantir que les ressources informatiques sont utilisées uniquement dans le cadre prévu par la Direction des Systèmes d'Informations. On doit ainsi maîtriser les 3 enjeux suivants :

1. La **Confidentialité** : Seules les personnes autorisées doivent pouvoir accéder aux ressources informatiques ;

J'utilise ici, par abus de langage, le terme « personne » pour désigner les entités qui ont accès aux ressources. En réalité, ce terme englobe les utilisateurs, les processus, les applications, etc...

2. L'**Intégrité** : Seules les personnes autorisées doivent pouvoir modifier les ressources informatiques ;
3. La **Disponibilité** : Les personnes autorisées doivent pouvoir accéder à tout instant aux ressources proposées ;

À cela, on peut également rajouter les 2 enjeux suivants, qui permettent d'atteindre un niveau de sécurité plus important :

4. La **Non-répudiation** : Une personne ne peut contester avoir effectué des actions sur les ressources informatiques ;
5. L'**Imputation** : Il doit être possible, à tout instant, d'identifier la personne qui aura effectué une action sur les ressources.

La sécurité informatique consiste donc à s'assurer, qu'à tout instant, les 5 propriétés précédentes des ressources informatiques sont assurées.

La définition précédente, tirée de Wikipédia, définie la sécurité informatique comme l'ensemble des moyens permettant de maintenir la sécurité des systèmes informatiques.

Or, on observe aujourd'hui que les services informatiques ne sont plus simplement responsable des moyens informatiques mis en place, mais qu'ils sont aussi et surtout responsables des données stockées et manipulées par ceux-ci. Il est d'ailleurs plus fréquent d'entrendre parler de « Direction des Systèmes d'Information » plutôt que de « Service Informatique ».

La sécurité informatique a donc dû évoluer et s'adapter pour aujourd'hui mettre en place et maintenir des compétences afin d'également sécuriser l'information utile des ressources informatiques. De ce fait, on pourra donc également parler de sécurité de l'information.

À mon sens, on pourrait donc utiliser la définition suivante de la sécurité informatique :

> « La sécurité informatique consiste à mettre en place et maintenir l'ensemble des moyens techniques et humains permettant de garantir que les ressources informatiques, et en particulier les données manipulées par celles-ci, seront disponibles en tout instant et que seules les personnes autorisées pourront y accéder et les modifier, sans qu'il leur soit possible de contester ces accès. »

2.1.2 Sécurité Informatique : La protection contre des attaques

J'ai donné dans la partie précédente une définition de la sécurité informatique. Cependant, bien qu'ayant présenté ce qu'était la sécurité, je n'ai pas abordé un point important : contre quoi est-il important de se protéger ?

Mettre en place une sécurité forte sur son Système d'Information a pour objectif d'empêcher, ou au moins de limiter, les intrusions sur le SI. Une **intrusion** est une violation de la politique de sécurité mise en place. Nous parlons alors d'une **attaque** lancée contre le système.

La suite des différentes phases de l'attaque composent le **scénario** de l'attaque. Dans la majorité des cas, nous tenterons de détecter une attaque en décelant chacune des étapes séparement. Le comportement jugé « anormal », engendré par l'intrusion elle-même par rapport à l'activité « normale » du système constitue la **signature** de l'attaque.

Une attaque informatique est la tentative d'exploitation d'une **faille**, ou vulnérabilité afin de procéder à une intrusion dans le SI. Ces vulnérabilités peuvent être de plusieurs types, et sont généralement des défauts de conception, de mise en place ou d'administration des différentes parties du système d'information [1].

1. On peut citer comme exemple des bogues dans le code source des applications, des défauts de conception ou de configuration des architectures ou encore d'administration.

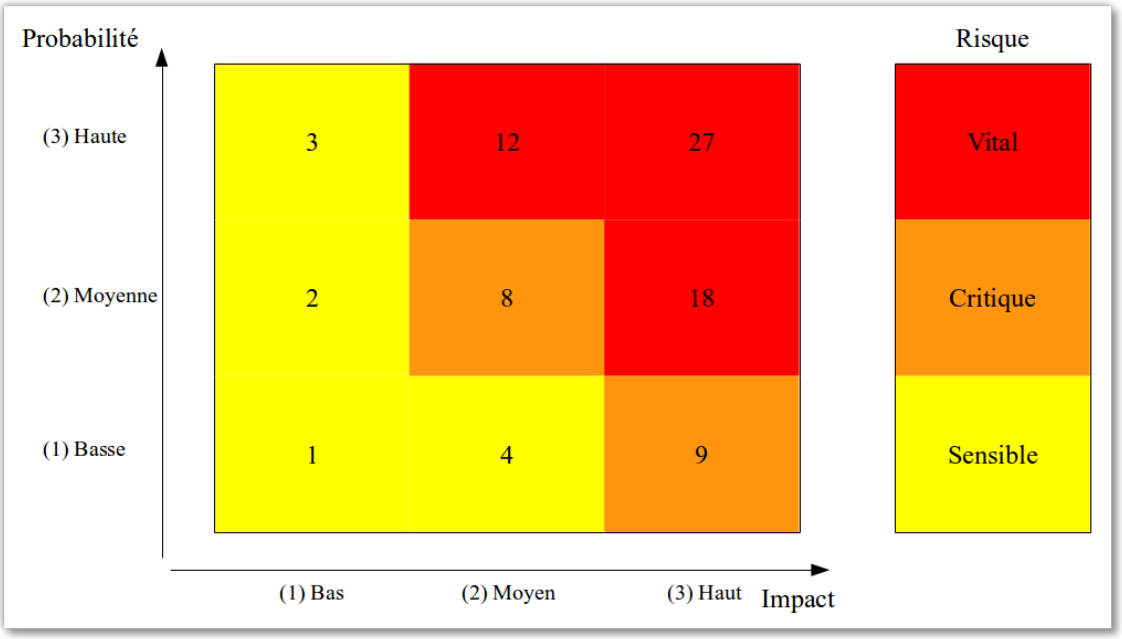


FIGURE 2 – Matrice de risques : Évolution du risque en fonction de l'exploitabilité d'une faille et des conséquences de l'exploitation

Une attaque est donc la tentative d'exploitation d'une faille sur une partie du système. Cette partie se nomme également un **actif**, ou *asset*. En sécurité, l'objectif est donc de chercher à se protéger des **risques** qui peuvent impacter les actifs du SI.

Le risque est fonction des menaces qui pèsent sur les assets. Ainsi, plus les vulnérabilités sont facilement exploitables et plus les conséquences de l'intrusions critiques pour l'activité, alors plus le risque sera élevé.

La matrice donnée en figure 2 présente l'évolution du risque en fonction de la criticité de l'exploitation d'une vulnérabilité, ainsi que la facilité d'utilisation de celle-ci.

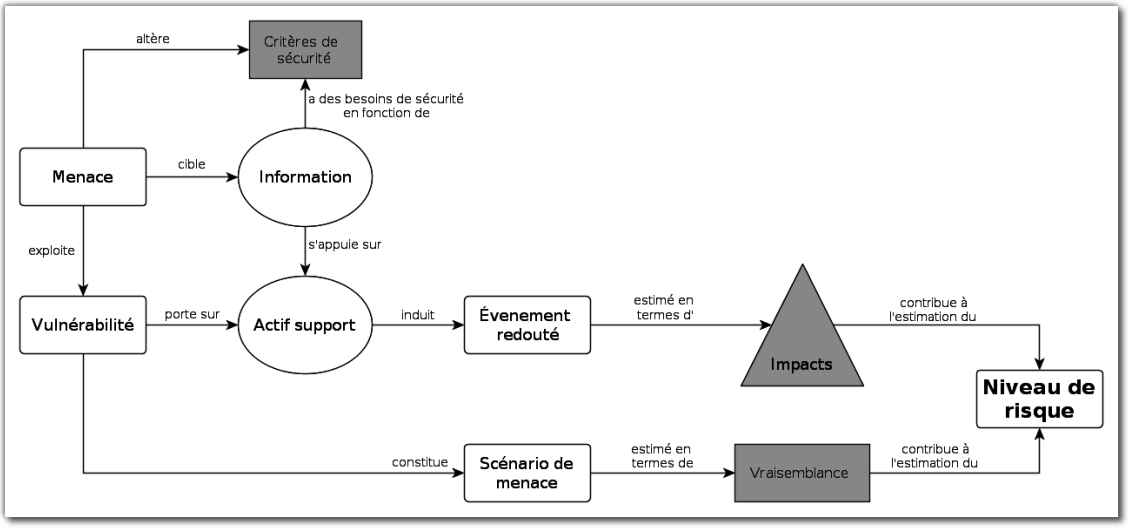


FIGURE 3 – Diagramme d'évaluation du niveau de risque d'un périmètre.

2.2 Approche théorique à la détection d'intrusions

Après avoir présenté succintement dans la première partie ce qu'était la sécurité informatique, je vais désormais faire de même pour la détection d'intrusions.

Je commencerai cette présentation en énonçant les différents enjeux de la détection d'intrusions du point de vue de la DSI [2].

Je continuerai en détaillant les principes de fonctionnement des IDS d'un point de vue général, en présentant notamment les architectures mises en place ainsi que le fonctionnement de la détection.

Enfin, je terminerai cette partie par une présentation des différents outils disponibles sur le marché de la détection d'intrusions, ainsi que leur fonctionnement global.

2.2.1 Caractéristiques des IDS

Un système de détection d'intrusions complet se compose de nombreuses parties, chacune ayant une tâche précise et essentielle dans le processus de détection.

Nous pouvons distinguer les différents blocs suivants :

- Les **sources de données**, à partir desquelles on vérifiera si une intrusion est en cours ;
- Le **moteur de détection**, qui va analyser les données reçues des sources précédentes afin de remonter des évènements ;
- La **réponse à la détection**. Suite aux évènements remontés par le moteur de détection, le système pourra choisir d'effectuer une action spécifique en réponse.

2. La DSI est la Direction des Systèmes d'Information. Ce service est celui qui met en place et gère les ressources informatiques en entreprise.

Les sources de données L'élément indispensable à tout système de détection d'intrusions est son panel de sources de données. En effet, c'est sur cette base que la totalité du système va se construire et la précision des données reçues de ces sources va conditionner la justesse des alertes émises par l'IDS.

La première tâche de l'IDS est donc de recueillir ces sources, puis de les normaliser dans un format commun. En effet, chaque application utilise son propre format de journalisation des évènements et il est essentiel de centraliser les informations dans un format standard à l'IDS.

Les IDS peuvent recevoir des données d'une ou plusieurs sources. Celles-ci peuvent être de nature très différentes. Nous pouvons notamment citer [3] :

- les systèmes d'exploitation ;
- des applications ;
- des équipements réseau (routeurs, commutateurs, ...) ;
- d'autres IDS ;

Les IDS basés sur des équipements réseau sont nommés *Network-Based Intrusion Detection System*, ou NIDS [4]. Ceux-ci sont interconnectés au réseau qu'ils monitorent et reçoivent la totalité du trafic à analyser.

Comme mentionné dans la liste précédente, un autre type de système de détection d'intrusions existe. Ils se nomment HIDS [5], ou *Host-Based Intrusion Detection System* puisqu'ils sont lancés comme application sur des ordinateurs, serveurs, smartphones (ou autres) et utilisent des informations tirées du système d'exploitation ou de certaines applications pour lever leurs alertes. Ces HIDS s'appuient sur sur des journaux d'évènements (ou *logs*), sur les accès à des fichiers sensibles du système (par exemple `/etc/passwd` sur un système Unix [6]), etc...

Nous pouvons ici observer que les données peuvent être de nature très différentes, et chaque IDS utilisera celles-ci dans le format qui lui est propre.

Une fois les données reçues, le système de détection d'intrusion va devoir analyser toutes les données reçues afin de déterminer si une activité anormale est en cours ou non. Cette action est réalisée par le moteur de détection.

Le moteur de détection Le moteur de détection d'un IDS est le coeur du système. C'est cette partie qui va analyser les données reçues puis remonter ou non des alertes.

Ce moteur peut se baser sur deux approches d'analyse :

1. Une approche par signature
2. Une approche comportementale

3. Liste non exhaustive.

4. Un NIDS, ou *Network-Based Intrusion Detection System*, est un système de détection d'intrusions basé sur des évènements provenant d'un réseau informatique.

5. Un HIDS, ou *Host-Based Intrusion Detection System*, est un système de détection d'intrusions basé sur des évènements provenant du système d'exploitation sur lequel il est lancé.

6. Un système de type Unix est un système d'exploitation dont le comportement est semblable à celui d'un système Unix. Les principaux systèmes Unix actuels sont les systèmes GNU/Linux, BSD, Mac OSX et Solaris

L'approche par **signature** est sans doute la méthode la plus utilsée par les IDS aujourd'hui. Elle consiste, comme son nom l'indique, à vérifier si les données reçues correspondent à une des signatures, correspondant à un comportement anormal, du système de détection d'intrusions.

Ces signatures peuvent être formulées de plusieurs manières, afin de couvrir le panel le plus important de cas détectables.

Si la source des données est le réseau, les signatures pourront par exemple être constituées des numéros de ports source et/ou destination, ou des adresses IP émettrice et/ou destinataire des packets.

Dans le cadre du monitoring des accès à un fichier, la signature pourra être le nom de l'utilisateur qui accède au fichier.

Ces exemples sont bien sûr déclinables à l'infini.

Le système de détection va donc remonter des alertes si les conditions définies dans la signature sont remplies par les données reçues. La détection par signature est une approche qui autorise par défaut tout le trafic, et lance des alertes sur le trafic jugé « anormal ».

L'approche de détection **comportementale** est fondamentalement différente de celle par signatures, bien que l'idée générale soit la même.

En lieu et place des signatures des attaques qui sont la base de la détection par signatures, l'approche comportementale va posséder des modèles de comportements « légitimes ».

Les données reçues par les IDS sont analysées de la même façon que lors de la précédente approche, à la différence que le système va chercher à détecter si les actions sont autorisées. Si elles ne le sont pas, elles lèveront des alertes.

L'apprentissage du modèle des comportements légaux peut se faire de différentes manières. La première est de partir d'une base vierge, et de rajouter à ce modèle des cas légitimes qui généreraient des évènements inutiles. La seconde technique est d'obtenir des statistiques des actions réalisées régulièrement et qui peuvent donc être considérées comme autorisées.

Contrairement à la première approche, nous pouvons observer que par défaut toutes les actions sont jugées comme anormales, et que des actions des administrateurs sécurité sont nécéssaires pour que les activités « normales » ne soient plus considérées comme illégitimes.

La réponse à la détection L'objectif de la détection d'intrusion est de détecter les intrusions, et surtout d'agir en réponse afin de limiter les actions qui peuvent être réalisées par l'attaquant. Pour cela, les IDS ont des capacités de réponse aux incidents.

Les possibilités de réponse sont multiples :

La première utilité d'un IDS est d'alerter lors d'une intrusion. Pour cela, ils disposent de plusieurs dispositifs, comme lever des alarmes dans l'interface de gestion, ou envoyer des mails aux ingénieurs sécurité. Les SIEM disposent également de fonctions plus avancées, comme la génération de graphes ou de rapports automatiques.

Cependant, un IDS peut également répondre de manière active aux intrusions. Dans ce cas, nous ne parlerons plus de système de détection d'intrusions, mais d'un système de **prévention** d'intrusions. Dans cette configuration, le système va réagir de lui même lorsqu'il recevra un évènement. Cette réponse peut être le blocage de l'action, l'ajout de

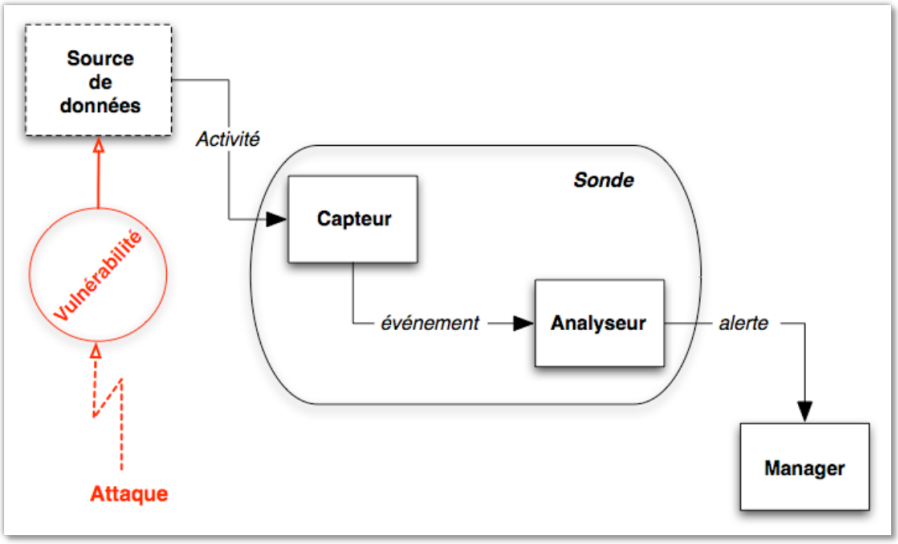


FIGURE 4 – Architecture d'une plateforme de détection d'intrusions. Article issu de l'article [BHM+06]

règles de pare-feu ou dans les *politiques utilisateurs* pour bloquer les actions futures de ce type.

2.2.2 Propriétés attendues des IDS

Les principaux enjeux de la détection d'intrusions, du point de vue de la DSI, sont de surveiller le niveau de sécurité du SI et de savoir, en temps réel, si des attaques sont en cours et si elles aboutissent ou non.

Nous attendons donc que les systèmes de détection d'intrusions possèdent certaines caractéristiques leur permettant d'être efficaces.

La première propriété attendue des IDS est d'apporter les informations les plus précises concernant les intrusions en cours. Notamment, nous cherchons à nous assurer que les alertes remontées soient fiables et pertinentes.

Premièrement, elles doivent être les plus fiables possible. La detection d'intrusions doit permettre aux responsables de la sécurité d'être alertés dès qu'une intrusion survient. Il faut donc que le nombre de faux négatifs (où aucune alerte n'est levée alors qu'une attaque est en cours) doit être le plus faible possible.

De la même façon, il est important qu'une alerte ne soit levée que si, et seulement si, une attaque est en cours, afin de réduire le temps de traitement des évènements non désirés. Le taux de faux positifs (alertes remontées sans cause réelle d'attaque) doit également être le plus bas possible.

On remarque, dans cette situation, que les deux cas sont opposés et qu'ils peuvent en un sens être liés.

Par exemple, si on choisit de créer des alertes pour chaque action entreprise sur le SI, le nombre de faux négatifs va très fortement baisser, puisque chaque opération, légitime ou non, aura pour conséquence la création d'une alerte. Cependant, les faux positifs seront en très grand nombre. A l'inverse, en ne remontant que très peu d'alertes il est possible de diminuer la quantité de faux négatifs, puisque seuls les cas très avérés d'attaques engendreront des évènements de sécurité. Cependant, on s'expose également à laisser passer des cas d'attaques plus larges qui n'auront pas été pris en compte.

L'un des plus grands défis de la détection d'intrusions est ici. S'assurer que les alertes ne sont causées que par des attaques réelles, et que dans le même temps chacune des attaques remonte des alertes aux administrateurs.

Je présenterai dans la partie suivante les 3 principaux outils que j'ai utilisé au cours de ce stage.

2.3 Principaux IDS

Les systèmes de détection d'intrusions sont multiples, et adaptés au type de données que l'on souhaite analyser.

Il est possible de ne retenir que deux grandes familles d'IDS :

1. les **Network-Based Intrusion Detection System**, ou NIDS, qui analysent des flux réseaux afin d'en tirer des informations ;
2. les **Host-Based Intrusion Detection System**, qui se basent sur des informations provenant du système d'exploitation.

J'ai principalement utilisé, au cours de mon stage, 3 IDS.

Le premier, **Snort** [7], est un NIDS utilisant une approche de détection par signatures. Il écoute le trafic d'une interface réseau et remonte des évènements si des paquets correspondent à une de ses signatures.

Le deuxième système de détection d'intrusions utilisé est **OSSEC** [8]. À la différence de Snort, c'est un *Host-based Intrusion Detection System*. Au lieu de surveiller le trafic réseau, il se base sur des actions effectuées sur le système d'exploitation pour générer des évènements de sécurité.

Enfin, j'ai également utilisé **Audit** [9], un HIDS dédié aux systèmes GNU/Linux. Celui-ci permet d'obtenir des informations précises sur les accès, par des utilisateurs ou des applications, à des fichiers ou répertoires de l'OS.

Je vais, dans la suite de ce chapitre, présenter ces trois systèmes de détection d'intrusions plus en détails, en insistant sur leur fonctionnement global, ainsi que sur le format des signatures qu'ils utilisent.

2.3.1 Snort

Comme je le mentionnais en introduction à cette partie, **Snort** est un outil de détection d'intrusions basé sur le réseau. Initialement développé par Martin Roesch, il l'est désormais par sa forte communauté et par la société SourceFire (récemment rachetée par Cisco). Malgré tout, Snort reste un logiciel libre diffusé sous licence GNU GPL ([FSF07]).

Disponible pour la plupart des plate-formes, je l'ai utilisé au cours de mon stage sur un système d'exploitation GNU/Linux *Debian Squeeze* [10]. L'application, ainsi que toute sa documentation, sont disponibles sur le site officiel de Snort : `https://www.snort.org/`.

7. Snort est un *Network-Based Intrusion Detection System*, qui permet de monitorer un réseau, en comparant les paquets transitant sur ce flux à des signatures prédéfinies. `https://www.snort.org/`

8. OSSEC est un *Host-Based Intrusion Detection System*, qui permet de réaliser des analyses de fichiers de *logs*, des contrôles d'intégrité, de la détection de *rootkits*. Il est compatible avec de nombreux systèmes d'exploitation, comme GNU/Linux, Windows, Mac OS, ou encore Solaris. `http://www.ossec.net/`

9. Le logiciel *Audit* est, sur les systèmes GNU/Linux, un système de détection d'intrusions qui permet de monitorer les accès à certains fichiers du système.

10. `https://www.debian.org/index.fr.html`

Snort est un outil de détection d'intrusions basé sur le réseau. Il analyse donc la totalité des données réseaux utilisant le protocole IP [11]. Ensuite, il analyse chacun des paquets reçus, et effectue une action si ceux-ci correspondent à une des règles définies.

Snort peut effectuer plusieurs actions à la réception d'un paquet jugé malicieux. La première action, la plus commune, est de créer un évènement afin d'alerter les administrateurs. Il est également possible d'effectuer d'autres actions, comme loguer les paquets reçus, activer d'autres règles Snort, bloquer les paquets, ou encore rejeter la connexion (en envoyant un paquet TCP *reset* ou ICMP *port unreachable*).

Dans cette dernière configuration, dans laquelle Snort bloque les paquets qui correspondent à ses règles, il ne se comporte plus comme un IDS, mais comme un **IPS** [12], ou *Intrusion Prevention System*. Dans cette configuration, au lieu de simplement détecter les intrusions, Snort réagit de façon active aux intrusions en les bloquant au moment où elles apparaîssent.

Les règles Snort étant basées sur la signature des paquets illégitimes, le moteur de Snort doit être capable d'analyser le contenu des flux réseaux. Afin de réaliser cette opération, Snort dispose donc d'un ensemble de décodeurs lui permettant d'analyser intrinsèquement le contenu des flux.

Les décodeurs accessibles sont :

- TCP
- UDP
- ICMP
- IP

Cela signifie qu'il est donc possible d'écrire des règles Snort en fonction des valeurs de certains champs ou options spécifiques au protocole utilisé.

Grâce à ses décodeurs, les règles Snort peuvent donc être très précises.

La base des signatures sont les source et destination du flux, à savoir les adresses IP et ports utilisés. Ensuite, les règles s'affinent, en intégrant des précisions sur le contenu des paquets.

En règle générale, la détection se basera sur le contenu du paquet. Il est donc possible d'utiliser le mot-clé `content:""` de Snort pour définir une chaîne de caractères qui doit être présente dans le paquet pour lever une alerte. Ce motif peut également être une expression régulière [13], plutôt qu'un contenu défini « en dur », au moyen du mot clé `pcre`.

11. Le protocole IP, ou *Internet Protocol*, désigne l'ensemble des protocoles utilisés sur des réseaux informatiques, notamment sur Internet. Il nécessite pour fonctionner la distribution d'adresses IP uniques sur le réseau (à tout instant, sur le réseau, deux machines ne peuvent avoir la même adresse IP.).

12. Un IPS, ou Système de Prévention d'Intrusions, est un logiciel permettant non pas de détecter les intrusions (comme le font les IDS), mais d'empêcher que ces intrusions se produisent. Le principe de fonctionnment reste cependant le même que pour les IDS, les actions considérées comme anormales par les règles de l'IPS seront bloquées.

13. Une expression régulière, ou rationnelle, est une chaîne de caractères, aussi appelée motif, qui décrit un ensemble de chaînes de caractères. Ces expressions sont souvent utilisées dans des applications afin de vérifier si les données saisies par l'utilisateur sont conformes à un format attendu.

La règle peut ensuite contenir des informations beaucoup plus précises sur le contenu du flux, ou sur le positionnement de ces motifs dans le paquet. Notamment, Snort peut décoder plus précisément le trafic HTTP [14] et il est possible de lui demander de ne rechercher des motifs que dans les *cookies* HTTP, dans les entêtes ou l'URI [15], ou encore filtrer sur des méthodes HTTP [16] ou sur des codes de réponses [17] particuliers. Enfin, les règles Snort peuvent également filtrer sur des numéros de séquences TCP, ou sur des `id` de datagrammes IP.

Exemples de règles Afin de mieux observer le principe de construction des règles Snort, je vais présenter ici plusieurs règles simples et plus avancées au travers d'un exemple simple : détecter une injection SQL [18] basique.

Cette première règle lève une alerte identifiée par le numéro 999001 et par le message `SQL Injection` si Snort détecte que la chaîne de caractères `SELECT * FROM` est transmise au serveur web sur le port tcp/80 (`$WEB_SERVER` est une variable Snort qui contient l'adresse IP du (ou des) serveur(s) Web).

```
alert tcp any any -> $WEB_SERVER 80 (msg:"SQL Injection"; content:"SELECT *
    FROM"; sid:1; rev:1;)
```

Listing 1 – Exemple de règle Snort : Détection d'une injection SQL.

La nouvelle version de la règle donnée ci-dessous apporte quelques précisions, tout en étant plus souple sur la détection. On précise que la connexion TCP doit être établie (`flow:established`), à destination du serveur (`flow:established,to_server;`) (ici `$WEB_SERVER`). Enfin, on assouplit la règle en effectuant une recherche de la chaîne `SELECT * FROM` non sensible à la casse des caractères (`nocase;`), mais en spécifiant qu'elle doit se trouver dans l'URI de la requête HTTP. On ajoute également un type à l'alerte, afin de faciliter le tri par les équipes d'exploitation des incidents.

```
alert tcp any any -> $WEB_SERVER 80 (msg:"SQL Injection (v2)"; flow:
    established,to_server; content:"SELECT * FROM"; nocase; http_uri;
    classtype:web-application-attack; sid:1; rev:2;)
```

Listing 2 – Exemple de règle Snort : Détection d'une injection SQL, version 2.

14. Le protocole HTTP, ou *HyperText Transfer Protocol*, est un protocole de communication client/serveur utilisé principalement par les navigateurs Web pour accéder à des serveurs contenant des données. Le protocole HTTP est un protocole de la couche application du modèle TCP/IP, reposant sur le protocole TCP.
15. Une URI, ou *Uniform Resource Identifier*, est la chaîne de caractères qui permet d'identifier de façon unique une ressource sur un réseau informatique (généralement sur Internet).
16. `GET`, `POST`, `HEAD`, etc...
17. https ://fr.wikipedia.org/wiki/Liste_des_codes_HTTP
18. Une injection SQL, ou *SQLi* est une attaque informatique visant à modifier les requêtes SQL exécutées par l'application attaquée afin d'induire un nouveau comportmeent et récupérer des informations depuis le serveur. Pour cela, l'attaquant envoie à l'application vulnérable un *payload* spécifique permettant de lancer l'attaque.

Enfin, la dernière version de la règle intègre une expression rationnelle qui va *matcher* sur une requête SQL plus large que dans les deux premières versions de la règle. Les options `Ui` de la *pcre* permettent (`U`) de manipuler l'URI décodée [19], et (`i`) de ne pas être sensible à la casse.

```
alert tcp any any -> $WEB_SERVER 80 (msg:"SQL Injection (v3)"; flow:
    established,to_server; content:"SELECT"; nocase; http_uri; pcre:"/
    SELECT.+FROM/Ui"; classtype:web-application-attack; sid:1; rev:3;)
```

Listing 3 – Exemple de règle Snort : Détection d'une injection SQL, version 3.

Snort est un NIDS performant, capable de traiter de grandes quantités d'évènements, et disposant d'un jeu de règles évolué permettant d'obtenir des informations très précises sur les incidents en cours sur le réseau.

Cependant, ses données ne fournissent des renseignements que si les intrusions utilisent le réseau, durant la phase d'infection ou d'exploitation. Pour monitorer des incidents différents, il faut utiliser d'autres types de systèmes de détection d'intrusions, comme les HIDS.

2.3.2 OSSEC

À la différence de Snort, **OSSEC** est un système de détection d'intrusions basé sur les systèmes d'exploitation plutôt que sur le réseau. Outil open-source maintenu par la société TrendMicro et la communauté, il est disponible sur `http://www.ossec.net/`.

Tout comme Snort, OSSEC est multiplateformes.

Au cours de mon stage, j'ai travaillé sur les versions Microsoft Windows et GNU/Linux d'OSSEC.

OSSEC a un mode de fonctionnement typique d'un IDS. Des agents sont déployés sur chacun des systèmes à monitorer et s'occupent de collecter les informations importantes. Ensuite, toutes les données sont centralisées sur un serveur OSSEC chargé de les analyser.

La portabilité d'OSSEC permet de déployer des agents sur de nombreux types de machines. Il est capable de récupérer des informations directement depuis les principaux OS : Microsoft Windows, GNU/Linux, *BSD et Unix en général. Cependant, la modularité d'OSSEC lui permet également de collecter des journaux d'évènements au format *syslog* depuis les principaux équipements réseaux utilisés dans les infrastructures. Enfin, les agents OSSEC supportent aussi le monitoring de serveurs de bases de données, comme MySQL ou Oracle, et les logs de nombreuses applications, comme *SSH*[20], l'authentification PAM

19. En cas d'utilisation des représentations hexadécimales de certains caractères dans l'URI, ils seront remplacés par leur valeur « en clair ». Exemple : La chaîne `SELECT%20*%20FROM` sera remplacée grâce à la clause `U` par la chaine `SELECT * FROM`.

20. SSH, ou *Secure Shell*, est le protocole de connexion et communication sécurisées à distance à des systèmes d'exploitation Unix.

d'Unix, les commandes `sudo` [21], des serveurs mails, FTP, Web, ou encore des firewalls ou des antivirus.

Les agents OSSEC ont la particularité de traiter principalement des fichiers de logs. Bien qu'il soit considéré comme un HIDS, il est également possible de le classer parmi les LIDS : *Logs-Based Intrusion Detection System*. Il dispose d'une liste des différents journaux d'évènements à surveiller, et les agents transmettent au serveur toute nouvelle entrée dans ces fichiers.

Cependant, les agents OSSEC disposent également d'autres fonctionnalités. Ils sont capables de réaliser des contrôles d'intégrité sur des fichiers et répertoires et sur des clés de Registre Windows, afin de pouvoir alerter à chaque modification des informations sensibles.

Les agents disposent également d'un système de détection des rootkits qui pourraient modifier des fichiers critiques de l'OS. Bien que j'ai eu l'occasion de manipuler la plupart des fonctionnalités d'OSSEC, je n'ai pas utilisé ce module de détection.

Dans une architecture OSSEC, les agents sont chargés d'effectuer le monitoring sur chaque système distant, et c'est le serveur qui collecte les données provenant de tous les agents et qui va réaliser l'analyse de celles-ci afin de remonter, le cas échéant, des alertes.

Le serveur OSSEC, disponible uniquement pour des systèmes d'exploitation GNU/Linux et *BSD, reçoit des agents des logs bruts, tirés des sources de données de ces derniers. La première étape de détection dans une infrastructure OSSEC est donc le décodage de ces logs. Ensuite, on peut écrire des règles génériques ne s'appuyant que sur ces logs décodés. Les règles OSSEC, comme celles de Snort, définissent la signature des actions jugées illégitimes sur le SI monitoré. On peut également donner à ces règles des priorités et des niveaux de criticité permettant d'accélerer le traitement des alertes et l'évaluation du niveau de sécurité du système.

Le fonctionnement interne d'un serveur OSSEC, et plus précisément son moteur de détection, est basé sur un format standard de stockage et de gestion des évènements. Les données reçues sont manipulées par des décodeurs qui, au moyen d'expressions rationnelles, vont extraire les informations importantes et nécéssaires à la détection, et les stocker dans des variables internes à OSSEC.

Les signatures définies dans les règles sont donc basées sur ces variables. Des alertes sont levées si les variables extraites des logs correspondent aux valeurs définies dans la règle.

Exemples de règles Tout comme pour Snort, je vais donner ici un exemple de configuration basique permettant de détecter tous les accès à des pages non présentes sur le site Web (code HTTP 404).

La configuration s'effectue en 2 parties. La première est la configuration de l'agent afin qu'il relève toute modification du fichier de logs du serveur Web présent sur la machine

21. Sur des systèmes Unix, la commande `sudo` permet à des utilisateurs standards de lancer des commandes avec les droits du *super-utilisateur* `root`.

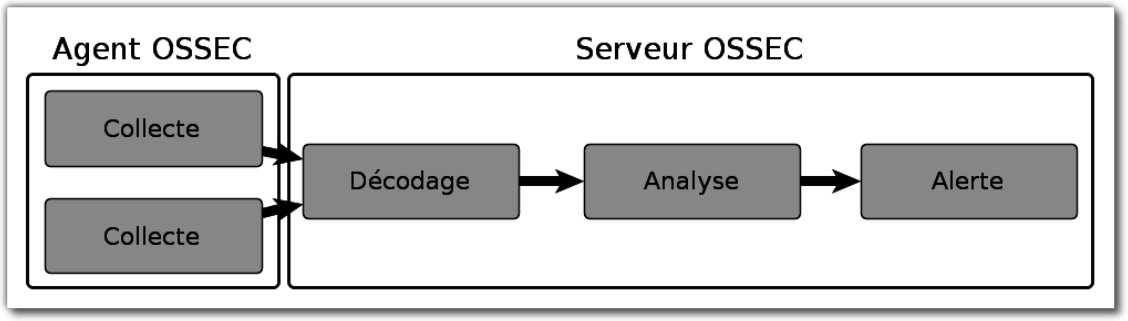


FIGURE 5 – Processus de traitement d'un incident par OSSEC. Image tirée de la documentation OSSEC : `http://www.ossec.net/files/auscert-2007-dcid.pdf`, page 14.

(ici un serveur Apache sur un système GNU/Linux Debian). La configuration se fait en seulement quelques lignes.

```
<localfile>
  <log_format>apache</log_format>
  <location>/var/log/apache2/access.log</location>
</localfile>
```

Listing 4 – Exemple de règle OSSEC : Configuration de l'agent pour le monitoring des logs Apache.

Les évènements récupérés par les agents Snort doivent être décodés afin de pouvoir écrire des règles sur ces contenus. Un décodeur OSSEC est très simple, il contient une expression régulière qui extrait des valeurs, et le nom de variables auxquelles affecter ces valeurs.

Dans notre cas, on extrait du log Apache : l'adresse IP source, la méthode HTTP, l'URI demandée et le code de retour HTTP.

```
<decoder name="apache_log">
  <regex>(\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}) - - [.+] "(\w+) (.+).*" (\d+)"</
    regex>
  <order>user,protocol,url,status</order>
</decoder>
```

Listing 5 – Exemple de règle OSSEC : Création d'un décodeur serveur pour les logs Apache.

Finalement, nous allons écrire une règle OSSEC simple, dans laquelle nous vérifierons la valeur du code HTTP. Si celui-ci est égal à 404, il faut remonter une alerte.

```
<group name="apache">
  <rule id="42001" level="0">
    <decoded_as>apache_log</decoded_as>
    <description>Apache Logs.</description>
  </rule>

    <rule id="42002" level="6">
```

```
    <if_sid>42001</if_sid>
    <status>404</status>
    <description>Apache Log : Access to an unknown page (404).</description
    >
  </rule>
</group>
```

Listing 6 – Exemple de règle OSSEC : Création de règles serveur pour les logs Apache.

Malgré qu'il nécessite une architecture de type client/serveur, OSSEC est très simple à déployer et mettre en place. Il intègre en effet un système de gestion centralisé des configurations et de déploiement automatique de la configuration sur les agents. De plus, OSSEC est un IDS très polyvalent puisqu'il est capable d'analyser n'importe quel fichier de logs. Il n'y a donc aucune modification à réaliser sur l'infrastructure de base ni de nouvel équipement à ajouter.

De plus, les possibilités offertes par OSSEC de vérification de l'intégrité de fichiers et de clés de Registre permettent ainsi d'obtenir une maîtrise plus fine du niveau de sécurité du SI.

J'ai tout de même trouvé une limite à OSSEC. Il est capable de détecter les modifications apportées à un fichier, mais pas de détecter les autres types d'accès (lecture, exécution). On doit pour cela utiliser d'autres IDS, et j'ai utilisé pour ma part *Audit*, un HIDS dédié aux systèmes GNU/Linux.

2.3.3 Audit

Sur les systèmes GNU/Linux, **Audit** est un système de détection d'intrusions qui permet de monitorer les accès à certains fichiers du système, en obtenant dans les journaux d'évènements des informations très précises sur le type d'accès au fichier, l'utilisateur qui réalise cet accès et d'autres informations plus précises, comme l'application qui est utilisée pour réaliser l'accès, le succès (ou non) de la tentative d'accès, etc...

Le fonctionnement d'Audit est très simple. Dans un fichier de configuration, il est possible de configurer le chemin d'accès au fichier, les types d'accès pour lesquels lever des alertes, et d'autres paramètres permettant d'affiner les cas dans lesquels créer des incidents. Le module noyau d'Audit intercepte ensuite tous les appels systèmes et lève des incidents si un de ceux-ci correspond aux règles définies dans le fichier de configuration.

Exemples de règles L'exemple de règle suivant génère des évènements à chaque accès par l'utilisateur `www-data` (utilisateur qui lance le serveur Web Apache sur le système) accède en lecture, écriture ou modification au fichier `/etc/passwd` (contenant les noms d'utilisateur et mots de passe) de l'OS.

```
-p war -w /etc/passwd -F uid 33
```

Listing 7 – Exemple de règle Audit : Accès en lecture ou écriture au fichier `/etc/passwd` par l'utilisateur `www-data` (uid:33).

Je ne présente ici que la gestion du monitoring des accès fichiers, mais Audit peut aussi effectuer d'autres tâches de détection d'intrusions, comme surveiller les connexions au système ou des accès réseaux, mais je n'ai utilisé dans ce stage que ses capacités de gestion des accès aux fichiers. C'est d'ailleurs pour cette raison que je l'ai utilisé, puisqu'il permettait de réaliser simplement ce dont j'avais besoin : savoir quand un utilisateur tentait d'accéder à des fichiers.

OSSEC est un HIDS très performant, notamment pour ses capacités de vérification de l'intégrité des fichiers. Cependant, il lui manque certaines précisions, comme le choix du type d'accès pour lesquels on souhaite avoir des alertes, ou simplement le fait de pouvoir remonter des alertes pour d'autres types d'accès que les modifications.

GNU/Linux Audit permet d'accéder à ces informations, de façon très précise et renforce ainsi le niveau de surveillance des accès aux fichiers sensibles de l'infrastructure.

Audit n'étant présent que sur les systèmes GNU/Linux, il est possible de trouver des solutions pour obtenir le même type de monitoring sur des OS Microsoft Windows. En particulier, on peut activer sur Windows les politiques d'audit d'accès, qui vont surveiller les accès à des fichiers ou des répertoires et écrire des *logs*. L'utilisation d'OSSEC pour recueillir ces journaux d'évènements fonctionne ensuite de la même façon que pour Audit.

On peut donc, en combinant OSSEC, Audit, et certaines fonctionnalités de Windows, obtenir un monitoring complet de l'accès aux fichiers et répertoires des serveurs et machines GNU/Linux et Windows.

2.3.4 Limitations de ces IDS

Les systèmes de détection d'intrusions représentent donc une part très importante de tout processus de sécurité dans une entreprise. Ils permettent d'obtenir des informations claires, détaillées et en temps réel sur les intrusions et, de manière générale, sur les atteintes portées à la politique de sécurité mise en place au sein du SI.

Malgré tout, ces IDS ont tout de même quelques inconvénients qui peuvent réduire leurs effets et, de fait, limiter leurs apports à la surveillance du périmètre.

En premier lieu, je vais aborder Snort. Étant basé sur le réseau, il se place en écoute sur celui-ci et analyse la totalité du trafic. Toutefois, ce point peut être nuancé en fonction des capacités physiques accessibles à Snort. Il doit en effet disposer des ressources matérielles (puissance de calcul, mémoire vive, accès réseau) suffisantes afin d'éviter la perte de paquets, et donc la perte d'information.

Le moteur de Snort s'appuie sur une file d'attente « circulaire » dans laquelle sont placés tous les nouveaux paquets. Snort doit donc récupérer les paquets dans cette file plus vite que celle-ci ne se remplit. Si les ressources sont sous dimensionnées, de nouvelles données pourront remplacer des anciens paquets réseaux non traités, et donc diminuer la précision des alertes remontées par l'IDS.

Pour éviter ce problème, il existe un deuxième système de détection d'intrusions réseau,

Suricata [22], qui, à la différence de Snort, utilise plusieurs threads. Il peut ainsi multiplier les processus de traitement des paquets réseau reçus, et donc accélerer celui-ci et vider la file d'attente plus rapidement.

Suricata permettrait ainsi d'accéder à des capacités de traitement très élevées [23].

Hormis le cas précis de Snort, j'ai également pu remarquer que les systèmes de détection d'intrusions ont de manière générale un inconvénient majeur, qui est intrinsèquement lié à leur fonctionnement.

Comme je le mentionnais dans la partie 2.2.1, les IDS peuvent utiliser deux types d'approches pour effectuer leur détection : une approche par signature, et une approche comportementale. Les deux approches nécessitent de définir en amont de la détection un ensemble de cas qui seront, pour la première approche, ceux dans lesquels il faudra lever des alertes, et dans la seconde, les cas pour lesquels l'IDS ne devra pas créer d'incidents.

Cependant, ces solutions ont le même problème : Que faire si l'IDS a à faire à un cas qui n'a pas été pris en compte ? Par exemple, si une intrusion est effectuée en utilisant une faille de type *O-day* [24] pour laquelle aucune signature existe, le système de détection d'intrusions utilisant ce type d'approche de détection ne créera pas d'alerte. Dans le cas inverse, si un utilisateur réalise sur le SI une opération légitime, mais qui n'aura pas été prise en compte dans l'IDS, alors celui-ci lèvera une alerte pour une action autorisée.

Dans le premier cas, on observe un cas dit « faux-négatif ». L'action réalisée est illégitime, mais aucune alerte de sécurité n'est créée. Dans le second cas, une intrusion est détectée alors qu'il n'y en a pas puisque le cas d'utilisation est autorisé. Cette situation est appelée « faux-positif ».

Ces deux cas peuvent également être inversés. Un IDS par signature peut utiliser une signature trop laxiste, qui remontera des évènements pour des actions légitimes (« faux-positifs »), et un système de détection d'intrusions par approche comportementale pourra penser qu'une intrusion est en réalité une activité légale (« faux-négatif »).

Le schéma de la figure 6 présente de façon graphique ces situations.

La mise en place d'un NIDS implique donc de mettre en place un processus clair et fiable de suivi des règles des IDS et surtout de maintien à jour de celles-ci, afin qu'elles aient le moins de retard possible sur l'état de l'art des attaques et vulnérabilités connues.

La mise en place d'un NIDS peut ne pas être efficace dans le cas où les flux réseaux sont chiffrés, comme lors de l'utilisation des protocoles HTTPs, IMAPs, etc...

Dans ces cas d'utilisation, il est impossible pour les IDS comme Snort d'accéder au contenu des flux réseaux qui transitent. Pour cela, il existe plusieurs scénarios permettant de conserver un monitoring efficace du périmètre.

La première solution est la mise en place d'un *reverse proxy* [25] en entrée du réseau.

22. `http://suricata-ids.org/`

23. `https://home.regit.org/2012/07/suricata-to-10gbps-and-beyond/`

24. Une faille dite « 0-day » est une faille qui n'a pas encore été dévoilée à la communauté et qui est donc encore inconnue des éditeurs de solutions de sécurité ou d'antivirus.

25. Un *reverse proxy*, ou proxy inverse, est un équipement logiciel ou matériel placé en entrée d'un réseau et permettant aux utilisateurs provenant de l'Internet d'accéder à des ressources placées à l'intérieur du

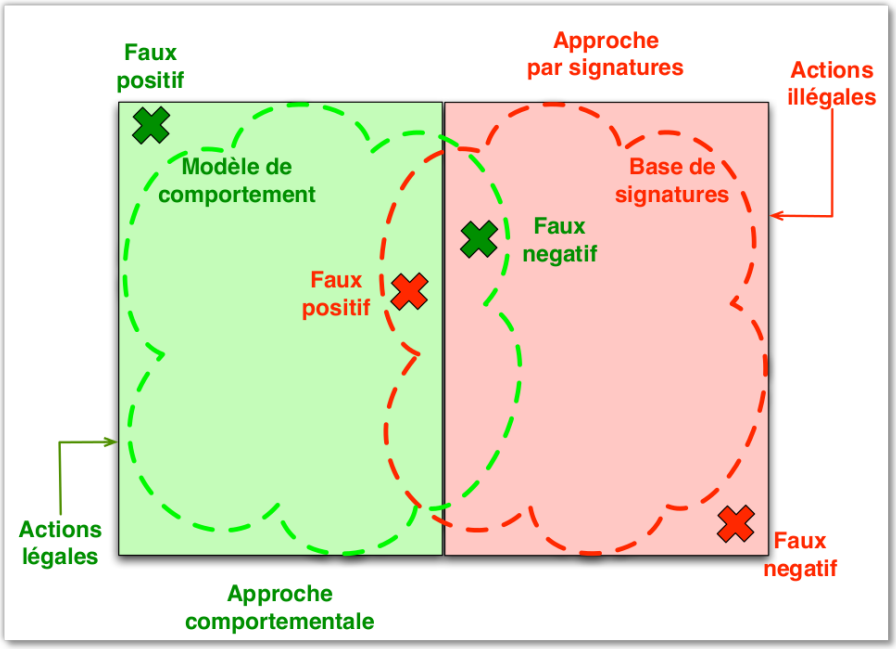


FIGURE 6 – Manque de fiabilité et/ou de pertinence si le modèle utilisé est incomplet ou incorrect. Image tirée du cours de Guillaume Hiet et Ludovic Mé sur les IDS donné dans le cadre du master Cryptis.

C'est ce proxy qui effectue toutes les étapes de chiffrement avec le client, puis l'intégralité du trafic entre ce proxy et les équipements internes est envoyée en clair. La sonde Snort peut alors être placée n'importe où entre ces deux machines pour analyser le trafic.

Si l'on souhaite augmenter la sécurité à l'intérieur du SI, la sonde Snort peut être placée directement derrière le *reverse proxy*, puis elle renverra la totalité du traffic de façon chiffré aux serveurs contactés.

Dans beaucoup d'architectures, la sonde Snort n'est pas placée en coupure du réseau (configuration de type *Main-In-the-Middle*[26]), mais plutôt en parallèle du réseau, recevant tout le trafic à destination des systèmes ou environnements que l'on souhaite surveiller (cette mise en place est généralement effectuée au moyen de *Port Mirroring* ou de TAP[27] réseau). Dans cette situation, la sonde reçoit la totalité du trafic, mais toujours chiffré (du fait de la réplication). Il est possible de permettre au serveur Snort de déchiffrer lui-même les données reçues en utilisant la clé privée du serveur Web. Ce genre de configuration est possible et sans dégradation du niveau de sécurité puisque les deux hôtes font partie du

réseau, le plus souvent après une phase d'authentification ou d'identification.

26. Une attaque dite par *Man-in-the-Middle* est une attaque au cours de laquelle l'attaquant se place entre la victime et le service qu'elle contacte, en détournant la totalité du trafic afin de pouvoir accéder au contenu des échanges. Si la victime est nommé *Alice*, le service qu'elle contacte *Bob* et que l'attaquant est *Ève*, alors, pour Alice, Ève se fera passer pour Bob et Bob dialoguera avec Ève en pensant le faire avec Alice.

27. Un TAP réseau est un dispositif réseau permettant de surveiller un réseau informatique sans le perturber. Les flux réseaux sont dupliqués afin qu'un dispositif réseau puisse accéder aux données qui transitent, sans modifier ces flux ni dégrader la qualité de service.

même domaine de confiance (la sonde Snort devient alors un asset critique au même titre que le(s) serveur(s) Web pour lesquels elle dispose des clés de chiffrement SSL).

Comme je l'ai expliqué dans cette partie consacrée aux IDS, on peut remarquer qu'ils ont d'excellentes capacités de détection. Leur format de règles simple et adaptable permet d'écrire en peu de temps des schémas de détection de cas très précis d'attaques.

De plus, les évènements remontés par ces systèmes de détection d'intrusions contiennent des informations très ciblées sur l'origine de l'attaque, dans le cas de Snort par exemple, et dans d'autres cas sur la réussite (ou non) de celle-ci. Toutes ces données sont très importantes et utiles lors de l'analyse de ces intrusions par des équipes d'inforensique ou par les administrateurs.

J'ai également pu remarquer qu'il pouvait survenir des cas où l'IDS n'avait pas le comportement escompté, et qu'il était possible de rencontrer des cas de faux-positifs ou faux-négatifs.

Mais il est possible que dans certains cas, les informations croisées de plusieurs systèmes de détection d'intrusions permettent de confirmer ou infirmer si une attaque est réellement en cours ou non.

C'est dans cette optique que je me suis intéressé à l'utilisation des SIEM, qui ont les capacités de corréler les évènements de sécurité provenant de plusieurs IDS.

2.4 Améliorer la fiabilité de la détection : la corrélation d'évènements

Comme je l'ai présenté dans la partie précédente de ce rapport, les IDS comme Snort ou OSSEC sont très spécifiques à la source de données qu'ils monitorent.

Snort permet d'obtenir des informations précises sur le contenu transitant sur le réseau au cours de l'attaque ainsi que sur les deux équipements qui communiquent. Cependant, il manque des données sur le résultat de l'attaque, et il est de plus totalement inefficace pour la surveillance de serveurs ou de postes de travail.

Pour palier à ce problème, j'ai montré qu'il était possible d'utiliser des HIDS, qui se basent sur des informations provenant du système d'exploitation, mais également sur les journaux d'évènements des applications « critiques » de l'infrastructure (comme des serveurs Web, *Active Directory*, anti-virus, etc...).

Dans de nombreux cas, les informations de tous les IDS présents sur l'infrastructure de détection sont importantes et complémentaires. Chaque système de détection d'intrusions donne des informations précises sur une partie spécifique des intrusions. Disposer de toutes ces données dans une même base commune permettrait donc d'obtenir des alertes beaucoup plus précises sur l'état de sécurité de l'infrastructure monitorée.

C'est dans cette optique que s'inscrit la corrélation d'évènements. Celle-ci s'effectue au sein d'un SIEM et va donc consister à agréger les alertes de plusieurs IDS puis de les relier entre elles afin de remonter des alertes plus précises.

Agissant comme un système de détection d'intrusions à part entière, la première tâche du SIEM est donc de collecter, tout comme un IDS, des données provenant de plusieurs sources. Ces données peuvent être des données « brutes », même si dans la plupart des cas les SIEM manipulent des évènements provenant d'équipements de sécurité, comme des systèmes de détection d'intrusions, des systèmes d'authentification ou des pare-feux.

Le but du SIEM étant de corréler les différents évènements, il doit pour cela les manipuler dans un format unique, indépendemment du format source. Le SIEM procède donc à une uniformisation, dans un format standard (comme par exemple le format IDMEF [28]), des différentes données qu'il reçoit.

Après avoir obtenu tous les évènements dans un même format, un SIEM peut effectuer les étapes de corrélation d'alertes. Celles-ci se basent sur des directives, qui sont similaires à des règles d'IDS comme celles présentées dans la partie 2.3.

Une règle d'IDS par signature contient les éléments qui permettent de distinguer les actions « légitimes » de celles qui ne le sont pas. Une directive de corrélation est sensiblement du même type. Elle se compose des identifiants de plusieurs règles d'IDS, et elle va ensuite générer des évènements de criticité plus ou moins élevée en fonction des alertes reçues.

28. Le format IDMEF, ou *Intrusion Detection Message Exchange Format* est un format conçu par l'IETF (dans la RFC 4765 (`http://www.rfc-editor.org/rfc/rfc4765.txt`)) afin de stocker et partager les informations de façon standard entre plusieurs IDS.

Plus le SIEM détecte d'évènements qui permettent de confirmer qu'une intrusion est en cours, plus il pourra remonter des alertes de criticité plus élevée. De même, il est possible pour le SIEM, si les IDS possèdent des signatures pour obtenir ce genre d'informations, de détecter qu'une tentative d'exploitation a échoué, et donc de ne lever qu'une alerte de plus faible criticité.

Grâce à la combinaison des évènements de plusieurs sources, on peut donc obtenir des alertes beaucoup plus précises que si elles avaient été traitées séparément. Ainsi, on peut diminuer le nombre de faux positifs et de faux négatifs générés par les IDS et donc augmenter la qualité de la supervision.

Bien que la principale fonctionnalité d'un SIEM du point de vue de la détection soit le moteur de corrélation, celui-ci possède d'autres fonctions qui peuvent faciliter le traitement et la gestion des incidents de sécurité.

Sans être exhaustif, on peut citer la génération automatique de graphiques et de rapports permettant une visualisation plus rapide et aisée de l'état de sécurité du périmètre surveillé. Un exemple de graphiques générés est donné en figure 7.

Afin de suivre l'évolution des alertes de sécurité créées par le SIEM, ce dernier peut généralement s'interfacer à un système de gestion de tickets, afin de coordonner le suivi entre tous les acteurs liés à la sécurité de l'infrastructure.

Enfin, la plupart des solutions de type SIEM disposent d'un système d'archivage des incidents et de toute trace ayant conduit à la génération de ceux-ci, afin de disposer de sauvegardes complètes des évènements survenus sur le SI. Pour garantir une meilleure sécurité, les données stockées pourront être chiffrées [29], signées [30], hachées [31], ou stockées sur des architectures distribuées et/ou dupliquées. La sécurité mise en place sur la sauvegarde de ces traces permettra de s'assurer qu'elles ne seront ni perdues ni modifiées au cours du temps, et permettra donc de les utiliser à des fins inforensiques ou juridiques.

Un SIEM, grâce à ses capacités d'aggrégation et de corrélation des évènements provenant de sources multiples, ainsi que ses fonctionnalités de *reporting* et de stockage sécurisé des *logs*, est un élément presque indispensable de toute architecture de détection d'intrusions.

Durant mon stage, j'ai utilisé le SIEM OSSIM, qui est déployé chez de nombreux clients de Conix Security. Je vais, dans la prochaine partie, le présenter plus en détail, énumérer

29. Chiffrer des données, ou chiffrement, consiste à appliquer aux données d'origine (ou données « *en clair* ») une fonction de chiffrement afin de rendre ces données impossibles à lire pour les personnes qui ne possèdent pas la clé de déchiffrement associée.

30. Signer des données, ou signature, consiste à appliquer aux données sources une fonction qui va générer une valeur unique associée aux données sources et à la personne qui effectue la signature. La moindre modification des données d'entrée et/ou de l'identifiant du signataire modifiera radicalement la signature créée. On peut ainsi vérifier que les données sont bien émises par une personne donnée et qu'elle n'ont pas été modifiées.

31. Hacher des données, ou hachage, consiste à appliquer aux données sources une fonction de hachage qui va associer à ces données une valeur unique (nommée *hash* ou condensat) permettant d'identifier cette donnée.

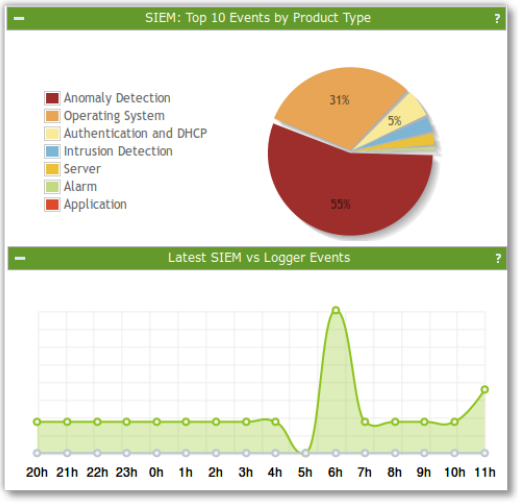


FIGURE 7 – Exemple de graphiques réalisés par le SIEM OSSIM. On peut observer dans le graphique du bas l'évolution du nombre d'évènements remontés dans le SIEM par heure, et dans celui du haut la répartition des sources des évènements. On remarque que ceux-ci ne proviennent pas que de systèmes de détection d'intrusions, mais aussi des systèmes d'exploitation, d'applications, ou encore des journaux d'authentification.

les fonctionnalités qui sont utilisées lors de la surveillance d'infrastructures clientes, et notamment les fonctions de suivi d'incidents et de corrélation d'évènements.

2.5 OSSIM : *Open-Source SIEM*

OSSIM, ou *Open-Source SIEM* est la version *Open-Source*[32] du SIEM *Alienvault Unified Security Management*, développé et commercialisé par la société Alienvault [33] [34] [35].

Cet outil est fourni sous la forme d'un système d'exploitation complet, basé sur un OS GNU/Linux Debian *Squeeze*. Il intègre les applications qui forment le cœur du SIEM (nommé OSSIM), mais également de très nombreux systèmes de détection d'intrusions avec des configurations complètes pour assurer leur fonctionnement au sein du SIEM. En particulier, OSSIM installe par défaut Snort et OSSEC, c'est pour cette raison que j'ai choisi d'utiliser ces différents IDS au cours de mon stage.

OSSIM étant un SIEM, il doit agréger des évènements de sécurité provenant de nombreux IDS afin de disposer du maximum d'informations pour afficher les indicateurs de sécurité les plus proches possible du niveau de sécurité réel du SI. Pour cela, il fonctionne selon une architecture clients/serveur.

Des sondes OSSIM [36] sont déployées sur les points stratégiques du réseau. Chaque sonde contient un certain nombre d'IDS qui vont effectuer leurs tâches de collecte, analyse et remontée d'évènements. Les agents OSSIM installés sur chaque sonde vont alors transmettre via le réseau chaque nouvel incident créé par les IDS au serveur OSSIM.

Ce dernier doit alors centraliser chacun des évènements qu'il reçoit, puis les traiter afin de les rendre disponibles au sein de son interface de reporting, et tenter de les corréler entre eux si certains correspondent aux règles définies.

Enfin, OSSIM dispose également d'un *logger*, un dispositif lui permettant d'archiver de façon sécurisée la totalité des logs qu'il reçoit, pour ainsi les mettre à disposition des personnes compétentes lors d'interventions inforensiques ou judiciaires.

Parmi les nombreuses fonctionnalités d'OSSIM, je n'ai utilisé au cours de mon stage que la corrélation des évènements de sécurité remontés par les différents SIEM. Cette corrélation permet, en liant entre elles plusieurs alertes de bas niveau obtenues par des IDS tels que Snort ou OSSEC, d'obtenir des alertes précises liées aux incidents qui surviennent sur le périmètre monitoré.

De plus, en se basant sur différentes sources afin de créer ces alertes de sécurité, le nombre de faux-positifs et faux-négatifs peut être diminué, puisque la présence (ou l'absence) de certains évènements permet de confirmer ou infirmer qu'une intrusion est (ou non) en cours.

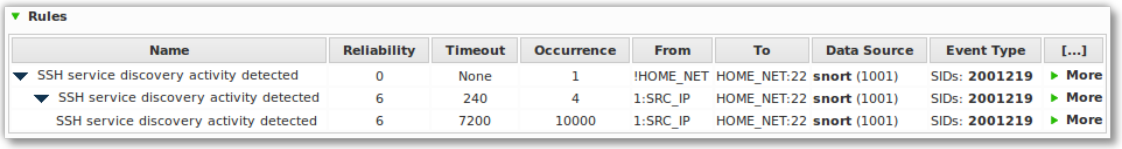
32. Le terme *Open-Source* désigne les logiciels dont la licence d'utilisation respecte certains critères, comme notamment l'accès au code-source, la libre distribution et la possibilité d'en créer des travaux dérivés.

33. La version commerciale intègre certaines fonctionnalités, comme des reporting plus évolués, qui ne sont pas accessibles dans la version Open-Source.

34. Version Open-Source : `http://www.alienvault.com/open-threat-exchange/projects`.

35. Version complète : `http://www.alienvault.com/products-solutions`.

36. Les sondes OSSIM sont des serveurs physiques présents dans l'infrastructure du SI.



Rules

Name	Reliability	Timeout	Occurrence	From	To	Data Source	Event Type	[...]
SSH service discovery activity detected	0	None	1	!HOME_NET	HOME_NET:22	**snort** (1001)	SIDs: **2001219**	More
SSH service discovery activity detected	6	240	4	1:SRC_IP	HOME_NET:22	**snort** (1001)	SIDs: **2001219**	More
SSH service discovery activity detected	6	7200	10000	1:SRC_IP	HOME_NET:22	**snort** (1001)	SIDs: **2001219**	More

FIGURE 8 – Exemple de directive de corrélation OSSIM. Plus les IDS détectent des scans SSH, plus le SIEM augmente la criticité de l'alerte. Directive issue d'OSSIM (en version Open-Source).

Les directives de corrélation d'OSSIM se présentent sous la forme de fichier XML [37] et contiennent les identifiants des évènements attendus par le système et le niveau de criticité associé à chaque étape de la corrélation.

Exemple de directive de corrélation Pour illustrer le fonctionnement des directives OSSIM, j'ai choisi une directive de corrélation issue de l'ensemble des directives inclues dans le serveur OSSIM en version Open-Source.

Cette directive affecte par défaut une criticité de 0 à chaque évènement consistant en une tentative de connexion SSH. Elle va ensuite corréler ces mêmes évènements entre eux, et augmenter la criticité de l'attaque si les IDS remontent de nombreux évènements en peu de temps. Les deux paliers de détection sont 4 évènements en 4 minutes, et 10000 évènements en 2 heures.

Cette directive est présentée dans la figure 8.

37. Le format XML, ou *Extensible Markup Language*, est un format de représentation des données générique et extensible, permettant de structurer les informations de façon hiérarchique.

En conclusion

Les systèmes de détection d'intrusions deviennent une part importante et non négligeable des processus de sécurité mis en place dans toute infrastructure informatique. Il est donc nécéssaire que ces IDS puissent couvrir la totalité du périmètre des systèmes d'information.

J'ai montré au cours de la présentation des différents IDS que j'ai utilisé dans mon stage qu'ils étaient très variés et permettaient d'obtenir des informations sur les évènements de sécurité qui surviennent à chacun des niveaux de l'infrastructure.

Les systèmes de détection d'intrusions basés sur le réseau, comme Snort, permettent d'apporter des données extrêmement précises sur les incidents de sécurité qui peuvent survenir sur un réseau d'entreprise, au moment où ils apparaissent.

De la même façon les HIDS qui utilisent les évènements et les informations disponibles sur le système d'exploitation hôte apportent des informations claires sur la réussite ou non des attaques. Des HIDS comme OSSEC permettent également, en s'appuyant sur les logs d'autres outils, de préciser les informations qu'ils remontent aux responsables de la sécurité.

J'ai également montré que ces technologies n'avaient pas que des avantages pour la détection. Les alertes émises par chaque IDS ne sont pas assez pertinentes pour obtenir une vision globale des intrusions, malgré qu'elles apportent des informations extrêmement précises sur le périmètre que ces systèmes surveillent. Les NIDS peuvent apporter une vision extérieure de l'attaque, au moyen des adresses IP des attaquants, tandis que seuls des IDS systèmes permettront de connaître le résultat de l'attaque ou de détecter la présence de malwares.

J'ai ensuite tenté de démontrer que les SIEM permettent de combiner les informations de tous les IDS présents sur l'infrastructure monitorée.

Les SIEM ont la capacité, au moyen notamment de règles de corrélation, de n'extraire que les informations essentielles des différentes sources de données disponibles au sein de l'architecture de détection et de les sublimer pour n'obtenir qu'une information concentrée et beaucoup plus pertinente sur les intrusions en cours.

Les SIEM, et la détection d'intrusions en général, est une activité importante de la sécurité, et c'est dans ce cadre que j'ai effectué mon stage de fin d'études de Master.

Durant ce stage, j'ai dû dresser une liste d'attaques fréquemment lancées contre des systèmes, pour pouvoir ainsi les détecter au moyen d'IDS. Conix Security, l'entreprise dans laquelle j'ai réalisé mon stage, utilisait et déployait déjà chez des clients des infrastructures de monitoring.

En particulier, Conix déploie le SIEM OSSIM que j'ai présenté dans la partie 2.5. C'est pour cette raison que j'ai utilisé au cours de mon stage ce SIEM et les différents IDS que j'ai présenté ici, qui sont intégrés par défaut dans OSSIM.

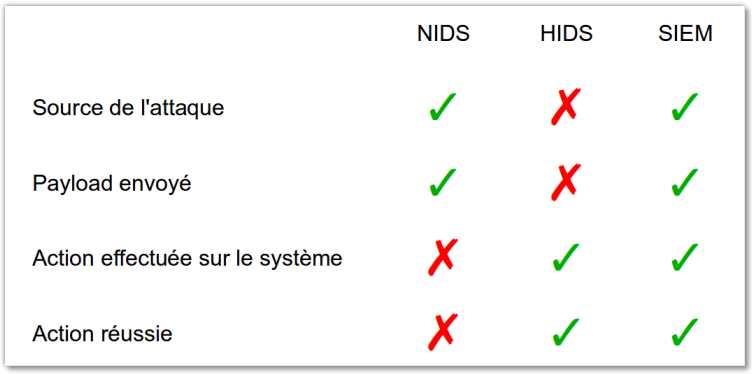


FIGURE 9 – Comparatif des fonctionnalités des IDS réseau, système et des SIEM, dans le cas d'attaques transitant, en premier lieu, par le réseau.

Après avoir présenté les différents outils de détection que j'ai utilisé, je vais désormais présenter les différentes attaques que j'ai tenté de détecter au cours de mon stage.

3

MISE EN PRATIQUE - SCÉNARIOS D'ATTAQUES

J'ai montré dans la précédente partie que la détection d'intrusions, pour être efficace, nécessitait que les règles des systèmes de détection d'intrusions soient maintenues à jour et correspondent à l'état de l'art des attaques connues.

C'est dans cette optique que j'ai réalisé mon stage de fin de Master en Sécurité Informatique. L'objectif était d'obtenir une liste de scénarios d'attaques fréquemment utilisées pour corrompre des infrastructures informatiques, et écrire les règles d'IDS permettant de détecter ces différents scénarios.

Afin de prendre en main les outils de détection d'intrusions utilisés par Conix, mon stage a débuté par la détection d'outils d'audit. Dans un second temps, j'ai travaillé sur des scénarios d'attaques plus évolués, utilisant plusieurs phases d'attaques, et pouvant être détectées par plusieurs systèmes de détection d'intrusions sur l'architecture cliente.

Dans cette première partie, je vais détailler les différents outils d'audit que j'ai cherché à détecter, leur fonctionnement, et surtout leurs particularités permettant de différencier ces actions anormales de celles qui sont légitimes sur le périmètre. Je donnerai également les règles simplifiées et configurations créées pour détecter ces outils [1].

1. Les règles complètes seront données en annexe A

3.1 Outils d'Audits Automatisés - Détection NIDS

J'ai principalement étudié au début de mon stage des outils dédiés à des attaques Web, ainsi que quelques outils spécifiques à d'autres types d'attaques.

Les attaques sur des sites Web sont les attaques les plus répandues contre des systèmes d'informations d'entreprises. En effet, les applications Web sont très rapides à développer et déployer, et elles permettent au personnel de l'entreprise d'accéder aux ressources internes avec une grande facilité. Cependant, les statistiques montrent que beaucoup d'entre elles sont vulnérables à des attaques très simples.

Ces attaques étant simples à exploiter, de nombreux outils d'exploitation « automatique » de ces failles sont disponibles sur Internet. Développés à la base pour permettre d'auditer rapidement et simplement des applications Web, ces logiciels sont finalement très souvent utilisés pour lancer des attaques.

Il est donc très courant d'observer que des attaquants se servent de ces outils pour lancer des attaques sur des plateformes clientes, et il nous a donc semblé judicieux de consacrer le début de ce stage à la détection de ces outils.

Ces outils d'audit Web ont une seconde particularité : ils sont tous lancés contre des cibles à distance, à travers le réseau. J'ai donc voulu déterminer s'il était possible de détecter ces outils en utilisant uniquement leur empreinte réseau.

Cela me permettait également de débuter ce stage en utilisant seulement Snort, un système de détection d'intrusions basé sur le réseau, et ainsi mieux maîtriser cet outil.

Je vais ici détailler en premier lieu la méthode que j'ai utilisé pour analyser chacun de ses outils, puis je présenterai ces outils, leur fonctionnement et les règles de détection associées.

3.1.1 Analyse réseau d'outils d'audit Web

L'analyse d'outils dédiés à des audits ou à des attaques peut être effectuée de plusieurs façons.

La première consiste à analyser le programme lui-même. S'il est *Open-Source*, il est possible de récupérer son code source puis d'en comprendre le fonctionnement. Si le programme est protégé par des droits qui empêchent l'accession au code source de façon légale, cette étape d'analyse est impossible. Il faut alors se contenter des informations disponibles dans les documentations à propos du fonctionnement de l'outil.

Cependant, cette analyse n'est pas toujours exhaustive, ni à jour par rapport aux versions du logiciel disponibles. J'ai donc opté pour une autre technique, qui permet d'obtenir une vision plus pratique du fonctionnement d'un outil. Elle consiste à utiliser ces logiciels en situation réelle, contre des applications Web, afin d'observer le comportement réel des applications.

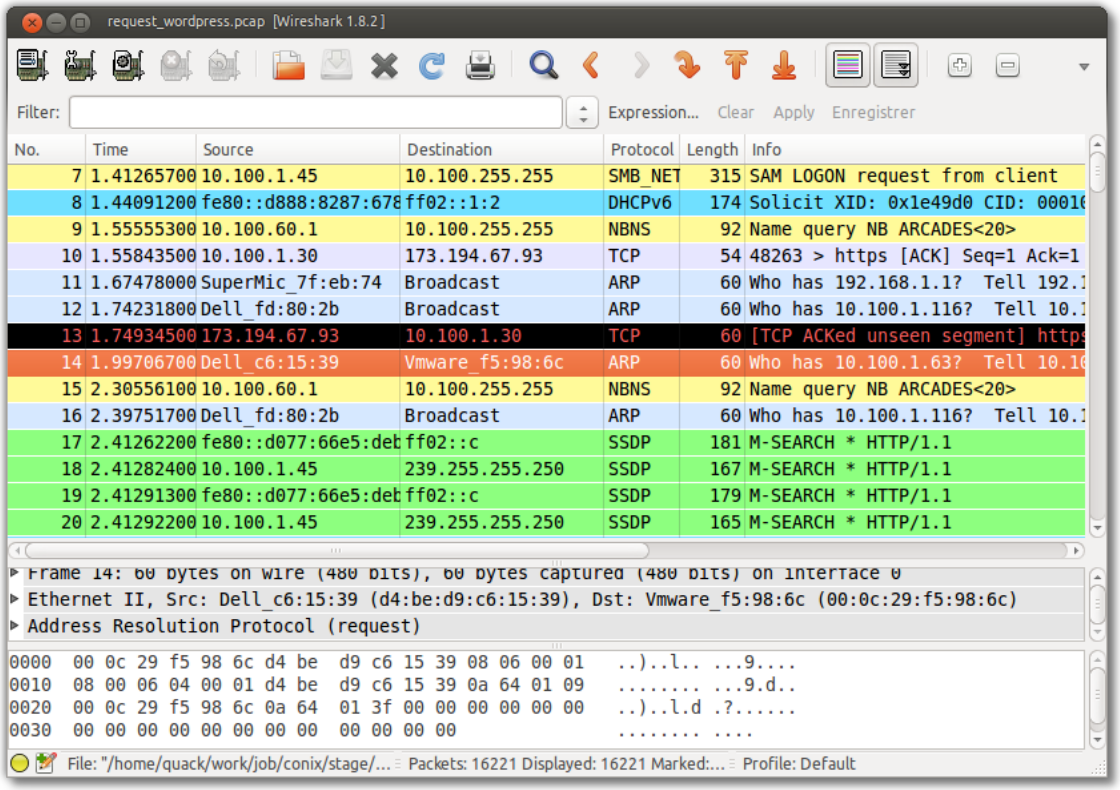


FIGURE 10 – Exemple de capture réseau analysée dans Wireshark

Cette technique d'analyse nécessite donc de mettre en place :

- Une serveur Web et une ou plusieurs applications Web vulnérables ;
- Une machine « d'attaque » sur laquelle seront installés chacun des logiciels analysés.

Afin de ne pas perturber le trafic interne de l'entreprise, ni de risquer que le serveur Web vulnérable ne soit exposé sur Internet et puisse être victime d'attaques et d'intrusions non désirées, j'ai choisi de n'utiliser que des machines virtuelles, non connectées à Internet, pour ainsi limiter leur impact.

Les attaques étant lancées contre les applications Web à travers le réseau, il est nécéssaire d'analyser le trafic réseau généré. Pour cela, j'ai réalisé des captures du réseau pendant les phases d'attaque au moyen de Wireshark [2] et tcpdump [3].

Ces captures réseau sont ensuite analysables dans des outils spécialisés, comme Wireshark par exemple (voir un exemple en figure 10), afin d'en tirer les motifs anormaux permettant de distinguer les outils d'attaque, que je vais présenter dans la prochaine partie.

2. `http://www.wireshark.org/`
3. `http://www.tcpdump.org/`

526	39.3898320	10.100.1.30	108.162.203.32	HTTP	285	GET /should/not/exist.html HTTP/1.1
548	40.1268970	10.100.1.30	108.162.204.32	HTTP	275	GET /readme.html HTTP/1.1
570	40.1407790	10.100.1.30	108.162.203.32	HTTP	298	GET /wp-includes/js/tinymce/tiny_mce.js HTTP/1.1
738	40.1736490	10.100.1.30	108.162.204.32	HTTP	290	GET /wp-includes/js/autosave.js HTTP/1.1
754	40.1773860	10.100.1.30	108.162.203.32	HTTP	315	GET /wp-content/themes/twentyten/languages/twentyten.pot

FIGURE 11 – Requêtes envoyées par Blind Elephant à l'application Web cible.

3.1.2 Outils analysés

J'ai étudié, au début de mon stage, 11 outils qui sont effectivement détectables par des outils de détection d'intrusions réseau.

Blind Elephant `http://blindelephant.sourceforge.net/`

Blind Elephant est un outil Open-Source permettant d'obtenir le nom de l'application [4] utilisée par un site Web, ainsi que sa version.

L'analyse du trafic HTTP entre l'attaquant et la cible permet d'observer les requêtes qui sont envoyées, ainsi que les réponses du serveur. Je donne, dans la figure 11 les premières requêtes envoyées par Blind Elephant.

Pour déterminer la version du CMS [5] utilisé par un site Web, Blind Elephant compare certaines pages bien précises disponibles sur le serveur avec ces même pages, dont il possède le *hash*. J'ai remarqué que chaque attaque lancée débute par une requête sur la ressource `/should/not/exist.html` qui est, normalement, inexistante sur la plupart des applications Web. C'est en utilisant cette page que Blind Elephant détermine le CMS utilisé. Ensuite, il effectue plusieurs requêtes sur des pages spécifiques à chacun des CMS.

La première requête reste la même à chaque tentative. C'est donc cette requête que j'ai choisi de détecter, afin de remonter des alertes pour une tentative d'attaque par Blind Elephant.

De ce fait, la règle Snort est relativement simple. La règle va créer des évènements à chaque requête HTTP `GET`, sur la ressource `/should/not/exist.html` [6].

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS
    BlindElephant"; content:"GET"; http_method; content:"/should/not/exist.
    html"; http_uri; sid:1;)
```

Listing 8 – Règle Snort de détection de Blind Elephant.

4. Par exemple : Wordpress, Joomla, etc...
5. Un CMS, *Content Management System*, est une application Web permettant la gestion et la mise à jour dynamique de sites Web. Les plus connus sont Wordpress(`https://wordpress.org/`), MediaWiki(`http://www.mediawiki.org/wiki/MediaWiki/fr`), ou encore Spip(`http://www.spip.net/`).
6. Pour faciliter la lecture de ce rapport, les règles Snort données ici sont volontairement tronquées pour n'en garder que l'essentiel. Les règles plus détaillées sont données dans en annexe A.1, page 87

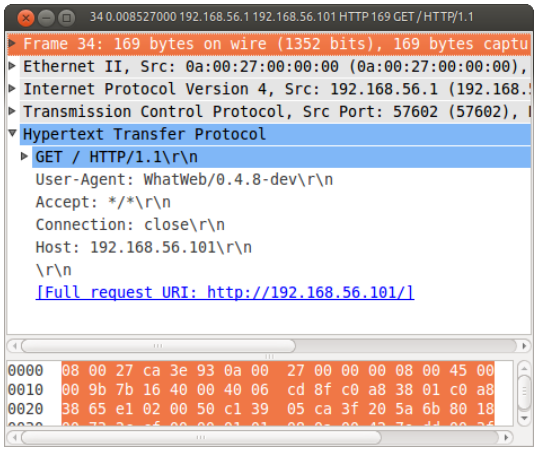


FIGURE 12 – Requête HTTP envoyée par WhatWeb à une application cible

WhatWeb `http://www.morningstarsecurity.com/research/whatweb`

WhatWeb est un outil d'audit similaire à Blind Elephant. Il obtient des informations précises, comme les applications utilisées, les modules activés, etc..., sur le serveur Web cible. Ces informations sont précieuses lors des audits ou des attaques puisqu'elles permettent de préciser les attaques à lancer, en ayant une connaissance complète de la cible. Ces données proviennent des entêtes HTTP, et ne sont donc pas intrusives à proprement parler.

Une analyse du contenu des paquets HTTP (comme présenté dans la figure 12) montre que le *User-Agent* utilisé par WhatWeb est `WhatWeb/0.4.8-dev`. Cela signifie que l'outil s'annonce sur le réseau. La détection est alors simplifiée puisque ce champ est directement accessible dans Snort [7].

La règle Snort est donc plutôt simple, puisqu'il faut simplement détecter le User-Agent dans la requête HTTP.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS
    WhatWeb"; content:"User-Agent: WhatWeb"; nocase; http_header; sid:2;)
```

Listing 9 – Règle Snort de détection de WhatWeb.

Skip Fish `https://code.google.com/p/skipfish/`

SkipFish est dédié à la reconnaissance des applications Web, et à la recherche de vulnérabilités de base. À la différence des 2 outils précédents, SkipFish effectue une reconnaissance active de sa cible. Les précédents outils tirent leurs informations des données présentes dans des communications « légitimes » entre un utilisateur et le site Web.

SkipFish est « actif » puisqu'il initie, de lui-même, des connexions « non légitimes » avec le serveur. L'outil permet d'obtenir une vision globale de l'application Web auditée ainsi

7. Cependant, je montrerai dans la partie 3.2 que ce type de détection a des certaines limites.

que son organisation. Pour obtenir la totalité des informations sur les pages accessibles, SkipFish effectue une requête sur la première page du site, liste tous les liens disponibles, puis relance cette opération sur tous les liens découverts. Il peut de cette façon afficher aux auditeurs une cartographie complète du site.

En analysant plus en détails le fonctionnement de ce logiciel, on remarque qu'il est difficile de le détecter de façon « comportementale ». En effet, toutes les requêtes lancées sont considérées comme normales puisqu'elles consistent à simplement demander une page au serveur. Les actions illégitimes sont le nombre de requêtes effectuées dans un très court laps de temps, et la recherche de vulnérabilités. Cependant, cette dernière ne peut pas non plus être détectée, puisqu'elle est effectuée hors-ligne [8].

J'ai donc privilégié ici une approche de détection par signature, en repérant une spécificité dans le *User-Agent* utilisé par SkipFish. En effet, il se camoufle en utilisant un *User-Agent* du navigateur web Mozilla Firefox [9] en y ajoutant, à la fin, la version de SkipFish utilisée, sous la forme `SF/$VERSION`.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS
    SkipFish"; content:"User-Agent"; nocase; http_header; pcre:"/User-Agent
    :.*SF\/\d+\.\d+/i"; http_header; sid:3;)
```

Listing 10 – Règle Snort de détection de SkipFish.

WebSploit `https://sourceforge.net/projects/websploit/`

WebSploit est un framework d'audit Web, disposant de plusieurs modules dédiés à des tâches spécifiques. De ses nombreux modules, je n'en ai retenu que 2, qui me semblaient être les plus intéressants : `pma` et `apache_users`.

Le premier d'entre eux est capable de déterminer si *PhpMyAdmin* [10] est installé sur le serveur Web et, le cas échéant, va obtenir son URI. Le second module étudié va lister l'ensemble des répertoires `public_html` [11] présents sur le système.

L'analyse de ces deux modules m'a permis d'observer qu'ils étaient tous les deux basés sur le même procédé. Ils utilisent chacun une technique d'attaque dite par « brute-force ». Cette technique consiste à tester toutes les combinaisons possibles jusqu'à en trouver une qui soit correcte. Cette technique est cependant améliorée par WebSploit puisqu'au lieu de tester tous les cas possibles un par un, l'attaque se base sur un dictionnaire, une liste de valeurs fréquemment rencontrées. Par exemple, le module `apache_users` testera les valeurs `root`, `admin`, `user`, `mail`, etc...

8. SkipFish ne détecte pas à proprement parler des vulnérabilités. Il déduit, à partir des pages Web, les failles qui pourraient être présentes sur le site.
9. `https://www.mozilla.org/fr/firefox/fx/`
10. `http://www.phpmyadmin.net/home_page/index.php`
11. Ce système, proposé par le serveur Web Apache, donne à chaque utilisateur du système la possibilité de possèder un dossier dans son répertoire personnel (par exemple) qui lui permet de gérer ses pages Web. Il a la particularité de rendre tous ces dossiers accessibles depuis Internet via une URI de la forme `/~<user_name>/`. `http://httpd.apache.org/docs/2.2/howto/public_html.html`

En inspectant le dictionnaire utilisé par le module `pma` de WebSploit, je me suis rendu compte qu'il utilisait énormément de variations basées sur la phrase « phpmyadmin ». Les $\frac{3}{4}$ des requêtes [12] contiennent cette chaîne. J'ai donc utilisé cette particularité et basé la détection de ce module sur la présence de cette chaîne dans la requête HTTP.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS
    WebSploit Module 'web/pma'"; content:"GET"; http_method; content:"/
    phpmyadmin"; http_uri; sid:4; )
```

Listing 11 – Règle Snort de détection du module `pma` de WebSploit.

La détection du second module, `apache_users`, est la même que pour le précédent module. Le dictionnaire contient une liste de 9000 noms d'utilisateurs possibles, qui seront tous testés. La règle va donc chercher les URI qui débutent par `/~`.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS
    WebSploit Module 'web/apache_users'"; content:"GET"; http_method;
    content:"/~"; http_uri; sid:5; )
```

Listing 12 – Règle Snort de détection du module `apache_users` de WebSploit.

La détection de ce type d'outils est efficace, cependant, elle a également des limites, atteintes par l'utilisation du brute-force. Si les 9000 valeurs du dictionnaire de WebSploit sont tentées, Snort va remonter autant d'évènements. Je présenterai donc plus loin dans ce rapport des techniques qui permettent de limiter la quantité d'alertes générées par ces IDS.

WebSecurify `https://www.websecurify.com/desktop/`

Je vais terminer la présentation des scanneurs Web par WebSecurify, qui est un scanner de vulnérabilités Web. Il détermine si une application Web est vulnérable à certaines attaques, en tentant de les exploiter.

Tout comme les précédents outils, WebSecurify réalise un *brute-force* de l'application basé sur un dictionnaire des *payloads* [13] permettant d'exploiter des failles courantes. La détection se fera de la même façon que pour SkipFish, c'est à dire en détectant dans le *User-Agent* la présence de la chaîne « WebSecurify » qui témoigne de l'utilisation de l'outil.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS WebSecurify";
    content:"User-Agent"; nocase; http_header; content:"Websecurify";
    nocase; http_header; distance:1; sid:6;)
```

Listing 13 – Règle Snort de détection de WebSecurify.

12. 76,7%, 66 requêtes sur 86.
13. Le *payload*, ou « charge utile », est le contenu réellement envoyé et manipulé par une application. Dans le contexte d'une attaque informatique, ce terme désigne le contenu ou le code exécutable qui va permettre de déclencer une attaque et/ou nuire à un système.

XSSer `http://xsser.sourceforge.net/`

XSSer est un outil de découverte et exploitation de failles XSS [14]. Basé sur un dictionnaire, les *payload*s utilisés sont récurrents, tout en comportant une partie aléatoire, censée permettre de ne pas se faire détecter par des IDS. Malgré tout, la partie « commune » de chaque requête est détectable.

Le dictionnaire de XSSer étant très diversifié, il n'y a pas de motif permettant de détecter tous les cas d'utilisation. J'ai donc dû imaginer un nouveau scénario de détection pour cet outil. J'ai choisi d'écrire deux règles Snort qui vont *matcher* [15] sur deux valeurs présentes dans plusieurs *payload*s d'XSSer. Il faudra ensuite vérifier la présence de ces deux évènements dans les journaux de Snort pour s'assurer que l'attaque est lancée par l'outil et non par une attaque manuelle.

La première règle correspond à la présence de la chaîne `><SCRIPT>alert('`, fréquemment utilisée dans des attaques de ce type. La seconde cherche le code Javascript permettant d'afficher des images : `<IMG SRC=`.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS XSSer 1"; content:
    "><SCRIPT>alert('"; nocase; http_uri; sid:7;)
```

Listing 14 – Règle Snort de détection du premier payload de Xsser.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS XSSer 2"; content:
    "<IMG SRC="; nocase; http_uri; sid:8;)
```

Listing 15 – Règle Snort de détection du second payload de Xsser.

DarkMySQLi

L'outil suivant, DarkMySQLi, permet d'exploiter automatiquement des failles de type « injection SQL ». L'étape de découverte n'est pas présente, et l'outil permet seulement d'automatiser les tâches répétitives d'exploitation réalisées manuellement par un auditeur.

L'outil est capable d'exploiter des injections SQL [16] afin d'obtenir la liste des bases de données, des tables, des utilisateurs de la base, ou encore de récupérer le contenu complet de plusieurs tables.

Pour cela, DarkMySQLi utilise quinze requêtes SQL différentes, permettant chacune d'exploiter des failles différentes. Chaque *payload* a été analysé et une règle pour chacun a été écrite. Afin de limiter la quantité de règles dans ce rapport, seule une d'entre elles est donnée ici, et la totalité des règles écrites pour la détection des attaques lancées par DarkMySQLi est donnée en annexe, page 90.

14. Une attaque de type XSS, ou *Cross-Site Scripting*, est une attaque Web visant à injecter dans une page Web du contenu afin de lancer des actions dans le navigateur des visiteurs de la page.
15. Remonter des informations si le paquet contient une certaine valeur.
16. Le langage SQL, ou *Structured Query Language*, est un langage informatique structuré permettant de manipuler les informations d'une base de données.

La règle suivante permet d'énumérer les bases de données présentes sur le serveur.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--dbs'
    "; content:"concat(0x1e,0x1e,COUNT(*),0x1e,0x20)"; nocase; http_uri;
    content:" FROM information_schema.schemata WHERE schema_name!=0x";
    nocase; http_uri; sid:9;)
```

Listing 16 – Règle Snort de détection de l'exploitation d'une injection SQL par DarkMySQLi afin d'obtenir les bases de données.

Fimap `https://code.google.com/p/fimap/`

Afin de continuer la détection d'outils dédiés à des attaques spécifiques, je me suis penché sur FImap, spécialisé dans l'exploitation de vulnérabilités de type « File Inclusion », qui consistent à accéder, à distance, à des fichiers du serveur Web depuis l'application, ou à inclure des pages Web illégitimes dans les pages du site.

De manière générale, l'attaquant tente d'accéder au fichier `/etc/passwd`, qui contient la liste des utilisateurs et leur mot de passe sur les systèmes Unix. C'est également à ce fichier que FImap tente d'accéder. On peut donc le détecter de deux façons :

1. Détecter, dans l'URI des requêtes HTTP, la présence du nom d'un fichier connu, comme `/etc/passwd` ;
2. Vérifier si le User-Agent contient le nom de l'outil.

J'ai utilisé la deuxième technique, qui me paraissaît plus précise et en conformité avec les précédentes règles écrites.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS (msg:"AUDIT_TOOLS FImap";
    content:"User-Agent\: fimap"; nocase; sid:10;)
```

Listing 17 – Règle Snort de détection de FImap.

sqlsus `http://sqlsus.sourceforge.net/`

Enfin, sqlsus est un outil dédié à l'exploitation d'injections SQL sur des applications utilisant des bases de données MYSQL. Son fonctionnement est le même que les outils précédents utilisant un dictionnaire.

La détection est ici basée sur le premier *payload* envoyé par sqlsus au site web audité. Ce payload est très spécifique et permet d'identifier avec une certaine assurance que l'outil utilisé est sqlsus.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS (msg:"AUDIT_TOOLS SQLsus";
    content:"UNION ALL SELECT BINARY"; nocase; http_uri; content:"CONCAT(0
    x5,1,0x7)"; nocase; http_uri; sid:11;)
```

Listing 18 – Règle Snort de détection de sqlsus.

XSSF `https://code.google.com/p/xssf/`

XSSF est un framework d'exploitation de vulnérabilités XSS sur des pages Web, développé par Ludovic Courgnaud lors de son stage de fin d'études de Master chez Conix en 2010. Cet outil est radicalement différent des outils précédents, puisqu'il ne réalise pas automatiquement l'exploitation de la vulnérabilité.

Les outils que j'ai pour l'instant étudié utilisent des dictionnaires pour déterminer comment exploiter une vulnérabilité, ou utilisent des *payload*s génériques sur les applications. XSSF est différent dans le sens où il nécessite que la vulnérabilité soit découverte et exploitable. XSSF se contente d'établir un canal de communication avec la victime au travers d'une faille XSS pour ensuite lancer des attaques spécifiques.

XSSF n'exploitant pas de faille, il est plus complexe de le détecter. En effet, il n'envoie pas de payload pour exploiter la faille XSS, seuls les modules sont envoyés à la victime.

La première solution envisagée a été d'écrire une règle Snort pour chacun des modules présents. Cependant, cela implique d'écrire de nouvelles règles pour tous les nouveaux plugins. Cela va également créer énormément de règles Snort supplémentaires, ce qui risquerait de créer des temps de traitement plus longs.

J'ai alors analysé le code source de XSSF plus en détail, ainsi que toutes les communications réseaux. En observant les communications au moment de l'envoi du premier payload, permettant d'établir le tunnel de communication, je me suis rendu compte que les fonctions Javascript envoyées, et spécifiques étaient toutes préfixées par la chaîne `XSSF_`.

J'ai donc écrit ma règle de détection Snort sur cette particularité.

```
alert tcp any any -> $HOME_NET any (msg:"AUDIT_TOOLS XSSF"; content:"
    function XSSF"; nocase; sid:12;)
```

Listing 19 – Règle Snort de détection de XSSF.

BEEF `http://beefproject.com/`

BEEF, ou « *The Browser Exploitation Framework* », est un framework d'exploitation de vulnérabilités Web semblable à XSSF.

Son fonctionnement est similaire à celui de XSSF. Il envoie une portion de code Javascript permettant de prendre le contrôle du navigateur. Comme pour XSSF, la détection est réalisée en cherchant dans les paquets HTTP la présence de fonctions Javascript préfixées de `BEEF_`.

```
alert tcp any any -> $HOME_NET any (msg:"AUDIT_TOOLS BEEF"; content:"
    function beef"; nocase; sid:13;)
```

Listing 20 – Règle Snort de détection de BEEF.

Les premiers outils présentés ici étaient tous des outils d'exploitation automatique de failles Web, permettant d'effectuer une reconnaissance de la cible en affichant, par exemple, des informations sur le serveur ou l'application, ou encore en récupérant l'organisation complète du site web. Certains de ces outils étaient également utilisés pour détecter la présence de vulnérabilités sur ces applications.

Ils utilisent des techniques dites « bêtes », c'est à dire qu'ils essayent énormément d'attaques en lançant de nombreuses requêtes. Ces outils sont relativement simples à détecter pour plusieurs raisons. En premier lieu, ils s'annoncent tous au serveur Web attaqué, soit en donnant leur nom dans le champ *User-Agent* des entêtes HTTP, soit en utilisant des valeurs connues et facilement identifiables dans leur dictionnaire [17].

Les deux derniers outils présentés étaient dédiés à l'exploitation de failles déjà découvertes, et bien que leur fonctionnement soit différent, leur détection reste également possible.

Bien que la détection d'outils soit précise, elle nécessite que les règles soient écrites pour chacun des cas d'utilisation possible, et qu'elles soient maintenues à jour avec l'évolution des logiciels.

J'ai donc voulu m'intéresser à la détection d'outils et d'attaques plus génériques.

17. Tous ces outils ayant à la base comme objectif la réalisation d'audits de sécurité, la présence de leur nom dans le *User-Agent* permet de distinguer facilement les alertes de sécurité remontées à cause de ces outils des alertes dûes à des attaques « réelles ».

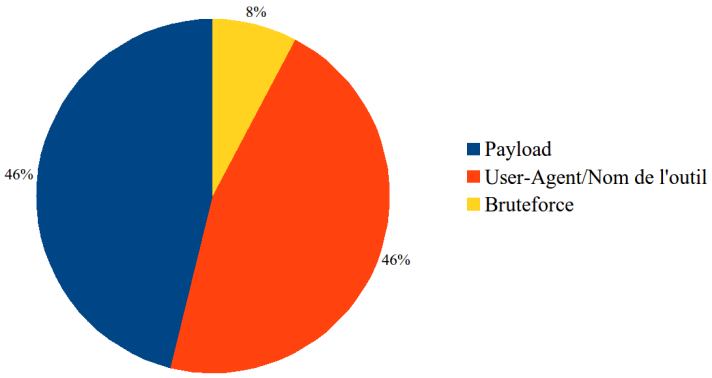


FIGURE 13 – Diagramme présentant des statistiques sur les parties des paquets utiles à la détection des outils d'audit.

3.2 Limites de la détection d'intrusions

Dans la partie précédente, j'ai dréssé la liste des outils que j'ai analysé et pour lesquels j'ai écrit des règles Snort permettant de les détecter.

J'ai également tenté d'écrire des règles de détection d'outils et d'attaques plus génériques, afin d'obtenir un meilleur niveau de détection sur les périmètres monitorés. Cependant, certains d'entre eux ne sont pas détectables, soit parce que leur fonctionnement est beaucoup trop générique pour que les règles soient pertinentes (le nombre de faux positifs serait trop important), soit parce que leur fonctionnement rendrait les règles écrites trop coûteuses en temps et/ou en ressources pour les équipements.

De façon similaire, j'ai pu remarquer que les règles de détection écrites pour les précédents outils sont facilement contournables, et que leur détection ne sera pas toujours réellement efficace.

Je vais dans cette partie vous présenter les raisons pour lesquels la détection d'intrusions n'est pas toujours efficace, en présentant certaines techniques de contournement d'IDS. Puis je dresserai la liste de plusieurs outils et attaques génériques que j'ai tenté de détecter, le niveau de détection que j'ai pu atteindre, ainsi que les raisons pour lesquels ce niveau n'a pas pu être atteint.

3.2.1 Contournement d'IDS

Les outils que j'ai présenté dans la partie 3.1 permettent de lancer des attaques ou d'effectuer des audits de sécurité sur des applications Web.

Comme je l'ai présenté dans la figure 13 [18], près de la moitié des outils sont détectés en utilisant soit une particularité dans le *User-Agent* utilisé, ou bien plus généralement en repérant leur nom dans une partie des échanges entre le client et le serveur Web.

Cependant, une détection basée uniquement sur cette donnée n'est pas totalement efficace.

18. Le détail de ces statistiques est donné en annexe B.

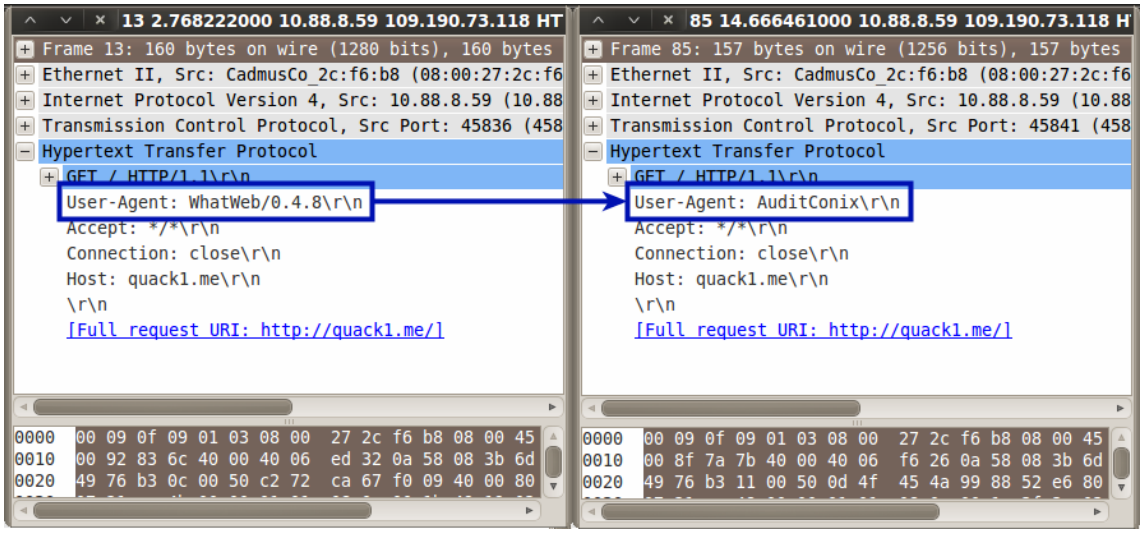


FIGURE 14 – Capture réseau d'une attaque lancée par l'outil WhatWeb, et capture de la même attaque lancée par l'outil modifié pour masquer son nom dans le *User-Agent*.

Dans la grande majorité des cas, l'outil sera utilisé tel quel par les attaquants, et la détection sera effectivement réussie.

En revanche, dans certains cas, ceux-ci modifieront les outils afin de les rendre plus discrets, et de ce fait, moins détectables.

La grande majorité des outils dont j'ai analysé le fonctionnement sont *Open-Source*, et peuvent donc être modifiés et réutilisés facilement. Le *User-Agent* n'est pas une part importante de l'attaque. Il est même totalement inutile, et ne sert qu'à annoncer au serveur les versions de l'OS [19], du navigateur et des plugins utilisés par le visiteur, ou encore l'outil utilisé pour accéder à la page Web.

Modifier cette valeur ne changera en aucun cas le déroulement de l'attaque, et peut donc être réalisé sans problème pour mieux masquer les outils utilisés.

La figure 14 présente une capture réseau d'une attaque lancée avec WhatWeb, puis la même attaque lancée par le même outil, après avoir modifié son *User-Agent*.

L'autre grande partie des outils pour lesquels j'ai écrit des règles de détection Snort sont détectés grâce à leur *payload*. Cette détection est plus efficace que la détection basée sur le nom ou le *User-Agent*, puisque c'est ce *payload* qui va conditionner la réussite ou non de l'attaque.

Si un attaquant modifie la charge qui est envoyée au serveur, il modifie le comportement du navigateur en réponse à cette charge et, de ce fait, modifie l'attaque lancée.

Les outils BEEF et XSSF ont été détectés en se basant sur le nom des fonctions utilisées. Ces noms de fonctions peuvent également être modifiés pour contourner les IDS. Comme le *User-Agent*, cette détection peut facilement être *bypassée* par un attaquant. Une détection plus efficace aurait pu se baser sur le contenu des fonctions envoyées au

19. Système d'Exploitation

client par ces deux outils, en se basant sur le fait que la modification de ce code-source entrainerait la modification du fonctionnement de l'outil.

Certaines règles de détection pouvant être facilement contournées, il est important de les auditer régulièrement, afin de s'apercevoir, au plus tôt, si elles sont vulnérables.

Dans la majorité des cas, une détection du payload envoyé sera plus efficace, plus précise, et moins facilement contournable qu'une détection basée uniquement sur le *User-Agent* ou sur le nom de l'outil utilisé qui serait présent dans les requêtes envoyées.

3.2.2 Attaques par « force brute »

Les premiers outils génériques que je vais présenter ici sont des outils qui utilisent des attaques par « brute-force ». Beaucoup des outils présentés dans la partie 3.1 utilisent également ce genre de méthode, c'est pourquoi je vais montrer quelles sont les différences entre ces outils et ceux qui ne sont pas détectables, et les raisons pour lesquelles cette détection est impossible.

J'ai étudié de nombreux outils au début de mon stage, et 4 utilisent ce genre de techniques.

1. Dirb – `http://dirb.sourceforge.net/`
2. WebSploit (module `dir_scanner`) – `https://sourceforge.net/projects/websploit/`
3. Uniscan (modules `Directory Checks`, `Files Checks` & `Dynamic Checks`) – `http://sourceforge.net/projects/uniscan/`
4. WebScarab – `https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project`

Dirb, le module `dir_scanner` de WebSploit ainsi que les modules `Directory Checks` et `Files Checks` d'Uniscan permettent de liste les fichiers et répertoires accessibles sur un site Web. Leurs dictionnaires de *payload*s contiennent des listes de fichiers et répertoires connus comme étant fréquemment utilisés par des applications Web, comme `/img/`, `/api/` ou `/wp-admin/` pour des blogs Wordpress [20] par exemple. Une partie des requêtes HTTP effectuées sur le serveur Web durant l'attaque est donnée dans la figure 15.

Ces outils utilisent des dictionnaires variés, et il n'est pas possible de pouvoir créer une règle sur ces payloads. Le seul point récurrent dans le fonctionnement de ces outils est l'envoi de requêtes HTTP `GET`.

Il est possible de détecter ces requêtes de cette façon, cependant énormément de faux-positifs perturberont la lecture des évènements, puisque la majorité des requêtes légitimes effectuées sur des sites Web se font en `GET`.

L'autre technique de détection consiste à ne lever d'alertes qu'à partir d'un nombre donné d'évènements reçus pour une règle définie. On peut de cette manière détecter qu'un brute-force est en cours si on reçoit plus de 120 requêtes `GET` par minute depuis la même

20. http ://wordpress.org/

1372	1.15972700	127.0.0.1	127.0.0.1	HTTP	264	GET /bankers/ HTTP/1.1
1379	1.17174400	127.0.0.1	127.0.0.1	HTTP	262	GET /banki/ HTTP/1.1
1396	1.19508900	127.0.0.1	127.0.0.1	HTTP	262	GET /banks/ HTTP/1.1
1400	1.19554500	127.0.0.1	127.0.0.1	HTTP	263	GET /banned/ HTTP/1.1
1406	1.20173000	127.0.0.1	127.0.0.1	HTTP	265	GET /bannerad/ HTTP/1.1
1420	1.21263000	127.0.0.1	127.0.0.1	HTTP	264	GET /banners/ HTTP/1.1
1427	1.21389300	127.0.0.1	127.0.0.1	HTTP	261	GET /bans/ HTTP/1.1
1431	1.21429300	127.0.0.1	127.0.0.1	HTTP	264	GET /bannery/ HTTP/1.1
1435	1.21467400	127.0.0.1	127.0.0.1	HTTP	263	GET /bantop/ HTTP/1.1
1451	1.21813600	127.0.0.1	127.0.0.1	HTTP	265	GET /banniere/ HTTP/1.1
1467	1.22626900	127.0.0.1	127.0.0.1	HTTP	265	GET /bar-code/ HTTP/1.1
1471	1.22676100	127.0.0.1	127.0.0.1	HTTP	264	GET /barcode/ HTTP/1.1
1481	1.22872900	127.0.0.1	127.0.0.1	HTTP	265	GET /barcodes/ HTTP/1.1
1491	1.23066500	127.0.0.1	127.0.0.1	HTTP	265	GET /baseline/ HTTP/1.1

FIGURE 15 – Capture réseau d'une attaque par force brute permettant de lister les répertoires accessibles sur un serveur Web.

adresse IP. Cependant, la règle peut toujours lever des faux-positifs, et le traitement de règles de ce type par Snort est plus coûteux qu'une règle « normale ».

En effet, pendant ces 60 secondes, Snort doit garder en mémoire les paquets reçus et vérifier si de nouveaux ne correspondent pas à cette règle, afin de lever ou non l'alerte.

3.2.3 Attaques Web génériques

Les attaques par force brute sont des attaques très générales, qui peuvent être lancées contre tout type de système. J'ai décrit dans la partie 3.1 des outils qui effectuaient des reconnaissances d'applications Web, mais également d'outils d'exploitation de vulnérabilités Web.

J'ai par exemple montré que l'on pouvait détecter des attaques de type *Injection SQL* (ou SQLi), *Local File Inclusion* (ou LFI) et *Cross Site Scripting* (ou XSS). Ces détections s'effectuent à partir de motifs précis tirés des attaques lancées.

Une attaque *SQLi* qui serait lancée en utilisant un *payload* différent de ceux qui sont pris en compte dans les règles de détection ne serait pas détectée. J'ai donc tenté de détecter ces 3 types d'attaques, SQLi, LFI et XSS de façon générique, ou au moins de façon plus générale que les règles de la partie précédente.

Afin d'écrire les règles les plus génériques possibles, j'ai étudié en détails le fonctionnement de ces 3 attaques.

Une attaque de type *Injection SQL* consiste à envoyer à l'application Web une valeur qui sera utilisée dans une requête SQL, et qui sera écrite de telle façon qu'une requête SQL différente de celle initialement écrite dans l'application soit exécutée. Une attaque par *Cross-Site Scripting*, ou XSS, permet de faire exécuter à un visiteur du site Web une portion de code externe à l'application. Enfin, les attaques LFI, ou *Local File Inclusion*, et RFI, ou *Remote File Inclusion*, permettent d'inclure dans les pages Web affichées des fichiers présents sur le serveur ou sur un site tiers. Typiquement, au cours d'une attaque par LFI, l'attaquant tente d'accéder au fichier `/etc/passwd`.

L'exploitation de ces attaques peut se faire de façons très différentes. Chaque attaquant exploite ce type de failles à sa manière, et écrire une règle générique qui permettrait de toutes les détecter est impossible.

J'ai donc préféré utiliser une approche plus générale que générique. La question n'était plus de détecter *toutes* les attaques SQLi, XSS et LFI possibles, mais d'en détecter le plus possible. Pour cela, j'ai dressé une liste des différents payloads qui revenaient le plus souvent dans les dictionnaires des différents outils, ainsi que ceux utilisés par les équipes d'audit de Conix Security.

Injection SQL Les attaques par injection SQL consistent à modifier les requêtes SQL exécutées sur le serveur de base de données. On retrouve donc majoritairement dans les payloads des *quotes* (`'`), ou encore des portions de code SQL. Généralement, Ces morceaux de requêtes sont du type `'X' = 'X'`, `'X' IS NOT NULL` ou `SELECT X FROM Y`. La liste complète est donnée dans l'annexe C.1.

À partir de la liste dressée, j'ai alors pu écrire la règle nécéssaire. Au lieu d'écrire plusieurs règles pour chaque motif identifié, j'ai choisi de n'en faire qu'une seule, contenant tous ces motifs factorisés au sein d'une expression régulière. Cette expression régulière tente de combiner la majorité des cas identifiés comme potentiellement illégitimes.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS SQLi.
    "; content:"GET"; http_method; content:"="; pcre:"/=.*([\('\"]?\w*[\)'\
    "]?\s*(((=|LIKE)\s*[\('\"]?\w*[\)'\"]?)|(IS\s*(NOT\s*|)NULL))|((=|LIKE|
    UNION(\s+ALL|)\s+|)SELECT.+(FROM|)))/i"; sid:14;)
```

Listing 21 – Règle Snort de détection générique d'attaques de type *SQLi*.

L'expression régulière détaillée est donnée en figure 22, page 100.

Cross-Site Scripting Une attaque de type *Cross-Site Scripting*, ou XSS est une attaque qui vise à injecter du code dans un site Web via l'exploitation d'une vulnérabilité.

Ces attaques consistent typiquement à injecter du code source permettant d'effectuer des actions sur les navigateurs visitant la page, pour ainsi récupérer des informations d'authentification ou des informations personnelles sur les utilisateurs.

Une attaque XSS est, généralement, effectuée en injectant du code Javascript. Ce code consiste le plus souvent en un `alert()` afin d'afficher une valeur à l'écran pour vérifier que l'attaque est fonctionnelle, ou en l'inclusion de code Javascript qui sera exécuté via la balise `javascript:`.

J'ai donc utilisé la même méthode de détection que pour les injections SQL, c'est à dire une expression régulière qui correspond à tous les *payload*s d'attaques XSS retenus. J'en donne la liste en annexe C.2, page 99.

La première règle écrite se base sur le fait que pour inclure du code Javscript qui sera éxécuté, les outils automatiques ainsi que les attaquants utilisent beaucoup les balises

HTML [21] `<script>` et `<img>`. En effet, ces deux balises permettent de charger dynamiquement des ressources ou du code source.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS XSS
    (1)"; content:"GET"; http_method; content:"=";pcre:"/<(script|img)(\s
    |\/)*(>|src=[\"' ]?)(javascript(:alert)?)?/i"; sid:15;)
```

Listing 22 – Règle Snort de détection générique d'attaques de type *XSS*.

En généralisant encore plus la règle précédemment écrite, j'ai également remarqué que la plupart des outils utilisaient un *payload* extrèmement simple pour vérifier la présence d'une attaque XSS. Ce payload contient la fonction Javascript `alert()`, qui affiche une valeur à l'écran.

J'ai donc écrit une deuxième expression régulière, basée cette fois-ci sur cette chaîne.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS XSS
    (2)";content:"GET";http_method;content:"=";pcre:"/(javascript:?)?alert
    \(/i"; sid:16;)
```

Listing 23 – Règle Snort de détection générique d'attaques de type *XSS*. *Version raccourcie*

Local File Inclusion Enfin, la troisième attaque Web générique que j'ai tenté de détecter est la LFI [22], ou *Local File Inclusion*. En exploitant une vulnérabilité, il est possible d'inclure dans une page Web un fichier présent sur le serveur distant. Typiquement, un attaquant tentera d'accéder au fichier `/etc/passwd` sur un système Unix, afin d'obtenir la liste des noms d'utilisateurs du système, et possiblement leurs mots de passe.

L'exploitation de ce type d'attaque se fait en envoyant au serveur, dans un paramètre ou toute autre variable, le nom du fichier auquel on souhaite accéder. La détection se base donc sur la présence, dans la requête HTTP, du nom de fichier `/etc/passwd`, ou de noms de fichiers spécifiques au système d'exploitation Microsoft Windows, comme `C:\` ou `cmd.exe`. Afin de maximiser les chances de les détecter, j'ai pris en compte dans l'expression régulière le fait que les caractères spéciaux, comme le slash (/) ou l'antislash (\), peuvent être encodés différement dans l'URI [23].

L'expression régulière associée s'en trouve donc allongée.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS LFI";
     content:"GET"; nocase; pcre:"/((((\.|%(25|)(2e|c0%(25|)(ae|2e)))
    {0,3}(\\|\/|%(25|)((5c|2f)|c0%(25|)(af|2f)|c1%(25|)9c))+)+)(etc
    (\\|\/|%(25|)((5c|2f)|c0%(25|)(af|2f)|c1%(25|)9c))passwd|C:\\|cmd\.exe)
    /i"; sid:17;)
```

21. Le HTML, ou *Hypertext Markup Language*, est le format de données utilisé pour représenter les pages Web. Son format permet de strucutrer sémantiquement et de mettre en forme le contenu des pages affichées.

22. Une LFI, ou *Local File Inclusion*, est une attaque Web qui permet à un attaquant d'inclure dans la page Web vulnérable un fichier présent sur le serveur Web. Typiquement, ce fichier est le fichier `/etc/passwd` sur les systèmes UNIX.

23. Par exemple, les chaînes `%2E%2E%2Fetc%2Fpasswd` et `%252E%25C0%25AE/etc%252Fpasswd` correspondent à la chaîne `/etc/passwd`

Listing 24 – Règle Snort de détection générique d'attaques de type *LFI*.

Je viens de montrer avec ces 4 exemples de règles Snort qu'il était possible de détecter, de façon générique, trois types d'attaques Web. Cependant, bien que cette détection soit en effet théoriquement possible, en pratique, ces règles peuvent difficilement être mises en place.

En premier lieu, l'utilisation d'expressions régulières aussi longues dans Snort est très coûteux en temps et en ressources, et va diminuer les performances des sondes. De la même façon, utiliser de nombreuses règles qui *matcheront* chacune sur un motif précis créera trop de règles sur les sondes Snort ce qui réduira également ses performances.

Un attaquant peut envoyer, pour exploiter une vulnérabilité, des *payload*s qui n'auront pas été pris en compte dans les règles et qui ne seront donc pas détectés. Par exemple, la règle de détection des attaques XSS donnée en page 48 se base sur l'utilisation des balises HTML `<script>` et `<img>`. Cependant, avec l'arrivée des navigateurs compatibles avec la norme HTML5 [24] et ses nouvelles balises multimédia, il est également possible d'exploiter des vulnérabilités XSS en utilisant par exemple la balise `<video>` sans être détecté par les règles existantes.

L'utilisation d'une fonction Javascript différente de `alert()` peut également permettre de contourner ces règles.

Enfin, comme je l'ai montré avec la règle de détection des *Local File Inclusion*, un attaquant peut également *bypasser* les règles d'un IDS en encodant certains caractères de l'URI. Encore une fois, si la règle ne prend pas en compte ces cas d'attaques, elle sera inefficace. Et en contrepartie, une règle trop générique, ou une expression régulière trop grande vont respectivement créer trop de faux positifs et consommer trop de ressources.

3.2.4 Le cas *nmap*

Enfin, un autre outil très utilisé lors d'audits ou d'attaques est *nmap* [25], un scanneur *Open-Source* qui permet de lister les ports réseau [26] ouverts sur une machine. Très utile pendant les phases de reconnaisance, *nmap* dresse la liste des services ouverts sur machine et accessible sur le réseau et permet ainsi de préparer les attaques et audits.

24. `http://fr.wikipedia.org/wiki/HTML5`
25. `http://nmap.org/`
26. Sur la couche 4, ou transport, du modèle TCP/IP, un port permet de distinguer les différents interlocuteurs de la machine. Chaque programme informatique écoute ou envoi des données sur un port différent de la machine. Sur un serveur, les ports les plus connus sont le 80 (HTTP pour un serveur Web), 22 (SSH), ou encore le 25 (serveur SMTP (`http://fr.wikipedia.org/wiki/Smtp`) pour un serveur de mails).

Pour déterminer si les ports sont ouverts, fermés, ou filtrés [27], *nmap* envoie des paquets TCP [28] afin d'établir une connexion avec la machine distante. Ces paquets TCP sont créés avec des valeurs choisies spécifiquement pour induire des réponses spécifiques sur le serveur contacté, et ainsi déterminer avec plus de précision le système d'exploitation ou le logiciel envoyant la réponse.

Les règles Snort précédentes ne détectent les comportements anormaux qu'à partir des informations générales disponibles sur les couches 3 (adresses IP), 4 (ports TCP ou UDP [29]), ou 5 (données de l'application) du modèle TCP/IP [30]. Les numéros de ports obtenus par Snort dans les entêtes TCP sont utilisés dans la plupart des règles.

Nmap défini certains drapeaux [31] et certaines options des paquets TCP envoyés. Snort intègre la recherche dans les *flags* TCP, mais pas dans les options. En effet, cette recherche est couteuse en ressources, et les règles écrites de cette façon ne concernent qu'une petite partie des données transitant sur un réseau.

C'est pour cette raison qu'il est impossible de détecter des scans *nmap* de façon sûre avec un HIDS tel que Snort. Les performances du système de détection d'intrusions seraient trop altérées pour assurer un niveau de détection suffisant.

Cependant, il serait toujours possible d'effectuer cette détection au moyen d'équipements réseau spécialisés, comme des routeurs ou des *switchs* qui peuvent manipuler et investiguer très rapidement dans des paquets réseau.

En conclusion

Comme je l'ai montré avec ces quatres exemples, il est possible d'écrire des règles de détection d'attaques génériques, comme les attaques Web ou des attaques par force brute.

Cependant, j'ai également montré que ces règles étaient soit trop larges, et généraient trop de faux-positifs, soit nécessitaient l'utilisation d'expressions régulières, ce qui diminuait les performances des systèmes de détection d'intrusions.

De la même façon, détecter des outils de bas niveau, comme *nmap* est difficilement réalisable par un NIDS tel que Snort, et il est préférable d'utiliser pour cela d'autres

27.

28. Le protocole TCP, ou *Transmission Control Protocol*, est un protocole de communication entre deux machines connectées, utilisant la couche 4 du modèle TCP/IP. Les données des applications utilisant le protocole TCP sont découpées en paquets et envoyées à la suite sur le réseau. Les spécifités de ce protocole assurent que les flux seront reconstruits dans le même ordre que les flux de départ. Le protocole TCP est défini dans la RFC 793 (`http://www.rfc-editor.org/rfc/rfc793.txt`).

29. Le protocole UDP, ou *User Datagram Protocol*, est un protocole de communication semblable à TCP, à la différence qu'UDP communique en mode « non-connecté ». L'intégrité et l'ordre d'arrivée des flux n'est donc pas garanti. Ce protocole est défini dans la RFC 768 (`http://www.rfc-editor.org/rfc/rfc768.txt`).

30. Le modèle TCP/IP définit le standard de communication réseau des systèmes informatiques. Il donne les fonctionnalités nécéssaires à la communication de systèmes hétérogènes sur un même réseau, et notamment une architecture en couches qui permet à des protocoles de haut niveau de s'appuyer sur les protocoles sous-jacent pour assurer les étapes d'établissement des connexions et de gestion du réseau. Il est inspiré du modèle OSI. Les différentes couches du modèle TCP/IP sont données en figure 26, page 104.

31. `http://fr.wikipedia.org/wiki/Transmission_Control_Protocol#Structure_d.27un_segment_TCP`

outils, plus spécialisés dans la manipulation de flux réseaux.

Enfin, l'écriture de règles de systèmes de détection d'intrusions est un exercice complexe, qui nécessite de prendre en compte les techniques de contournement qui sont utilisables par les attaquants pour passer outre les IDS.

Nous avons notamment soumis ces règles, à des fins de tests, aux équipes d'audits de Conix Security. Une application Web vulnérable a été installée sur un serveur, et les règles de détection écrites précédemment ont été mises en place sur des IDS présents sur le système.

J'ai ainsi pu remarquer que même si les règles sont précises, ou parfois très généralistes, il est impossible d'être exhaustif et de détecter tous les cas d'attaques. Notamment, de nombreuses documentations et articles sont disponibles en ligne sur ce sujet. Voir par exemple les liens [Ale12], [Che11], [Dah10], [OWA12].

J'ai ainsi pu observer que certaines limites de la détection d'intrusions basée sur des évènements réseau sont facilement atteintes dès que l'on souhaite détecter des attaques plus évoluées que de simples outils. Plus généralement, un IDS réseau ne pourra détecter que les attaques pour lesquelles des règles auront été écrites.

De ce fait, les règles des systèmes de détection d'intrusions peuvent être facilement contournées, et il faut donc réflechir à de nouvelles méthodes de détection basées, soit sur une autre donnée que l'empreinte réseau, soit sur un enchaînement d'évènements qui permettent de confirmer ou infirmer les intrusions détectées.

3.3 Scénarios d'attaques évolués – Corrélation N/HIDS et SIEM

Les systèmes de détection d'intrusions sont des systèmes importants dans les infrastructures informatiques. Ils permettent de connaître, en temps réel, l'état de sécurité du SI. J'ai montré, dans la partie 2 de mon rapport, qu'il fallait, pour que les informations soient pertinentes, que les règles des IDS soient correctes.

Dans la partie 3.1, j'ai dressé une liste des outils que j'ai analysé et réussi à détecter uniquement au moyen d'un système de détection d'intrusions basé sur le réseau. Cependant, j'ai également montré que certains outils et certaines attaques sont impossibles à détecter, tout au moins avec un seul outil comme Snort.

Un deuxième inconvénient des systèmes de détection d'intrusions réseaux, qui est également un avantage, est que les informations apportées sont extrêmement précises et spéficiques au réseau. Sur une attaque lancée contre un système d'information depuis Internet, les données des NIDS, comme les adresses IP ou le contenu des paquets, sont utiles puisqu'ils permettent de savoir qui lance l'attaque, contre qui, et quel est le *payload* utilisé. Cependant, ils n'apportent aucune information sur la réussite ou non de l'attaque

À l'inverse, si l'on cherche à détecter un malware provenant de l'intérieur du réseau, un NIDS ne permettra pas de savoir quels sont les postes utilisateurs impactés, ni le nom du malware, etc...

Afin de palier à ces lacunes, il est possible de mettre en place des HIDS, des systèmes de détection d'intrusions qui se basent sur des évènements obtenus depuis les systèmes d'exploitation des machines. Comme je le présentai dans la partie 2.3.2 et 2.3.3, ces outils sont précis et permettent, eux aussi, d'obtenir des informations fiables, dès lors que leurs règles sont suffisament bien écrites.

Cependant, les HIDS possèdent les mêmes inconvénients que les NIDS. Les informations remontées sont spécifiques à la plateforme source.

C'est pour ces raisons qu'il est intéressant d'utiliser un SIEM, qui permet de corréler les évènements de sécurité provenant de plusieurs IDS, pour ainsi obtenir des alertes plus pertinentes et concrètes, sur des attaques plus génériques, tout en diminuant la quantité de faux-positifs.

J'ai donc pu détecter, avec plus de certitude, des attaques Web, comme je le montrerais dans cette partie, mais j'ai aussi choisi, au lieu de ne détecter que des outils, d'imaginer des scénarios d'attaques, plus complets et plus vastes, répartis sur plusieurs parties du SI.

Je vais donc présenter comment on peut détecter avec plus de certitude, des attaques ou des actions anormales du point de vue de la sécurité d'un système informatique.

Je débuterais cette partie en présentant comment il est possible de détecter une attaque web générique, comme une LFI, au moyen de plusieurs IDS et d'un SIEM.

Je présenterais ensuite d'autres scénarios plus avancés, comme la détection généralisée d'une attaque contre un SI, qui se basera sur un *scan nmap* puis un attaque sur SSH, ou un *monitoring* de l'authentification sur les postes de travail.

3.3.1 Détection avancée d'une attaque de type LFI

J'ai déjà présenté les attaques de type LFI dans la partie 3.2.3. J'ai également donné des pistes et des règles Snort qui permettent de détecter ce genre d'attaques. Cependant, j'ai également montré que cette règle pouvait être relativement facilement contournée, et que l'utilisation d'expressions régulières est très coûteux en ressources.

J'ai donc cherché une nouvelle méthode de détection, tirant profit des différents IDS à ma disposition.

Cette solution se compose donc de plusieurs parties. Une détection d'un motif réseau basique, puis la détection, soit d'un accès à ce fichier sur le serveur web, soit la détection du contenu du fichier sur le réseau.

La détection par Snort en amont de l'attaque est simple, et a déjà été détaillée en page 48. Cependant, la règle a été considérablement réduite, pour qu'elle ne *matche* plus que sur le nom du fichier de mots de passe des systèmes Unix : `/etc/passwd`.

Snort possède la particularité de confondre les caractères encodés dans le format des URI, et en texte « clair ». Par exemple, si la règle Snort contient la clause `content:"/etc/passwd"`, elle lèvera des évènements pour les paquets contenant `/etc%2Fpasswd`, ou `/%65tc/%70asswd`. Le problème de l'encodage des caractères est donc résolu de base par Snort par l'utilisation de la clause `content`.

La nouvelle règle de détection d'une attaque de type LFI sur une application Web pourrait donc être la suivante :

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS LFI";
    content:"GET"; nocase; http_method; content:"/etc/passwd"; http_uri;
   sid:18;)
```

Listing 25 – Règle Snort de détection basique d'une attaque de type *LFI*.

Encore une fois, cette règle permet seulement de savoir qu'un attaquant est en train de lancer une attaque de type *Local File Inclusion* sur une application Web. Pour savoir si l'attaque a réellement réussi, il faut employer d'autres IDS, et en particulier des systèmes de détection d'intrusions basés sur des évènements du système d'exploitation hôte.

Cette détection est réalisée au moyen de deux HIDS, OSSEC et Audit, présentés respectivement en pages 17 et 20. Ces outils vont nous permettre de vérifier si l'utilisateur `www-data`, qui lance les applications Web, accède au fichier `/etc/passwd`.

La règle de configuration d'Audit est donc identique à la règle définie dans la partie 2.3.3 de ce rapport.

```
-p war -w /etc/passwd -F uid 33
```

Listing 26 – Règle Audit de détection des accès en lecture ou écriture au fichier `/etc/passwd` par l'utilisateur `www-data` (uid:33).

De cette manière, on retrouve dans la console du SIEM, ou dans les logs bruts des IDS, deux évènements. Le premier, émis par Snort, et le second par le couple Audit/OSSEC qui détecte l'accès au fichier. L'objectif de la « double détection » étant de diminuer le nombre de faux positifs, on cherche également à n'avoir qu'une seule alerte de sécurité par intrusion ou attaque.

Il est donc possible d'utiliser les directives de corrélation d'un SIEM comme OSSIM [32], qui ont deux avantages.

En premier lieu, ils permettent de corréler les évènements de plusieurs IDS, ou plus généralement de différentes sources de données, pour ne lever qu'une alerte de sécurité, plus précise.

Ensuite, les directives de corrélation d'un SIEM peuvent ajuster le niveau de criticité des alertes levées en fonction des évènements qui apparaîssent au cours du temps.

La directive de corrélation écrite pour les attaques LFI se décompose en 3 niveaux.

Au moment de la détection par Snort du payload de l'attaque, le SIEM génère une alerte de niveau 4 [33].

À ce moment là, OSSIM se place en attente de l'évènement provenant d'OSSEC. S'il obtient, dans une période de temps définie, l'évènement correspondant à l'accès au fichier `/etc/passwd`, le SIEM va remonter la criticité à 8, car il obtient la confirmation que l'attaque est réellement en cours.

Si aucun évènement Audit/OSSEC n'est levé, alors la criticité est placée à 5, car une attaque a été lancée contre un site Web monitoré, sans toutefois être effective.

Un diagramme représentant l'évolution du niveau de criticité en fonction des évènements reçus est donné en figure 16.

3.3.2 Détection avancée d'attaques lancées par des outils automatiques

La corrélation d'évènements de sécurité peut être utile, comme je viens de le montrer, pour détecter une attaque Web générique.

Il est également possible d'utiliser les SIEM pour détecter avec plus de précision les attaques lancées par des outils d'audit.

32. OSSIM, ou *Open Source Security Information Management* est une solution SIEM Open-Source développée par la société Alienvault, et déployée chez ses clients par Conix Sécurity. `http://communities.alienvault.com/`.

33. Une alerte de niveau 4 est considérée comme étant de criticité moyenne. En dessous de 4, la criticité est faible, au dessus de 7 elle est élevée.

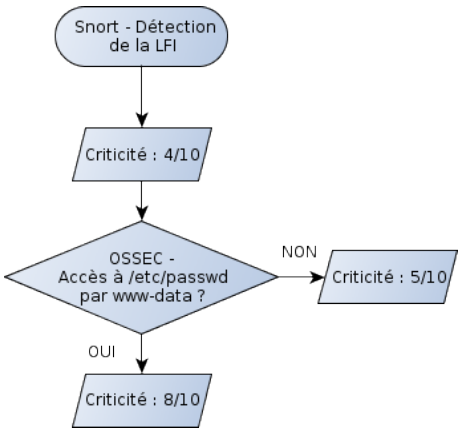


FIGURE 16 – Schéma d'évolution du niveau de criticité d'une attaque Web de type *LFI*.

Un outil très utilisé est le *proxy*[34] Burp[35]. Il est capable d'intercepter les requêtes envoyées aux serveurs Web audités, de les modifier, et surtout de réutiliser ces requêtes comme des « modèles », afin de lancer des *scans* de vulnérabilités sur ces applications, en utilisant des requêtes paraissant légitimes.

Burp étant un outil très utilisé par les auditeurs et les attaquants, je me suis également penché sur son fonctionnement afin de tenter de le détecter.

Le module *scanner* de Burp est un scanner de vulnérabilités comme WebSecurify ou XSSer. Il tente d'injecter sur l'application attaquée de nombreux *payload*s, pour déterminer si un ou plusieurs vulnérabilités sont présentes.

La première piste de réflexion concernant la détection avait été de rechercher un motif connu dans les paquets reçus, comme le *User-Agent* par exemple. Cependant, Burp utilise comme base à toutes ses requêtes une des requêtes effectuées par le navigateur de l'auditeur sur l'application auditée. Il est donc impossible d'utiliser cette méthode.

La deuxième technique est de rechercher des *payload*s spécifiques à Burp dans les requêtes réalisées sur le site Web. Ici aussi, ce type de détection est complexe à mettre en place, du fait des spécificités dans le fonctionnement de Burp. Le scanner est paramétrable en amont de l'attaque, afin de spécifier comment Burp doit tester les vulnérabilités, et quels sont les *payload*s qu'il doit utiliser.

J'ai donc choisi d'utiliser les performances du moteur de corrélation d'OSSIM afin de détecter avec efficacité le scanner de Burp.

Il est possible d'écrire une règle Snort pour un payload donné de Burp. En revanche, il est possible que ce *payload* ne soit pas utilisé lors du *scan* des vulnérabilités. Dans ce

34. Un *proxy* est un équipement logiciel ou matériel qui est placé, sur un réseau, entre les ressources internes et l'Internet. Il permet de filtrer et/ou modifier le contenu auquel les utilisateurs internes du réseau accède.

35. `http://portswigger.net/burp/`

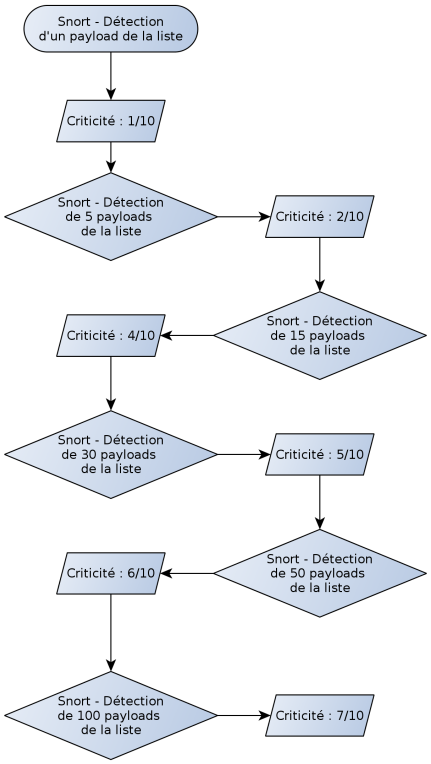


FIGURE 17 – Schéma d'évolution du niveau de criticité d'une attaque lancée avec le *scanner* de Burp.

dernier cas, la détection par Snort sera inefficace. Sinon, nous recevrons un évènement concernant cette tentative d'attaque.

Il est donc possible de faire de même pour un certain nombre de *payload*s utilisés par Burp. De cette façon, à chaque attaque utilisant le *scanner* de Burp, un certain nombre d'alertes seront remontées par Snort. Plus ce nombre est important, plus la probabilité que l'attaque soit réellement lancée avec Burp sera grande.

C'est ce principe que j'ai choisi de mettre en pratique au cours de mon stage. J'ai établi une liste des *payload*s les plus fréquemment utilisés par le *scanner* de Burp. Cette liste est donnée en annexe, page 103.

Chaque motif de cette liste a fait l'objet de l'écriture d'une règle basique dans Snort.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS Burp
    Scanner Payload 1"; content:"alert(document.cookie)"; nocase; sid:19;)
```

Listing 27 – Règle Snort de détection basique d'une payload utilisé par le *scanner* de Burp.

J'ai ensuite écrit une directive de corrélation dans OSSIM, qui se charge de mettre à jour la criticité de l'alerte levée au fur et à mesure de la réception de ces évènements par Snort, qui est présentée dans la figure 17.

De la même façon, la détection des outils présentés dans la partie 3.1 ou des attaques Web génériques présentées en partie 3.2.3 peut être améliorée en utilisant cette méthode.

Dans le cas des outils, j'ai extrait une liste des *payload*s qui étaient les plus utilisés au cours de l'utilisation de ces logiciels. J'ai ensuite écrit les règles NIDS et la directive de corrélation nécéssaires à la mise en place de ce type de détection.

Concernant les attaques Web génériques, j'avais massivement utilisé des expressions régulières pour détecter une grande partie des *payload*s utilisés lors de ce type d'attaque. J'ai effectué l'étape inverse, c'est à dire que j'ai utilisé la liste de ces motifs que j'avais dressée auparavant (et que j'ai donné en annexe, page 98), puis j'ai de la même manière que précédement, écrit les règles Snort et les directives de corrélation adéquates.

3.3.3 Détection d'une attaque « générique »

Une attaque lancée contre un SI se compose généralement de deux phases. Une phase de reconnaissance, pendant laquelle l'attaquant tente d'obtenir le plus d'informations possibles sur sa cible, puis la phase d'attaque à proprement parler.

Un exemple d'attaque peut donc se constituer d'un *scan nmap* sur une (ou des) machine(s) du SI, afin de déterminer les différents services ouverts sur celle-ci, puis de tentatives d'intrusion sur le service SSH (par un bruteforce pour trouver le nom d'utilisateur, puis l'exploitation d'une faille connue par exemple).

J'ai ici choisi de présenter des attaques sur SSH, mais le fonctionnement de la détection est très simplement adaptable à tout autre service Web.

La première partie de l'attaque est un scan *nmap*. Comme je l'expliquais en page 49, il est complexe de détecter cet outil avec un NIDS seul, tel que Snort. Cependant, dans le cadre d'une détection plus avancée, mettant en place un SIEM, il est possible de mettre en place une règle Snort générique, qui se base sur le nombre de connexions réalisées en une courte période de temps, et la présence de certains *flags* dans les paquets TCP.

Un exemple de règle Snort de détection d'un scan `SYN` est donné dans le listing 28. Elle ne lève une alerte qu'après avoir reçu 150 paquets SYN, en 20 secondes, provenant de la même IP source.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT TOOLS nmap SYN scan";
    flags:S; classtype:attempted-recon; threshold: type both, track by_src,
     count 150, seconds 20; sid:20;)
```

Listing 28 – Règle Snort de détection d'un scan *SYN* effectué par *nmap*.

Après avec effectué une reconnaissance de la cible, l'attaquant peut tenter d'exploiter une vulnérabilité sur le logiciel utilisé par le serveur SSH du serveur. La détection de cette exploitation se fait au moyen d'une règle Snort spécifique à cette vulnérabilité, qui sera de la même forme que les règles présentées dans la partie 3.1, consacrée à la détection d'outils d'audit Web.

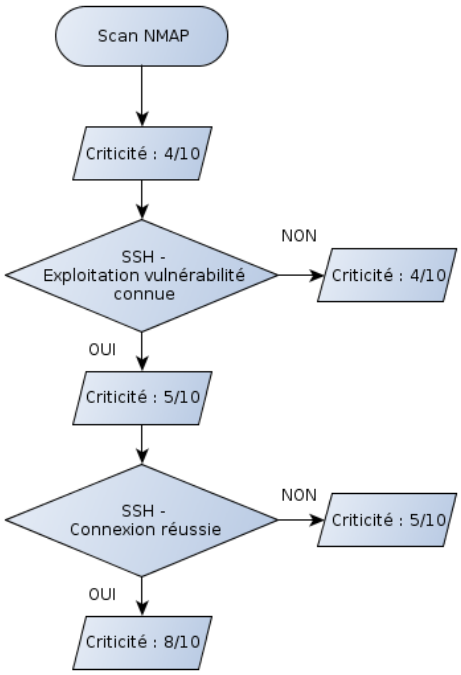


FIGURE 18 – Schéma d'évolution du niveau de criticité d'une attaque « generique ».

Avec les deux premières règles, le SIEM reçoit des évènements de sécurité pour l'étape de reconnaissance (le scan), puis pour la tentative de d'exploitation d'une faille sur un service présent sur le système. Il reste à déterminer si l'attaque a réellement réussi ou non.

Cette étape est réalisée grâce aux journaux d'évènements [36] du service SSH. Ils sont directement intégrés au SIEM OSSIM, et aucune configuration n'est donc nécéssaire.

La dernière étape avant la mise en place des scénarios dans le SIEM est de fixer les enchaînements et les différentes criticités des évènements.

Au moment de la détection, la criticité est définie à un niveau 4. Si l'exploitation d'une faille sur un service est détectée par Snort (ici, l'exploitation d'une faille sur le serveur SSH), la criticité est augmentée, et passe à un niveau 5. Si l'exploit SSH n'est pas détecté, la criticité reste à 4.

Enfin, si les logs du serveur SSH indiquent qu'une connexion est réussie suite à l'exploitation de la faille, le niveau de dangerosité devient critique, puisque l'attaquant pourrait avoir obtenu les informations nécéssaires pour se connecter au serveur. Le niveau de criticité passe donc à 8.

Un diagramme représentant l'évolution du niveau de criticité en fonction des évènements reçus est donné en figure 18.

36. ou *logs*

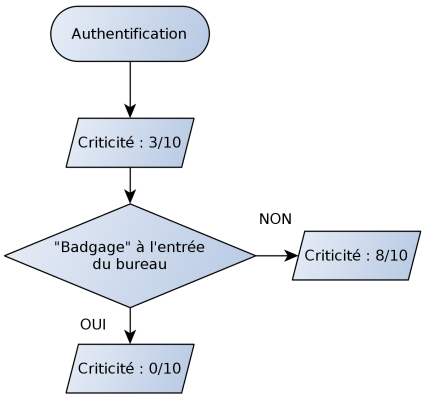


FIGURE 19 – Schéma d'évolution du niveau de criticité des évènements lors de la surveillance des authentifications.

3.3.4 Surveillance avancée de l'authentification

Enfin, un troisième scénario de *monitoring* a été imaginé. Celui-ci est différent des deux scénarios précédents puisqu'il ne se base pas sur des évènements provenant de serveurs, mais sur des informations extraites des postes utilisateurs.

En effet, la gestion des authentifications est une tâche importante de la sécurité des systèmes d'information. Elle doit en effet garantir que seules les personnes autorisées peuvent se connecter aux machines, et surtout que seules ces personnes utilisent les comptes qui leur sont associés. Il est en effet possible que grâce à des malwares, des attaquants puissent utiliser les postes et les comptes utilisateurs depuis l'extérieur du réseau, par exemple.

Il est donc possible d'utiliser un HIDS, comme OSSEC, pour recevoir des évènements dans le SIEM à chaque connexion réussie d'un utilisateur.

À ce moment de la détection, la criticité est évaluée à 3. En effet, l'information seule de connexion n'est pas en soit critique au niveau de la sécurité.

Cependant, on peut vérifier, de façon « relativement » sûre, si l'utilisateur est présent ou non dans les bureaux au moment où il se connecte, en se basant sur les informations remontées par les lecteurs de badge disposés à l'entrée de ceux-ci.

Si l'utilisateur n'a pas *badgé*, et qu'il est pourtant connecté, on peut raisonnablement penser qu'une intrusion a été réussié, ou qu'un malware a infecté un ou plusieurs postes utilisateurs, et qu'il faut alors alerter les administrateurs en remontant une alerte de sécurité de niveau 8.

En revanche, si l'utilisateur a prouvé sa présence dans le bureau avec son badge, la criticité de l'alerte est rabaissé à 0.

Un diagramme représentant l'évolution du niveau de criticité en fonction des évènements reçus est donné en figure 19.

3.4 La corrélation d'évènements : solution miracle ?

Dans la partie 3.1 de ce rapport, j'ai montré comment j'ai analysé des outils permettant de lancer automatiquement des attaques sur une application Web, puis j'ai présenté les différentes façon de les détecter au moyen de systèmes de détection d'intrusions, notamment d'un NIDS.

Ensuite, j'ai dressé une liste [37] des limites de cette détection, en présentant comment il était possible pour un attaquant de lancer des attaques qui ne seraient pas détectées par les règles précédemment écrites. J'ai également listé quelques outils et attaques que j'ai analysé et pour lesquels je n'ai pas pu écrire de règles de détection efficaces.

Enfin, j'ai montré comment on pouvait dépasser certains limites de la détection d'intrusions en utilisant conjointement plusieurs IDS, pour détecter de façon plus précise les attaques.

En conclusion de cette partie, je vais tenter de répondre à la question suivante :

La corrélation d'évènements et les SIEM sont-ils la solution miracle pour détecter toutes les attaques ?

Les SIEM permettent, comme je l'ai précisé plus haut dans ce rapport, de corréler des évènements provenant de nombreuses sources. Je n'ai présenté que des directives qui aggrégaient des *logs* issus de systèmes de détection d'intrusions, ou d'un lecteur de badge [38].

Les SIEM, et notamment OSSIM, sont également capables de recevoir et manipuler des évènements provenant d'équipements réseaux, comme des commutateurs [39] ou des routeurs, provenant d'équipements de sécurité, comme des *proxy*s ou des pare-feux, ou encore provenant directement de certains services installés sur des machines, comme SSH, des *logs* d'authentification Windows, ou des *logs* de serveurs Web comme Apache par exemple.

Toutes ces sources permettent de surveiller et lever des alertes de sécurité précises, puisque le SIEM a accès à des informations provenant de plusieurs points du SI.

De plus, la multiplication des sources et des évènements permettent d'écrire des directives de corrélation plus abouties, et donc de remonter des alertes plus précises et plus pertinentes, en se basant sur une base plus large et plus solide.

Malgré tout, les SIEM ne résolvent pas tous les problèmes.

Pour qu'ils soient efficaces et que leurs alertes soient pertinentes, il est indispensable que les NIDS et HIDS soient correctement configurés, et que leurs règles de détection soient maintenues à jour au cours du temps. Si les IDS sur lesquels repose le SIEM ne remontent aucun évènement, ou si la quantité de faux-positifs est trop importante, les alertes émises par le SIEM ne seront pas plus pertinentes que les simples évènements des IDS.

37. non exhaustive
38. Partie 3.3.4, page 59, *Surveillance avancée de l'authentification.*
39. ou *switchs*

NIDS		HIDS		SIEM	
Monitoring du trafic réseau	✓	Accès aux fichiers	✓	Corrélation des évènements	✓
Contexte	✗	Intégrité des fichiers et dossiers	✓	Reporting	✓
Détection précise des *payloads*	✓	Processus séparés	✓	Contexte	✓✗
Utilisation possible de *payloads* génériques	✓✗	Accès réseau des processus	✗	Faux-positifs/négatifs	✓✗
Utilisateur source de l'évènement	✗	Édition de paramètres systèmes	✓	Temps de maintenance/gestion	✓✗
Logiciels utilisés	✗	Utilisateurs	✓	Attaques multi-niveaux	✓
Adresses IP	✓				

FIGURE 20 – Comparatif détaillé des fonctionnalités des NIDS, HIDS et SIEM.

De plus, la mise en place et le maintien en conditions opérationnelles d'un SIEM demande, approximativement, deux fois plus de ressources (en termes humains et temporels), puisqu'il faut maintenir le jeu de règles des IDS et les directives de corrélation du SIEM.

En conclusion, et suite aux nombreuses expérimentations et manipulations que j'ai pu effectuer au cours de mon stage, je pense que l'utilisation d'un SIEM au sein d'une architecture de détection d'intrusions facilite grandement la gestion des évènements de sécurité.

Les SIEM possèdent, en plus du moteur de corrélation, des fonctionnalités permettant de filtrer et limiter les évènements reçus en fonction des investigations effectuées, et possèdent également des fonctions de *reporting* et de navigation au sein des alertes de sécurité qui rendent leur traitement plus aisé.

Enfin, il faut garder à l'esprit que pour qu'un SIEM soit efficace et produise des alertes pertinentes, il faut que les IDS sur lesquels il se base soient correctement maintenus, que leurs règles soient à jour, et que les directives de corrélation doivent également être évaluées régulièrement.

4

SECURITY OPERATIONS CENTER : LA DÉTECTION D'INTRUSIONS EN PRATIQUE

Une grande partie de mon stage a été consacrée à la mise en place de scénarios d'attaques, à l'analyse de ceux-ci, et à l'écriture de règles IDS permettant de les détecter.

Le reste du temps a été consacré à la participation aux activités du *SOC* de Conix Security.

Cette activité professionnelle et professionnalisante représentant une part non négligeable de mon stage, je vais la présenter ici, en détaillant dans un premier temps ce qu'est un *SOC*, puis en présentant plus précisément les activités auxquelles j'ai pris part.

4.1 *Security Operations Center*

Un SOC, ou *Security Operations Center* est une équipe de la DSI qui est en charge de la gestion et du maintien de la sécurité du système d'information.

De façon plus générale, un SOC désigne (définition tirée de la page anglophone de Wikipédia sur les SOC [Wik10]) :

> « les personnes, processus et technologies qui permettent d'obtenir un état du niveau de sécurité à travers la détection, le confinement et la résolution des menaces informatiques. Un SOC gère les incidents de sécurité pour une entreprise, assurant qu'ils sont convenablement identifiés, analysés, documentés, résolus et investigués. Le SOC monitore également les applications pour identifier les possibles « cyber-attaques » ou intrusions (évènements), et détermine si ce sont des attaques réelles et malicieuses (incidents), et si elles peuvent avoir des impacts sur l'activité de l'entreprise. »

Il est donc de la responsabilité du SOC de gérer les évènements et incidents de sécurité qui surviennent sur le SI surveillé. Il doit être capable de les détecter, de procéder à des investigations afin d'obtenir le plus d'informations possible à leur sujet, puis de les résoudre.

Le SOC peut également donner des préconisations et règles de bonnes pratiques afin d'éviter que ces incidents se reproduisent. Le SOC permet également de maintenir un haut niveau de sécurité sur le système d'information qu'il monitore.

4.2 La détection des intrusions appliquée par les SOC

La gestion des évènements de sécurité est une tâche complexe, qui nécessite que ceux-ci soient, en premier lieu, détectés.

Pour cela, les SOC utilisent des systèmes de détection d'intrusions, identiques à ceux que j'ai présenté dans ce rapport et que j'ai également utilisé au cours de mon stage.

La première activité d'un SOC est la gestion de ces IDS, et en particulier leur mise en place et leur intégration au sein d'un SI. Cette intégration consiste à analyser le périmètre que le client cherche à monitorer, à *designer* l'infrastructure du SIEM et des IDS qui seront mis en place, puis à les installer et les configurer afin d'apporter une configuration de base pour les premières détections.

Afin d'apporter un meilleur niveau de détection, le SOC doit également maintenir les règles des systèmes de détection d'intrusions à jour, tout comme les directives de corrélation du SIEM mis en place.

Au cours de mon stage chez Conix Security, j'ai participé aux activités du SOC pour des clients de la société.

J'ai notamment mis en place chez ses clients les différentes règles de détection d'intrusions pour le NIDS Snort, ainsi que les directives de corrélation du SIEM OSSIM permettant d'améliorer la qualité des alarmes remontées par ce dernier.

J'ai également travaillé sur une configuration générique du NIDS OSSEC, qui puisse être déployée par défaut sur l'ensemble des serveurs Microsoft Windows des infrastructures clientes. J'ai détaillé cette partie dans la partie 4.4.1 de ce rapport.

L'activité principale d'un *Security Operations Center* étant la détection des intrusions, le SOC est responsable, pas seulement du maintien en conditions opérationnelles des SIEM, mais également de leur exploitation.

Les membres d'un SOC sont donc chargés de la gestion des alarmes créées par le SIEM, et notamment de leur résolution.

La résolution de ces alarmes de sécurité s'effectue en plusieurs étapes : la détection des intrusions, la recherche des informations sur celles-ci, puis l'envoi d'alertes aux responsables de la sécurité sur le SI, qui procéderont ensuite à la remédiation.

Lorsqu'une alarme est levée par les SIEM, il faut lancer des investigations permettant de lever les doutes qui subsistent (l'alarme est-elle un faux-positif ?) puis recueillir le maximum d'informations sur celle-ci.

4.3 Levée de doutes

La levée de doutes permet d'écarter les faux-positifs qui seraient éventuellement remontés par le SIEM. Après ce premier tri, la recherche d'informations importantes à propos de ces évènements est également nécéssaire pour faciliter leur résolution. Enfin, des rapports d'incidents complets sont envoyés aux personnes concernées.

Au cours de mon stage, j'ai également participé à cette partie des activités du SOC de Conix afin de préparer mon entrée dans l'entreprise à la suite du stage.

Les évènements générés par les IDS ou bien les alarmes créées par les SIEM proviennent de sources de données, desquelles ils extraient des informations qui sont affichées dans les alertes.

Cependant, les systèmes de détection d'intrusions ne possèdent pas toujours toutes les informations nécéssaires à l'appréciation de la criticité des alertes. J'ai en effet pu faire un constat simple à propos des règles des IDS : la plupart d'entre eux ne basent leur détection que sur un motif présent dans un paquet du réseau, sans se soucier du contexte dans lequel a été découvert ce paquet.

De nombreux faux-positifs sont obtenus de cette façon, et des investigations humaines sont indispensables. Les connaissances obtenues par l'expérience et par le contexte métier du client permettent de différencier les incidents réels des faux-positifs. Mais il faut pour cela obtenir des informations précises sur l'intrusion détectée.

La première étape, lorsqu'un évènement ou une alarme sont créés, est d'obtenir plus de données concernant le contexte de détection.

Au cours d'une détection basée sur le réseau, des captures réseaux sont prises avec des outils tels que *tcpdump* ou *Wireshark*[1]. De cette façon, il m'a été possible d'observer dans quelles conditions le trafic a été considéré comme malicieux, et ainsi de pouvoir affirmer si l'alarme est dûe à une réelle intrusion ou non.

Si les évènements sont remontés par des HIDS, il est intéressant de lire les journaux d'évènements des serveurs et services associés aux incidents. J'ai de ce fait pu recueillir des informations sur les actions en cours sur les applications au moment de l'« intrusion » pour déterminer avec plus de précision si l'alarme est un faux-positif.

Une fois que l'incident a été qualifié, il faut obtenir le plus d'informations possibles sur l'intrusion qui a été détectée.

Ces informations doivent être le plus exhaustives et précises possibles, afin d'assurer deux objectifs :

1. Informer les acteurs impliqués sur l'attaque, ses origines et ses conséquences ;
2. Permettre une résolution rapide et sûre.

Pour cela, le SOC rédige des fiches d'incidents qui contiennent toutes ces données, et qui permettent de répondre à plusieurs questions concernant l'intrusion : *Qui ? Quoi ? Quand ? Comment ? Où ?*

En répondant à toutes ces questions, je peux présenter dans un fichier d'incident les différents entités mises en causes, en obtenant notamment les adresses IP impliquées, et si possible les noms d'hôtes ou de domaines.

Ensuite, je peux également définir le fonctionnement de l'intrusion. Si c'est une attaque lancée contre un service, le *payload* applicatif lancé ou les différentes étapes de l'attaque sont listés. Dans le cas d'un malware, son nom, son fonctionnement ainsi que les actions malicieuses lancées sont détaillées dans la fiche d'incident.

La période au cours de laquelle s'est déroulée l'intrusion est également notée, ainsi que la méthode utilisée.

C'est pour obtenir ces informations que des éléments de contexte sont nécéssaires. Les *logs* permettent justement, sur une attaque réseau, d'obtenir les précédents échanges entre la(les) victime(s) et l'(les) attaquant(s). Sur une intrusion utilisant un malware ou partant d'un système qui dispose d'un HIDS, ces *logs* ainsi que les différents journaux du système permettent de retrouver des traces d'une précédente activité malicieuse qui pourrait conduire à l'intrusion en cours d'analyse.

Pour ma part, toutes les informations recueillies sont notées dans des fichiers textes permettant de centraliser les informations. Un exemple de modèle est donné en annexe E, page 105.

Suite à l'obtention de ces informations, des fiches d'incidents complètes et détaillées sont envoyées aux personnes responsables de la sécurité, afin qu'ils puissent lancer les actions nécéssaires à la résolution de ces incidents.

1. voir section 3.1.1, page 33.

Ainsi, chaque fiche d'incident contient des recommandations sur les actions à effectuer suite aux incidents relevés.

La plupart des incidents que j'ai eu l'occasion de traiter durant mon stage étaient liés à la présence de malwares sur des postes utilisateurs du système d'information client. Dans ces cas de figure, les principales recommandations sont d'effectuer des analyses antivirus complètes, afin de supprimer les malwares présents, et de *blacklister* les domaines et adresses IP contactées par ces malwares.

Si les intrusions proviennent de l'extérieur du réseau et visent des services du SI, nous recommandons d'appliquer les mises à jours et *patchs* proposés par les éditeurs des logiciels utilisés, ainsi que de procéder à des audits réguliers afin d'évaluer le niveau de sécurité de ces services.

4.4 Activités connexes

Au cours de mon stage, j'ai travaillé sur la détection de scénarios d'attaques précis, auxquels j'avais réflechi auparavant.

J'ai également participé aux activités du SOC de Conix pour des clients de l'entreprise, comme je l'ai présenté précédement. Au cours de ce temps passé au sein de l'équipe du SOC, j'ai travaillé sur plusieurs autres projets que je vais désormais présenter.

4.4.1 Configuration générique OSSEC

J'ai déjà présenté dans la partie 2.3.2 le HIDS OSSEC que j'ai utilisé au cours de mon stage.

Dans l'objectif de mettre en place cet IDS sur un grand nombre de serveurs et de postes utilisateurs chez des clients de Conix Security, nous souhaitions obtenir un fichier de configuration générique pour OSSEC à déployer par défaut lors de chaque installation sur une machine Windows.

Lors d'une installation du SIEM OSSIM, OSSEC est installé par défaut, et possède une configuration plutôt fournie. Elle vérifie notamment l'intégrité des fichiers sensibles du système (comme les fichiers de configuration de Windows : `%WINDIR%\win.ini`, `%WINDIR%\system.ini` ou `C:\boot.ini`), des executables utilisés par le système, et situés dans `%WINDIR%\System32\`, ainsi que des clés de la base de registre Windows.

Cette configuration est relativement complète pour une utilisation standard, cependant nous souhaitions obtenir un *monitoring* plus avancé qui nous permette de surveiller les modifications apportées à des parties sensibles de la configuration du serveur.

Nous avons identifié plusieurs informations à surveiller :

- Les fichiers du répertoire de l'administrateur ;
- Les programmes lancés au démarrage du système ;
- Les programmes installés ;
- La configuration Windows Update ;
- La configuration du DNS.

Pour ne pas surcharger mon rapport avec des fichiers de configuration, je donne les parties de configuration rajoutées au fichier par défaut d'OSSEC en annexe F.

4.4.2 Amélioration d'une solution SIEM éditeur

Les activités du SOC de Conix contenant également une part d'intégration de la solution SIEM OSSIM, j'ai manipulé à plusieurs reprises ce système d'exploitation, et notamment son installation sur des machines.

OSSIM est une distribution GNU/Linux basée sur Debian Squeeze [2], qui intègre par défaut les outils nécéssaires à la mise en place du SIEM. Étant basée sur Debian, l'in-

2. `http://www.debian.org/`

stallation de cette distribution est relativement simple pour tout utilisateur d'un système GNU/Linux.

Cependant l'installation de cette distribution sur un serveur, en préparation d'un déploiement chez un client, échouait sur la machine utilisée par notre client, et fonctionnait lors d'installations dans des machines virtuelles.

J'ai donc cherché à résoudre ce problème d'installation et j'ai observé une comportement spécial de l'installeur d'OSSIM.

En effet, celui-ci a été modifié de telle façon qu'il est impossible de choisir le partitionnement à appliquer sur les disques lors de l'installation. Or, ce partitionnement et la taille des partition étant fixés dans l'image ISO, si l'espace disponible sur la machine utilisée est insuffisant, l'installation de la distribution échoue sans plus d'informations.

J'ai donc dû modifier l'installeur présent sur l'image ISO pour ré-activer la présence du menu de choix du partitionnement.

Il faut quatre étapes principales pour retrouver ce menu dans l'installeur :

1. Désarchiver l'ISO ;
2. Trouver le bon fichier de configuration à modifier ;
3. Modifier ce fichier ;
4. Recréer l'ISO.

Pour « désarchiver » une image ISO, la meilleure technique que j'ai trouvé est de la monter sur le système comme un système de fichiers, puis de copier tous les fichiers dans un dossier afin de les modifier [3].

Le nom du fichier de configuration de l'installeur n'est pas documenté sur Internet, et la technique que j'ai utilisé pour le trouver est basique : ouvrir chaque fichier présent sur l'image ISO pour trouver celui qui semble contenir les informations souhaitées.

Après quelques recherches, ce fichier se trouve dans le répertoire `/simple-cdd`, et OSSIM en utilise deux : `defaultA.preseed` et `defaultB.preseed`. Le premier installe la version serveur d'OSSIM, le second installe la version sonde.

Dans ces fichiers, une section nommée `Partitioning` contient les étapes de partitionnement automatique des disques durs. Pour ré-activer le menu de partitionnement, j'ai remplacé cette partie par trois lignes de configuration dédiée à l'accès aux menus de partitionnement.

3. Afin de ne pas surcharger mon rapport, je ne donne pas ici toutes les commandes Linux utilisées. Elles ont cependant fait l'objet d'un article sur mon blog, article qui a inspiré cette partie de mon rapport. `http://quack1.me/linux_hack_install.html`

```
d-i partman-auto/choose_recipe select All files in one partition (
    recommended for new users)
d-i partman-auto/choose_recipe select Desktop machine
d-i partman-auto/choose_recipe select Multi-user workstation
```

Listing 29 – Lignes de configuration des fichiers .preseed permettant d'activer les menus de partitionnement dans les installeurs Gnome

Enfin, l'image ISO est recréée à partir des fichiers modifiés, et gravée sur un CD pour pouvoir être ensuite testée puis installée sur des machines physiques.

Conix Security dédie énormément de temps à l'expertise sur la solution SIEM OSSIM afin de former ses collaborateurs et pour améliorer le produit distribué aux clients.

En particulier, Conix a pendant plusieurs années développé *SIDSS*, un dérivé d'OSSIM, qui était déployé chez ses clients. Aujourd'hui, Conix a arreté le développement de *SIDSS* et ne travaille plus qu'avec OSSIM en maintenant un partenariat fort avec Alienvault, l'éditeur de ce SIEM.

4.4.3 Rédaction d'un article pour MISC Mag

En France, la revue bimestrielle MISC (pour *Multi-System & Internet Security Cookbook*) est le magazine de référence en matière de sécurité informatique. Chacun des numéros possède un « dossier », c'est à dire un ensemble de trois à quatre articles traitant d'un sujet précis.

Le dossier du numéro 69, publié le 30 Août 2013, est dédié aux SIEM et à la détection d'intrusions. C'est donc tout naturellement que mon maître de stage Adrien Vernois et moi-même avons proposé un article, intitulé *« SIEM/IDS : L'union fait-elle la force ? »*.

Dans cet article, nous avons tenté de répondre à cette question, afin de montrer si la mise en place et la gestion d'un SIEM est réellement un avantage au sein du processus de détection d'intrusions.

Nous avons procédé par étapes, en observant plusieurs IDS en action lors de la détection d'une attaque Web ciblée : l'exploitation d'une faille de type LFI identifiée par une CVE [4].

Nous avons dans un premier temps détecté l'exploitation de cette faille au niveau réseau avec Snort, puis au niveau du système avec OSSEC. Nous avons alors présenté les avantages de chacun de ses IDS, mais aussi leurs inconvénients.

Ensuite, nous avons montré comment il est possible de corréler ces évènements au travers d'un SIEM, notamment OSSIM, afin d'obtenir des alertes de sécurité plus précises.

4. CVE, ou *Common Vulnerabilities and Exposures*, est un dictionnaire recensant les informations publiques liées à des vulnérabilités de sécurité informatique. De façon plus courante, une CVE est un identifiant de la forme `CVE-AAAA-NNNN` (avec `AAAA` l'année de publication et `NNNN` un numéro incrémenté) qui permet d'identifier les vulnérabilités découvertes et publiées. Les CVE sont disponibles sur plusieurs sites Web, notamment `http://cve.mitre.org/`.

Enfin, la conclusion permet de montrer qu'un SIEM a des avantages et facilite la gestion des évènements de sécurité qui surviennent sur un SI, malgré que son utilisation ne résolve pas tous les problèmes inhérents à la détection d'intrusions.

La rédaction de cet article a été un excellent exercice qui m'a permis de mettre en pratique les connaissances obtenues au cours de mon stage. J'ai ainsi travaillé sur un cas pratique de détection d'une intrusion, que j'ai ensuite pu présenter dans cet article.

D'un point de vue non technique, la rédaction de cet article a été une bonne préparation à la rédaction de ce rapport, notamment dans la structuration des idées et l'enchaînement des parties. C'est également très intéressant et c'est une excellente expérience de pouvoir écrire dans une revue à tirage national au cours d'un stage.

CONCLUSION

Au cours de mon stage, j'ai manipulé de nombreux outils utilisés lors d'attaques ou d'audits d'applications Web. J'ai remarqué que ces outils, malgré leur quantité importante et les différentes informations qu'ils apportent, fonctionnent tous relativement de la même façon.

J'ai également remarqué que les attaques lancées contre des systèmes informatiques sont relativement semblables entre elles, et définir des scénarios d'attaques variés est donc complexe.

La détection de ces outils, attaques et scénarios, est rendue possible par l'utilisation de systèmes de détection d'intrusions s'appuyant sur des données précises provenant de différents endroits du système.

J'ai présenté des IDS réseau comme Snort, ou encore des HIDS comme OSSEC ou Audit qui basent leurs alertes sur des évènements provenant des systèmes d'exploitation sur lesquels ils sont installés.

La détection des intrusions survenant sur un système d'information est donc possible grâce à l'utilisation de nombreux IDS. Cette détection est complète, puisqu'elle peut se baser sur des évènements provenant de toutes les parties d'un SI : les postes utilisateurs, les serveurs, le réseau, les équipements réseaux, ...

Cependant, j'ai également montré que les alertes n'étaient pas toujours pertinentes. Les IDS remontent souvent des faux-positifs, c'est à dire des alertes qui ne sont pas liées à des intrusions mais à des utilisations normales du système. Dans le même temps, il est également possible de rencontrer des faux-négatifs, des comportements illégitimes qui ne sont pas détectés comme des intrusions par les IDS

Afin de palier à ce problème, je me suis penché sur l'utilisation des SIEM afin d'obtenir de meilleures informations concernant les intrusions qui surviennent sur un système d'information. En particulier, j'ai manipulé le moteur de corrélation présent sur ceux-ci, et qui

permet de corréler entre eux les évènements provenant de plusieurs IDS, ou plus généralement de plusieurs sources de données du système.

En utilisant ces SIEM, j'ai remarqué qu'il était possible d'écrire des règles de détection IDS beaucoup plus génériques, qui vont générer plus d'évènements, et de ce fait, plus de faux-positifs.

Au contraire, les directives de corrélation présentes sur le SIEM seront beaucoup plus précises et permettront d'éliminer ces faux positifs pour ne créer d'alarmes que lorsque les intrusions ou les évènements de sécurité seront réellement avérés.

Les SIEM permettent donc d'obtenir des informations beaucoup plus pertinentes que l'utilisation seule d'un ou plusieurs systèmes de détection d'intrusions. Cependant, cette détection n'est pas parfaite, et malgré que le nombre de faux-positifs et faux-négatifs soit diminué, il ne permet pas de totalement les éliminer.

La détection d'intrusions n'est pas uniquement basée sur la détection des attaques à partir de leur signature. Il est possible d'utiliser d'autres techniques, en remplacement ou en complément de cette approche.

On peut par exemple aborder la détection d'intrusions comportementale. Dans ce cas, on ne définit plus l'ensemble des comportements anormaux pour lesquels il faut lever des alertes, mais l'ensemble des utilisations légitimes du système, pour lesquels il ne faut pas remonter d'alarmes. Ainsi, le problème est pris à l'envers et tout comportement non prévu sera considéré comme une intrusion. Cette technique peut se baser sur la PSSI [5] du système d'information, et ainsi coller au plus près des utilisation autorisées du SI.

On peut également chercher à réaliser des analyses comportementales du réseau permettant de définir les activités suspectes. Une période d'apprentissage définit les seuils de traffic ou d'utilisation de certains protocoles et/ou destinations/sources au delà desquels cette activité sera considérée comme supicieuse.

Il existe également des techniques de détection d'intrusions basées sur des analyses protocolaires. L'analyse générique d'un protocole réseau ou système permet de détecter si l'utilisation qui en est faite est légitime ou non, en inspectant les données du protocole lui-même, et non plus une séquence d'octets à chercher dans le message [6].

Ces deux nouvelles techniques de détection d'intrusions pourront faire l'objet de stages l'an prochain chez Conix, qui recherche et recrute de nombreux stagiaires depuis plusieurs années.

Ce stage réalisé en entreprise à la fin de mon Master en Sécurité de l'Information et Cryptologie à été pour moi une grande réussite et m'a permis de mettre en pratique et de

5. La PSSI, ou Politique de Sécurité du Système d'Information, est développée par le RSSI d'une entreprise et définit l'ensemble des utilisations autorisées et interdites du système. La PSSI donne également des règles concernant la gestion des mots de passe, concernant l'utilisation ou non de certains services Internet ou concernant l'accès aux mails professionnels en dehors du lieu de travail par exemple.

6. Par ces fonctions d'introspection et de décodage des protocoles HTTP et TCP, Snort réalise en partie de l'analyse protocolaire.

consolider les connaissances acquises lors de mon cursus universitaire et de mes recherches personnelles.

J'ai pû, grâce à l'utilisation de nombreux outils d'audits, apprendre de nouvelles techniques et parfaire mes connaissances dans ce domaine. Cependant, malgré l'utilisation importante de ces outils, mon stage était consacré en grande partie à la détection d'intrusions.

Ce stage m'a donc permis d'obtenir énormément de connaissances sur ce sujet, notamment par la manipulation importante des IDS tels que Snort ou OSSEC, ainsi qu'en matière de corrélation d'évènements par l'utilisation du SIEM OSSIM.

Enfin, la manipulation et la gestion avancée d'outils de détection d'intrusions ainsi que les activités connexes, telles que la levée de doutes et l'investigation sur ces incidents ont conforté mon choix de m'orienter professionnellement dans le milieu de la détection d'intrusions et de l'inforensique.

« *Stay hungry. Stay foolish* »
Steve Jobs – Université de Stanford – 12 Juin 2005

GLOSSAIRE

O-day
: Une faille dite « 0-day » est une faille qui n'a pas encore été dévoilée à la communauté et qui est donc encore inconnue des éditeurs de solutions de sécurité ou d'antivirus.. 22

Audit
: Le logiciel *Audit* est, sur les systèmes GNU/Linux, un système de détection d'intrusions qui permet de monitorer les accès à certains fichiers du système.. 14, 20, 21, 53, 54, 71

CMS
: Un CMS, *Content Management System*, est une application Web permettant la gestion et la mise à jour dynamique de sites Web. Les plus connus sont Wordpress(`https://wordpress.org/`), MediaWiki(`http://www.mediawiki.org/wiki/MediaWiki/fr`), ou encore Spip(`http://www.spip.net/`).. 35

CVE
: CVE, ou *Common Vulnerabilities and Exposures*, est un dictionnaire recensant les informations publiques liées à des vulnérabilités de sécurité informatique. De façon plus courante, une CVE est un identifiant de la forme `CVE-AAAA-NNNN` (avec `AAAA` l'année de publication et `NNNN` un numéro incrémenté) qui permet d'identifier les vulnérabilités découvertes et publiées. Les CVE sont disponibles sur plusieurs sites Web, notamment `http://cve.mitre.org/`.. 69

DSI
: La DSI est la Direction des Systèmes d'Information. Ce service est celui qui met en place et gère les ressources informatiques en entreprise.. 9, 63

expression régulière
: Une expression régulière, ou rationnelle, est une chaîne de caractères, aussi appelée motif, qui décrit un ensemble de chaînes de caractères. Ces expressions sont souvent

utilisées dans des applications afin de vérifier si les données saisies par l'utilisateur sont conformes à un format attendu.. 15, 50

HIDS

Un HIDS, ou *Host-Based Intrusion Detection System*, est un système de détection d'intrusions basé sur des évènements provenant du système d'exploitation sur lequel il est lancé.. 10, 50, 52, 53, 59, 60, 65, 67, 71, 107

HTML

Le HTML, ou *Hypertext Markup Language*, est le format de données utilisé pour représenter les pages Web. Son format permet de strucutrer sémantiquement et de mettre en forme le contenu des pages affichées.. 48, 49

HTTP

Le protocole HTTP, ou *HyperText Transfer Protocol*, est un protocole de communication client/serveur utilisé principalement par les navigateurs Web pour accéder à des serveurs contenant des données. Le protocole HTTP est un protocole de la couche application du modèle TCP/IP, reposant sur le protocole TCP.. 16, 35, 45

IDMEF

Le format IDMEF, ou *Intrusion Detection Message Exchange Format* est un format conçu par l'IETF (dans la RFC 4765 (`http://www.rfc-editor.org/rfc/rfc4765.txt`)) afin de stocker et partager les informations de façon standard entre plusieurs IDS.. 25

IDS

Un IDS, ou Système de Détection d'Intrusions, est un logiciel permettant de détecter, au moyen de règles définies par l'utilisateur, des activités anormales ou suspectes sur un système cible, permettant ainsi de détecter des intrusions, réussies ou non.. vi, vii, 4, 43, 44, 51–54, 60–64, 67, 69, 71–73, 87

injection SQL

Une injection SQL, ou *SQLi* est une attaque informatique visant à modifier les requêtes SQL executées par l'application attaquée afin d'induire un nouveau comportmeent et récupérer des informations depuis le serveur. Pour cela, l'attaquant envoie à l'application vulnérable un *payload* spécifique permettant de lancer l'attaque.. 16, 39, 81, 94, 98, 100

IP

Le protocole IP, ou *Internet Protocol*, désigne l'ensemble des protocoles utilisés sur des réseaux informatiques, notamment sur Internet. Il nécessite pour fonctionner la distribution d'adresses IP uniques sur le réseau (à tout instant, sur le réseau, deux machines ne peuvent avoir la même adresse IP.).. 15, 52, 57, 65, 66

IPS

Un IPS, ou Système de Prévention d'Intrusions, est un logiciel permettant non pas de détecter les intrusions (comme le font les IDS), mais d'empêcher que ces intrusions se produisent. Le principe de fonctionnment reste cependant le même que pour les IDS, les actions considérées comme anormales par les règles de l'IPS seront bloquées.. 15

ISO 27000
La suite ISO/CEI 27000 (ou « Normes ISO 27k ») contient les normes publiées par l'ISO (*International Organization for Standardization*) et l'IEC (*International Electrotechnical Commission*) au sujet de la sécurité informatique. Ces normes contiennent notamment les bonnes pratiques en matière de management de la sécurité, ainsi que pour la définition et la mise en place d'un système de gestion de la sécurité de l'information.. v

LFI
Une LFI, ou *Local File Inclusion*, est une attaque Web qui permet à un attaquant d'inclure dans la page Web vulnérable un fichier présent sur le serveur Web. Typiquement, ce fichier est le fichier `/etc/passwd` sur les systèmes UNIX.. 48, 52–54, 69, 95, 98, 99

Main-In-the-Middle
Une attaque dite par *Man-in-the-Middle* est une attaque au cours de laquelle l'attaquant se place entre la victime et le service qu'elle contacte, en détournant la totalité du trafic afin de pouvoir accéder au contenu des échanges. Si la victime est nommé *Alice*, le service qu'elle contacte *Bob* et que l'attaquant est *Ève*, alors, pour Alice, Ève se fera passer pour Bob et Bob dialoguera avec Ève en pensant le faire avec Alice.. 23

modèle TCP/IP
Le modèle TCP/IP définit le standard de communication réseau des systèmes informatiques. Il donne les fonctionnalités nécéssaires à la communication de systèmes hétérogènes sur un même réseau, et notamment une architecture en couches qui permet à des protocoles de haut niveau de s'appuyer sur les protocoles sous-jacent pour assurer les étapes d'établissement des connexions et de gestion du réseau. Il est inspiré du modèle OSI. Les différentes couches du modèle TCP/IP sont données en figure 26, page 104.. 50, 104

NIDS
Un NIDS, ou *Network-Based Intrusion Detection System*, est un système de détection d'intrusions basé sur des évènements provenant d'un réseau informatique.. 10, 50, 52, 57, 60, 64

Open-Source
Le terme *Open-Source* désigne les logiciels dont la licence d'utilisation respecte certains critères, comme notamment l'accès au code-source, la libre distribution et la possibilité d'en créer des travaux dérivés.. 28, 33, 44, 49

OSSEC
OSSEC est un *Host-Based Intrusion Detection System*, qui permet de réaliser des analyses de fichiers de *logs*, des contrôles d'intégrité, de la détection de *rootkits*. Il est compatible avec de nombreux systèmes d'exploitation, comme GNU/Linux,

Windows, Mac OS, ou encore Solaris. `http://www.ossec.net/`. 14, 17–21, 25, 28, 30, 53, 54, 59, 64, 67, 69, 71, 73, 107

OSSIM
OSSIM, ou *Open Source Security Information Management* est une solution SIEM Open-Source développée par la société Alienvault, et déployée chez ses clients par Conix Sécurity. `http://communities.alienvault.com/`.. 54, 55, 58, 60, 64, 67–69, 73

payload
Le *payload*, ou « charge utile », est le contenu réellement envoyé et manipulé par une application. Dans le contexte d'une attaque informatique, ce terme désigne le contenu ou le code exécutable qui va permettre de déclencer une attaque et/ou nuire à un système.. 38–41, 44–49, 52, 55–57, 65, 96–98

port réseau
Sur la couche 4, ou transport, du modèle TCP/IP, un port permet de distinguer les différents interlocuteurs de la machine. Chaque programme informatique écoute ou envoi des données sur un port différent de la machine. Sur un serveur, les ports les plus connus sont le 80 (HTTP pour un serveur Web), 22 (SSH), ou encore le 25 (serveur SMTP (`http://fr.wikipedia.org/wiki/Smtp`) pour un serveur de mails).. 49

proxy
Un *proxy* est un équipement logiciel ou matériel qui est placé, sur un réseau, entre les ressources internes et l'Internet. Il permet de filtrer et/ou modifier le contenu auquel les utilisateurs internes du réseau accède.. 55, 60

PSSI
La PSSI, ou Politique de Sécurité du Système d'Information, est développée par le RSSI d'une entreprise et définit l'ensemble des utilisations autorisées et interdites du système. La PSSI donne également des règles concernant la gestion des mots de passe, concernant l'utilisation ou non de certains services Internet ou concernant l'accès aux mails professionnels en dehors du lieu de travail par exemple.. 72

reverse proxy
Un *reverse proxy*, ou proxy inverse, est un équipement logiciel ou matériel placé en entrée d'un réseau et permettant aux utilisateurs provenant de l'Internet d'accéder à des ressources placées à l'intérieur du réseau, le plus souvent après une phase d'authentification ou d'identification.. 22, 23, 103

RSSI
Dans une entreprise, le RSSI est le Responsable de la Sécurité des Systèmes d'Informations. Généralement rattaché à la Direction des Systèmes d'Information, il est chargé de toutes les missions en rapport avec la sécurité du SI et des données qu'il contient.. 3

SI
Un SI, ou Système d'Information désigne la totalité des ressources liées à la gestion

de l'information (généralement informatisée) en entreprise.. vi, 4, 7, 18, 21, 23, 52, 53, 57, 60, 63, 64, 66, 70–72

SIEM

Un SIEM, ou *Security Information Management System*, ou Système de Gestion de la Sécurité du Système d'Information. Un SIEM permet de centraliser les informations relatives à la sécurité du SI, notamment en collectant les *logs* de différentes sources (services système, IDS, etc...), puis en offrant la possibilité de les corréler entre eux afin de lever des alertes de sécurité plus ciblées. Enfin, les solutions SIEM sont souvent équipées de systèmes d'archivage à haute sécurité et haute disponibilité permettant d'utiliser les informations stockées lors de procédures judiciaires.. vii, 3, 4, 25, 52, 54, 57–61, 63, 64, 67, 69–73

Snort

Snort est un *Network-Based Intrusion Detection System*, qui permet de monitorer un réseau, en comparant les paquets transitant sur ce flux à des signatures prédéfinies. `https://www.snort.org/`. 14–19, 21–25, 28, 30, 35, 36, 38, 39, 41, 43, 44, 46, 49, 50, 52–58, 64, 69, 71–73, 87–90, 93–95

SOC

Un SOC, ou *Security Operations Center*, est dans une entreprise le service dédié à la gestion de la sécurité, notamment à la sécurité de l'information. Généralement, le SOC est responsable de la gestion des évènements de sécurité généré par les IDS.. 3, 105

SQL

Le langage SQL, ou *Structured Query Language*, est un langage informatique structuré permettant de manipuler les informations d'une base de données.. 39, 40, 46, 47, 82

SSH

SSH, ou *Secure Shell*, est le protocole de connexion et communication sécurisées à distance à des systèmes d'exploitation Unix.. 17, 53, 57, 58, 60

SSII

Une SSII, ou Société de Services en Ingénierie Informatique, est une entreprise spécialisée dans la réalisation de missions de services liées à l'informatique.. 1

TAP

Un TAP réseau est un dispositif réseau permettant de surveiller un réseau informatique sans le perturber. Les flux réseaux sont dupliqués afin qu'un dispositif réseau puisse accéder aux données qui transitent, sans modifier ces flux ni dégrader la qualité de service.. 23

TCP

Le protocole TCP, ou *Transmission Control Protocol*, est un protocole de communication entre deux machines connectées, utilisant la couche 4 du modèle TCP/IP. Les données des applications utilisant le protocole TCP sont découpées en paquets et envoyées à la suite sur le réseau. Les spécifités de ce protocole assurent que les flux

seront reconstruits dans le même ordre que les flux de départ. Le protocole TCP est défini dans la RFC 793 (`http://www.rfc-editor.org/rfc/rfc793.txt`).. 50, 57

UDP
Le protocole UDP, ou *User Datagram Protocol*, est un protocole de communication semblable à TCP, à la différence qu'UDP communique en mode « non-connecté ». L'intégrité et l'ordre d'arrivée des flux n'est donc pas garanti. Ce protocole est défini dans la RFC 768 (`http://www.rfc-editor.org/rfc/rfc768.txt`).. 50

Unix
Un système de type Unix est un système d'exploitation dont le comportement est semblable à celui d'un système Unix. Les principaux systèmes Unix actuels sont les systèmes GNU/Linux, BSD, Mac OSX et Solaris. 10, 17, 18, 40, 48, 53

URI
Une URI, ou *Uniform Resource Identifier*, est la chaîne de caractères qui permet d'identifier de façon unique une ressource sur un réseau informatique (généralement sur Internet).. 16, 53

XML
Le format XML, ou *Extensible Markup Language*, est un format de représentation des données générique et extensible, permettant de structurer les informations de façon hiérarchique.. 29

XSS
Une attaque de type XSS, ou *Cross-Site Scripting*, est une attaque Web visant à injecter dans une page Web du contenu afin de lancer des actions dans le navigateur des visiteurs de la page.. 39, 41, 47–49, 81, 95, 98, 99, 101

TABLE DES FIGURES

1 Organigramme du groupe Conix 2

2 Matrice de risques : Évolution du risque en fonction de l'exploitabilité d'une faille et des conséquences de l'exploitation 8
3 Diagramme d'évaluation du niveau de risque d'un périmètre. 9
4 Architecture d'une plateforme de détection d'intrusions. Article issu de l'article [BHM+06] 12
5 Processus de traitement d'un incident par OSSEC. Image tirée de la documentation OSSEC : `http://www.ossec.net/files/auscert-2007-dcid.pdf`, page 14. 19
6 Manque de fiabilité et/ou de pertinence si le modèle utilisé est incomplet ou incorrect. Image tirée du cours de Guillaume Hiet et Ludovic Mé sur les IDS donné dans le cadre du master Cryptis. 23
7 Exemple de graphiques réalisés par le SIEM OSSIM. On peut observer dans le graphique du bas l'évolution du nombre d'évènements remontés dans le SIEM par heure, et dans celui du haut la répartition des sources des évènements. On remarque que ceux-ci ne proviennent pas que de systèmes de détection d'intrusions, mais aussi des systèmes d'exploitation, d'applications, ou encore des journaux d'authentification. 27
8 Exemple de directive de corrélation OSSIM. Plus les IDS détectent des scans SSH, plus le SIEM augmente la criticité de l'alerte. Directive issue d'OSSIM (en version Open-Source). 29
9 Comparatif des fonctionnalités des IDS réseau, système et des SIEM, dans le cas d'attaques transitant, en premier lieu, par le réseau. 31

10 Exemple de capture réseau analysée dans Wireshark 34
11 Requêtes envoyées par Blind Elephant à l'application Web cible. 35
12 Requête HTTP envoyée par WhatWeb à une application cible 36

13 Diagramme présentant des statistiques sur les parties des paquets utiles à la détection des outils d'audit. 43
14 Capture réseau d'une attaque lancée par l'outil WhatWeb, et capture de la même attaque lancée par l'outil modifié pour masquer son nom dans le *User-Agent*. 44
15 Capture réseau d'une attaque par force brute permettant de lister les répertoires accessibles sur un serveur Web. 46
16 Schéma d'évolution du niveau de criticité d'une attaque Web de type *LFI*. . 55
17 Schéma d'évolution du niveau de criticité d'une attaque lancée avec le *scanner* de Burp. 56
18 Schéma d'évolution du niveau de criticité d'une attaque « generique ». . . . 58
19 Schéma d'évolution du niveau de criticité des évènements lors de la surveillance des authentifications. 59
20 Comparatif détaillé des fonctionnalités des NIDS, HIDS et SIEM. 61

21 Diagramme présentant des statistiques sur les parties des paquets utiles à la détection des outils d'audit. 96

22 Visualisation graphique de l'expression régulière de détection générale d'attaques de type injection SQL. 100
23 Visualisation graphique de l'expression régulière de détection générale d'attaques de type XSS. 101
24 Visualisation graphique de l'expression régulière de détection générale d'attaques de type XSS. *Version raccourcie* 101
25 Visualisation graphique de l'expression régulière de détection générale d'attaques de type LFI. 102

26 Architecture en couches du modèle TCP. 104

LISTINGS

1 Exemple de règle Snort : Détection d'une injection SQL. 16
2 Exemple de règle Snort : Détection d'une injection SQL, version 2. 16
3 Exemple de règle Snort : Détection d'une injection SQL, version 3. 17
4 Exemple de règle OSSEC : Configuration de l'agent pour le monitoring des logs Apache. 19
5 Exemple de règle OSSEC : Création d'un décodeur serveur pour les logs Apache. 19
6 Exemple de règle OSSEC : Création de règles serveur pour les logs Apache. 19
7 Exemple de règle Audit : Accès en lecture ou écriture au fichier `/etc/passwd` par l'utilisateur `www-data` (uid :33). 20
8 Règle Snort de détection de Blind Elephant. 35
9 Règle Snort de détection de WhatWeb. 36
10 Règle Snort de détection de SkipFish. 37
11 Règle Snort de détection du module `pma` de WebSploit. 38
12 Règle Snort de détection du module `apache_users` de WebSploit. 38
13 Règle Snort de détection de WebSecurify. 38
14 Règle Snort de détection du premier payload de Xsser. 39
15 Règle Snort de détection du second payload de Xsser. 39
16 Règle Snort de détection de l'exploitation d'une injection SQL par Dark-MySQLi afin d'obtenir les bases de données. 40
17 Règle Snort de détection de FImap. 40
18 Règle Snort de détection de sqlsus. 40
19 Règle Snort de détection de XSSF. 41
20 Règle Snort de détection de BEEF. 41
21 Règle Snort de détection générique d'attaques de type *SQLi*. 47
22 Règle Snort de détection générique d'attaques de type *XSS*. 48
23 Règle Snort de détection générique d'attaques de type *XSS*. *Version raccourcie* 48
24 Règle Snort de détection générique d'attaques de type *LFI*. 48
25 Règle Snort de détection basique d'une attaque de type *LFI*. 53

26 Règle Audit de détection des accès en lecture ou écriture au fichier `/etc/-passwd` par l'utilisateur `www-data` (uid :33). 54
27 Règle Snort de détection basique d'une payload utilisé par le *scanner* de Burp. 56
28 Règle Snort de détection d'un scan *SYN* effectué par *nmap*. 57
29 Lignes de configuration des fichiers .preseed permettant d'activer les menus de partitionnement dans les installeurs Gnome 69
30 Règle Snort détaillée de détection de Blind Elephant. 87
31 Règle Snort détaillée de détection de WhatWeb. 88
32 Règle Snort détaillée de détection de SkipFish. 88
33 Règle Snort détaillée de détection du module `pma` de WebSploit. 88
34 Règle Snort détaillée de détection du module `apache_users` de WebSploit. 89
35 Règle Snort détaillée de détection de WebSecurify. 89
36 Règle Snort détaillée de détection de XSSer. 89
37 Règle Snort détaillée de détection de XSSer. 89
38 Règle Snort détaillée de détection de DarkMySQLi – Détection des colonnes d'une table. 90
39 Règle Snort détaillée de détection de DarkMySQLi – Fuzzing d'une table. . 90
40 Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base, *blind SQLi* (1er payload). 90
41 Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base, *blind SQLi* (2eme payload). 90
42 Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base (1er payload). 91
43 Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base (2eme payload). 91
44 Règle Snort détaillée de détection de DarkMySQLi – Dump de la base, *blind SQLi*. 91
45 Règle Snort détaillée de détection de DarkMySQLi – Dump de la base. . . . 91
46 Règle Snort détaillée de détection de DarkMySQLi – Récupération de toutes les informations possibles (1er payload). 91
47 Règle Snort détaillée de détection de DarkMySQLi – Récupération de toutes les informations possibles (2eme payload). 92
48 Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données, *blind SQLi* (1er payload). 92
49 Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données, *blind SQLi* (2eme payload). 92
50 Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données (1er payload). 92
51 Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données (2eme payload). 92
52 Règle Snort détaillée de détection de DarkMySQLi – Récupération des informations sur les bases de données, *blind SQLi*. 93

53 Règle Snort détaillée de détection de fimap. 93
54 Règle Snort détaillée de détection de sqlsus. 93
55 Règle Snort détaillée de détection de XSSF. 94
56 Règle Snort détaillée de détection de BEEF. 94
57 Règle Snort détaillée de détection générique d'attaques de type *SQLi*. . . . 94
58 Règle Snort détaillée de détection générique d'attaques de type *XSS*. 95
59 Règle Snort détaillée de détection générique d'attaques de type *XSS*. *Version raccourcie* 95
60 Règle Snort détaillée de détection générique d'attaques de type *LFI*. 95
61 Modèle de fichier permettant la prise d'informations concernant des intrusions. 105
62 Règle à ajouter afin d'obtenir une configuration générique pour OSSEC . . 107

BIBLIOGRAPHIE

[Ale12] AlertLogic.com. *IDS/IPS Signature Bypassing (Snort)*. 2012. http://www.alertlogic.com/idsips-signature-bypassing-snort/.

[BHM+06] Christophe Bidan, Guillaume Hiet, Ludovic Mé, Benjamin Morin, and Jacob Zimmermann. Vers une détection d'intrusions à fiabilité et pertinence prouvables. *Revue de l'électricité et de l'électronique*, 9 :75, 10 2006.

[Che11] Mohammed Cherifi. Sql injection : Les techniques d'évasion de filtres. 2011. http://www.mcherifi.org/hacking/sql-injection-les-techniques-devasion-de-filtres.html.

[Dah10] Johannes Dahse. *SQLi filter evasion cheat sheet (MySQL)*. 2010. http://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/.

[eLM12] Guillaume Hiet et Ludovic Mé. Détection d'intrusions. In *Master Cryptis*, 2012.

[FSF07] Free Software Foundation FSF. Gnu general public license. 2007. https://www.gnu.org/licenses/gpl.html.

[Ope13] OpenSuse. Understanding linux audit. 2013. http://doc.opensuse.org/products/draft/SLES/SLES-security_sd_draft/cha.audit.comp.html.

[OWA12] OWASP. *XSS Filter Evasion Cheat Sheet*. 2012. https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet.

[Ruf09] Nicolas Ruff. La sécurité est un Échec. In *SSTIC*, 2009.

[Tea13] Snort Team. Snort user manual (v2.9.5). 2013. https://s3.amazonaws.com/snort-org/www/assets/166/snort_manual.pdf.

[Wik04] Wikipédia. Snort. 2004. https://fr.wikipedia.org/w/index.php?title=Snort.

[Wik07] Wikipédia. Portail :Sécurité Informatique. 2007. https://fr.wikipedia.org/w/index.php?title=Portail:Sécurité_informatique.

[Wik10] Wikipédia. *Information security operations center*. 2010. `http://en.wikipedia.org/wiki/Information_security_operations_center`.

ANNEXE A

RÈGLES SNORT DÉTAILLÉES

Dans la partie 3 de mon rapport, j'ai présenté les différents outils d'audits que j'ai analysé et détecté au cours de mon stage.

Afin de ne pas surcharger ce rapport, je n'ai donné dans ces parties que des règles de détection simplifiées, permettant de voir le fonctionnement de la détection de manière rapide.

Je donne ici les règles complètes, telles qu'elles ont été écrites et mises en place dans les IDS au cours du stage.

A.1 Outils d'audit automatisés

Blind Elephant

Blind Elephant est un outil d'audit présenté en page 35 de ce rapport. La règle simplifiée est détaillée en page 35.

Cette règle complète ajoute une référence, ainsi qu'un *tag* permettant de trier les alertes remontées par Snort.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS
    BlindElephant \"Fail Case\" test";content:"GET";http_method;nocase;
    content:"/should/not/exist.html";http_uri;flow:established, to_server;
    classtype:attempted-recon; reference:url,blindelephant.sourceforge.net
    /; sid:999003;rev:2;)
```

Listing 30 – Règle Snort détaillée de détection de Blind Elephant.

WhatWeb

Whatweb est un outil d'audit présenté en page 36 de ce rapport. La règle simplifiée est détaillée en page 36.

Cette règle complète ajoute un *tag* permettant de trier les alertes remontées par Snort.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS
    WhatWeb [User-Agent] detected";content:"WhatWeb";nocase;http_header;
    pcre:"/User-Agent:.*WhatWeb\/\d+.\d+/i"; classtype:attempted-recon; sid
    :999004;rev:1;)
```

Listing 31 – Règle Snort détaillée de détection de WhatWeb.

SkipFish

SkipFish est un outil d'audit présenté en page 36 de ce rapport. La règle simplifiée est détaillée en page 37.

Cette règle complète ajoute une référence, ainsi qu'un *tag* permettant de trier les alertes remontées par Snort, mais aussi un `threshold`, qui permet de ne lever une alerte qu'après un nombre donnée d'évènements reçus au cours d'une période de temps donnée.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS
    SkipFish [User-Agent] detected";content:"User-Agent"; nocase;
    http_header; pcre:"/User-Agent:.*SF\/\d+\.\d+/i"; http_header; flow:
    established,to_server; classtype:attempted-recon;  threshold: type both
    , track by_src, count 10, seconds 300; reference:url,code.google.com/p/
    skipfish/; sid:999002; rev:4;)
```

Listing 32 – Règle Snort détaillée de détection de SkipFish.

WebSploit

WebSploit est un outil d'audit présenté en page 37 de ce rapport. La règle simplifiée est détaillée en page 38.

Cette règle complète ajoute une référence, ainsi qu'un *tag* permettant de trier les alertes remontées par Snort, mais aussi un `threshold`, qui permet de ne lever une alerte qu'après un nombre donnée d'évènements reçus au cours d'une période de temps donnée.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS Brute
    -Force to find PhpMyAdmin. Possible tools : WebSploit Module 'web/pma'"
    ; content:"GET"; http_method; content:"/phpmyadmin"; http_uri; nocase;
    classtype:attempted-recon; sid:999007; rev:1;)
```

Listing 33 – Règle Snort détaillée de détection du module `pma` de WebSploit.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT TOOLS
    WebSploit Module 'web/apache_users'"; content:"GET"; http_method;
    content:"/~"; http_uri; classtype:attempted-recon; threshold: type both
    , track by_src, count 150, seconds 60; sid:999008; rev:1;)
```

Listing 34 – Règle Snort détaillée de détection du module `apache_users` de WebSploit.

WebSecurify

WebSecurify est un outil d'audit présenté en page 38 de ce rapport. La règle simplifiée est détaillée en page 38.

Cette règle complète ajoute une référence, ainsi qu'un *tag* permettant de trier les alertes remontées par Snort, mais aussi un `threshold`, qui permet de ne lever une alerte qu'après un nombre donnée d'évènements reçus au cours d'une période de temps donnée.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS WebSecurify
    Scanner User-Agent Detected"; content:"User-Agent"; nocase; http_header
    ; content:"Websecurify"; nocase; http_header; distance:1; classtype:
    attempted-recon; threshold: type both, track by_src, count 10, seconds
    300; reference:url,http://www.websecurify.com/; sid:999022; rev:1;)
```

Listing 35 – Règle Snort détaillée de détection de WebSecurify.

xsser

xsser est un outil d'audit présenté en page 39 de ce rapport. La règle simplifiée est détaillée en page 39.

Cette règle complète ajoute une référence, ainsi qu'un *tag* permettant de trier les alertes remontées par Snort, mais aussi un `threshold`, qui permet de ne lever une alerte qu'après un nombre donnée d'évènements reçus au cours d'une période de temps donnée.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS XSSer 1st payload
    Detected"; content:"><SCRIPT>alert('"; nocase; http_uri; classtype:web-
    application-attack; flowbits:set,audit.xsser.payload0; threshold: type
    limit, track by_src, count 1, seconds 60; sid:999028; rev:1;)
```

Listing 36 – Règle Snort détaillée de détection de XSSer.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS XSSer 2d payload
    Detected"; content:"<IMG SRC="; nocase; http_uri; classtype:web-
    application-attack; flowbits:set,audit.xsser.payload1; threshold: type
    limit, track by_src, count 1, seconds 60; sid:999029; rev:1;)
```

Listing 37 – Règle Snort détaillée de détection de XSSer.

DarkMySQLi

DarkMySQLi est un outil d'audit présenté en page 39 de ce rapport. Cet outil possède de nombreux modules et j'ai, pour tous les détecter, écrit 15 règles de détection Snort. Une règle simplifiée est détaillée en page 40.

Ces règles complètes ajoutent un *tag* permettant de trier les alertes remontées par Snort, mais aussi un `threshold`, qui permet de ne lever une alerte qu'après un nombre donnée d'évènements reçus au cours d'une période de temps donnée.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--
    findcol' attack"; flow:to_server,established; content:" AND 1=2 UNION
    SELECT "; nocase; http_uri; classtype:web-application-attack; threshold
    : type limit, track by_src, count 1, seconds 60; sid:999030; rev:1;)
```

Listing 38 – Règle Snort détaillée de détection de DarkMySQLi – Détection des colonnes d'une table.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--fuzz
     --blind' attack"; flow:to_server,established; content:" and (SELECT 1
    from "; nocase; http_uri; content:" limit 0,1)=1 "; nocase; http_uri;
    classtype:web-application-attack; threshold: type limit, track by_src,
    count 1, seconds 60; sid:999031; rev:1;)
```

Listing 39 – Règle Snort détaillée de détection de DarkMySQLi – Fuzzing d'une table.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--
    schema --blind' attack (1)"; flow:to_server,established; content:" and
    ((SELECT COUNT(table_name) FROM information_schema.TABLES WHERE
    table_schema=0x"; nocase; http_uri; classtype:web-application-attack;
    threshold: type limit, track by_src, count 1, seconds 60; sid:999032;
    rev:1;)
```

Listing 40 – Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base, *blind SQLi* (1er payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--
    schema --blind' attack (2)"; flow:to_server,established; content:" and
    ascii(substring((SELECT table_name FROM information_schema.TABLES WHERE
     table_schema=0x"; nocase; http_uri; classtype:web-application-attack;
    threshold: type limit, track by_src, count 1, seconds 60; sid:999033;
    rev:1;)
```

Listing 41 – Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base, *blind SQLi* (2eme payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--
    schema' attack (1)"; flow:to_server,established; content:"concat(0x1e,0
    x1e,COUNT(table_schema),0x1e,0x20)"; nocase; http_uri; classtype:web-
    application-attack; threshold: type limit, track by_src, count 1,
    seconds 60; sid:999034; rev:1;)
```

Listing 42 – Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base (1er payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--
    schema' attack (2)"; flow:to_server,established; content:"concat(0x1e,0
    x1e,table_schema,0x1e,table_name,0x1e,column_name,0x1e,0x20)"; nocase;
    http_uri; classtype:web-application-attack; threshold: type limit,
    track by_src, count 1, seconds 60; sid:999035; rev:1;)
```

Listing 43 – Règle Snort détaillée de détection de DarkMySQLi – Détection des schémas de la base (2eme payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--dump
     --blind' attack"; flow:to_server,established; content:" and ((SELECT
    COUNT(*) FROM "; nocase; http_uri; content:" and ascii(substring((
    SELECT concat("; nocase; http_uri; content:" ) FROM "; nocase; http_uri
    ; classtype:web-application-attack; threshold: type limit, track by_src
    , count 1, seconds 60; sid:999036; rev:1;)
```

Listing 44 – Règle Snort détaillée de détection de DarkMySQLi – Dump de la base, *blind SQLi*.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--dump
    ' attack"; flow:to_server,established; content:"concat(0x1e,0x1e,COUNT
    (*),0x1e,0x20) FROM "; nocase; http_uri; classtype:web-application-
    attack; threshold: type limit, track by_src, count 1, seconds 60; sid
    :999037; rev:1;)
```

Listing 45 – Règle Snort détaillée de détection de DarkMySQLi – Dump de la base.

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--full
    ' attack (1)"; flow:to_server,established; content:"concat(0x1e,0x1e,
    COUNT(*),0x1e,0x20)"; nocase; http_uri; content:" FROM
    information_schema.columns WHERE table_schema!=0x"; nocase; http_uri;
    classtype:web-application-attack; threshold: type limit, track by_src,
    count 1, seconds 60; sid:999038; rev:1;)
```

Listing 46 – Règle Snort détaillée de détection de DarkMySQLi – Récupération de toutes les informations possibles (1er payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--full
    ' attack (2)"; flow:to_server,established; content:"concat(0x1e,0x1e,
    table_schema,0x1e,table_name,0x1e,column_name,0x1e,0x20)"; nocase;
    http_uri; content:" FROM information_schema.columns WHERE table_schema
    !=0x"; nocase; http_uri; classtype:web-application-attack; threshold:
    type limit, track by_src, count 1, seconds 60; sid:999039; rev:1;)
```

Listing 47 – Règle Snort détaillée de détection de DarkMySQLi – Récupération de toutes les informations possibles (2eme payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--dbs
    --blind' attack (1)"; flow:to_server,established; content:" and ((
    SELECT COUNT(schema_name) FROM information_schema.schemata where
    schema_name != 0x"; nocase; http_uri; classtype:web-application-attack;
     threshold: type limit, track by_src, count 1, seconds 60; sid:999040;
    rev:1;)
```

Listing 48 – Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données, *blind SQLi* (1er payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--dbs
    --blind' attack (2)"; flow:to_server,established; content:" and ascii(
    substring((SELECT schema_name from information_schema.schemata where
    schema_name != 0x"; nocase; http_uri; classtype:web-application-attack;
     threshold: type limit, track by_src, count 1, seconds 60; sid:999041;
    rev:1;)
```

Listing 49 – Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données, *blind SQLi* (2eme payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--dbs'
     attack (1)"; flow:to_server,established; content:"concat(0x1e,0x1e,
    COUNT(*),0x1e,0x20)"; nocase; http_uri;  content:" FROM
    information_schema.schemata WHERE schema_name!=0x"; nocase; http_uri;
    classtype:web-application-attack; threshold: type limit, track by_src,
    count 1, seconds 60; sid:999042; rev:1;)
```

Listing 50 – Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données (1er payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--dbs'
     attack (2)"; flow:to_server,established; content:"concat(0x1e,0x1e,
    schema_name,0x1e,0x20)"; nocase; http_uri; content:" FROM
    information_schema.schemata WHERE schema_name!=0x"; nocase; http_uri;
    classtype:web-application-attack; threshold: type limit, track by_src,
    count 1, seconds 60; sid:999043; rev:1;)
```

Listing 51 – Règle Snort détaillée de détection de DarkMySQLi – Récupération des bases de données (2eme payload).

```
alert tcp any any -> $SERV_IP_TEST any (msg:"AUDIT_TOOLS DarkMySQLi '--info
    --blind' attack (1)"; flow:to_server,established; content:" and (
    SELECT 1 from mysql.user limit 0,1)=1"; nocase; http_uri; classtype:web
    -application-attack; threshold: type limit, track by_src, count 1,
    seconds 60; sid:999044; rev:1;)
```

Listing 52 – Règle Snort détaillée de détection de DarkMySQLi – Récupération des informations sur les bases de données, *blind SQLi*.

Fimap

Fimap est un outil d'audit présenté en page 40 de ce rapport. La règle simplifiée est détaillée en page 40.

Cette règle complète ajoute une référence, ainsi qu'un *tag* permettant de trier les alertes remontées par Snort, mais aussi un `threshold`, qui permet de ne lever une alerte qu'après un nombre donnée d'évènements reçus au cours d'une période de temps donnée.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS (msg:"AUDIT_TOOLS FImap Scan
    Detected"; flow:to_server,established; content:"|0d 0a|User-Agent\:
    fimap"; nocase; classtype:web-application-attack; reference:url,code.
    google.com/p/fimap/; threshold: type limit, track by_src, count 1,
    seconds 60; sid:999045; rev:1;)
```

Listing 53 – Règle Snort détaillée de détection de fimap.

SQLsus

SQLsus est un outil d'audit présenté en page 40 de ce rapport. La règle simplifiée est détaillée en page 40.

Cette règle complète ajoute une référence, ainsi qu'un *tag* permettant de trier les alertes remontées par Snort, mais aussi un `threshold`, qui permet de ne lever une alerte qu'après un nombre donnée d'évènements reçus au cours d'une période de temps donnée.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS (msg:"AUDIT_TOOLS SQLsus
    Attack Detected"; content:"UNION ALL SELECT BINARY"; nocase; http_uri;
    content:"CONCAT(0x5,1,0x7)"; nocase; http_uri; flow:to_server;
    classtype:web-application-attack; reference:url,toto; threshold: type
    limit, track by_src, count 1, seconds 60; sid:999046; rev:1;)
```

Listing 54 – Règle Snort détaillée de détection de sqlsus.

XSSF

XSSF est un outil d'audit présenté en page 41 de ce rapport. La règle simplifiée est détaillée en page 41.

Cette règle complète ajoute une référence à l'alerte permettant d'obtenir des informations sur celle-ci.

```
alert tcp any any -> $HOME_NET any (msg:"AUDIT_TOOLS XSSF payload detected"
    ; content:"function XSSF"; nocase; classtype:web-application-attack;
    reference:url,code.google.com/p/xssf/; sid:999013; rev:1;)
```

Listing 55 – Règle Snort détaillée de détection de XSSF.

BEEF

BEEF est un outil d'audit présenté en page 41 de ce rapport. La règle simplifiée est détaillée en page 41.

Cette règle complète ajoute une référence à l'alerte permettant d'obtenir des informations sur celle-ci.

```
alert tcp any any -> $HOME_NET any (msg:"AUDIT_TOOLS BEEF payload detected"
    ; content:"function beef"; nocase; classtype:web-application-attack;
    reference:url,beefproject.com/; sid:999014; rev:1;)
```

Listing 56 – Règle Snort détaillée de détection de BEEF.

A.2 Attaques Web génériques

Injections SQL

Les attaques de type injection SQL ont été présentées en page 47 de ce rapport. La règle simplifiée est détaillée en page 47.

Cette règle complète ajoute un *tag* permettant de trier les alertes remontées par Snort.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS SQL
    Injection."; content:"GET"; http_method; content:"="; pcre:"/=.*([\('\"
    ]?\w*[\)'\"]?\s*(((=|LIKE)\s*[\('\"]?\w*[\)'\"]?)|(IS\s*(NOT\s*|)NULL))
    |((=|LIKE|UNION(\s+ALL|)\s+|)SELECT.+(FROM|)))/i"; classtype:web-
    application-attack; sid:999010; rev:1;)
```

Listing 57 – Règle Snort détaillée de détection générique d'attaques de type *SQLi*.

Cross-Site Scripting

Les attaques de type XSS ont été présentées en page 47 de ce rapport. La règle simplifiée est détaillée en page 48.

Cette règle complète ajoute un *tag* permettant de trier les alertes remontées par Snort.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS XSS
    (1)"; content:"GET"; http_method; content:"="; pcre:"/<(script|img)(\s
    |\/)*(>|src=[\"' ]?)(javascript(:alert)?)?/i"; classtype:web-
    application-attack; sid:999011; rev:1;)
```

Listing 58 – Règle Snort détaillée de détection générique d'attaques de type *XSS*.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS XSS
    (2)"; content:"GET"; http_method; content:"="; pcre:"/(javascript:?)?
    alert\(/i"; classtype:web-application-attack; sid:999012; rev:1;)
```

Listing 59 – Règle Snort détaillée de détection générique d'attaques de type *XSS*. *Version raccourcie*

Local File Inclusion

Les attaques de type LFI ont été présentées en page 48 de ce rapport. La règle simplifiée est détaillée en page 48.

Cette règle complète ajoute un *tag* permettant de trier les alertes remontées par Snort.

```
alert tcp any any -> $SERV_IP_TEST $HTTP_PORTS_TEST (msg:"AUDIT_TOOLS Local
     File Inclusion or Path Traversal"; content:"GET"; nocase; pcre:"
    /(((\.|%(25|)(2e|c0%(25|)(ae|2e))){0,3}(\\|\/|%(25|)((5c|2f)|c0%(25|)(
    af|2f)|c1%(25|)9c))+)+)(etc(\\|\/|%(25|)((5c|2f)|c0%(25|)(af|2f)|c1
    %(25|)9c))passwd|C:\\|cmd\.exe)/i"; classtype:web-application-attack;
    sid:999009; rev:1;)
```

Listing 60 – Règle Snort détaillée de détection générique d'attaques de type *LFI*.

ANNEXE B

PARTIES SUR LESQUELLES SONT BASÉES LA DÉTECTION D'OUTILS D'AUDIT

Les outils d'audits analysés dans la partie 3.1.2 sont détectés en recherchant dans leurs traces réseau des motifs particuliers.

Ces motifs peuvent être de plusieurs types. Le *User-Agent*, un *payload* spécifique ou encore une attaque par brute-force. Dans la conclusion de cette partie, j'ai donné un graphique (que je reprend ici) qui affiche les statistiques de détection de ces outils. Les valeurs permettant d'établir ces statistiques sont données ici.

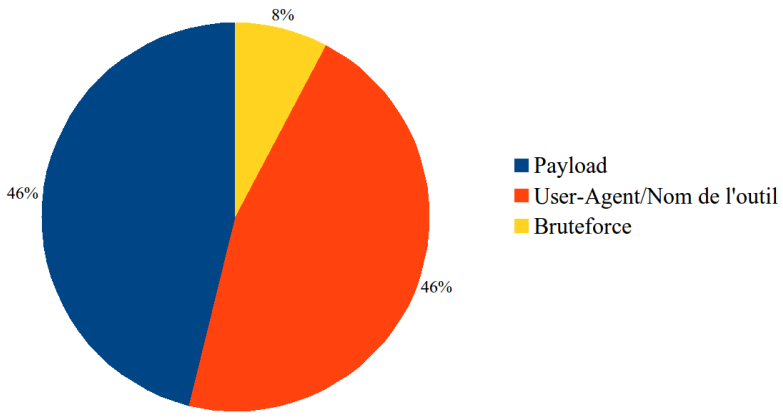


FIGURE 21 – Diagramme présentant des statistiques sur les parties des paquets utiles à la détection des outils d'audit.

Outil	**Emplacement**
Blind Elephant	*Payload*
WhatWeb	*User-Agent*
SkipFish	*User-Agent*
WebSploit – module `pma`	Brute-Force
WebSploit – module `apache_users`	*Payload*
WebSecurify	*User-Agent*
XSSer – 1	*Payload*
XSSer – 2	*Payload*
DarkMySQLi	*Payload*
FImap	*User-Agent*
sqlsus	*Payload*
XSSF	*Payload*
BEEF	*Payload*

TABLE B.1 – Détail des emplacement des motifs détectés pour chaque outil analysé.

Emplacement	**Nombre**	**Moyenne**
Payload	6	46%
User-Agent	6	46%
Brute-Force	1	8%
Total	13	100%

TABLE B.2 – Statistiques de l'emplacement des motifs détectés pour chaque outil analysé.

ANNEXE C

PAYLOADS FRÉQUEMMENT UTILISÉS LORS D'ATTAQUES WEB

Dans la partie 3.2.3, intitulée « Attaques Web génériques », j'ai présenté des attaques Web génériques, comme les LFI ou les attaques XSS. J'ai également présenté un outil d'audit automatique : Burp.

Afin de détecter ces attaques, j'ai écrit des expressions régulières qui correspondent aux motifs fréquemment utilisés lors de ces attaques.

Voici les listes de *payload*s que j'ai utilisé afin d'écrire mes expressions régulières.

C.1 Injections SQL

Une injection SQL, ou *SQLi* est une attaque informatique visant à modifier les requêtes SQL executées par l'application attaquée afin d'induire un nouveau comportmeent et récupérer des informations depuis le serveur. Pour cela, l'attaquant envoie à l'application vulnérable un *payload* spécifique permettant de lancer l'attaque.

La figure 22 présente graphiquement l'expression régulière de détection de ces motifs.

- `'`
- `;`
- `%25%27`
- `AND 1=2`
- `OR 1=1`
- `OR 1=2`
- `OR '1'='1'`
- `OR '1' LIKE '1'`
- `OR SELECT ... FROM`

- `OR 1 IS NOT NULL`
- `AND d = d AND`
- `AND .* = .* AND`
- `SELECT SLEEP( d)`
- `"X" = "X"`
- `"X" LIKE "X"`
- `"X" IS NOT NULL`
- `"X" IS NULL`
- `= SELECT ... FROM`
- `LIKE SELECT ... FROM`
- `UNION SELECT ... FROM`

C.2 Cross-Site Scripting

Une attaque de type XSS, ou *Cross-Site Scripting*, est une attaque Web visant à injecter dans une page Web du contenu afin de lancer des actions dans le navigateur des visiteurs de la page.

Les figures 23 et 24 présentent graphiquement les expressions régulières de détection de ces motifs.

- `alert(1)`
- `<script>alert(1)</script>`
- `<script src="`
- `<script src="javascript:alert`
- `<img src="javascript`
- `javascript:alert(`

C.3 Local File Inclusion

Une LFI, ou *Local File Inclusion*, est une attaque Web qui permet à un attaquant d'inclure dans la page Web vulnérable un fichier présent sur le serveur Web. Typiquement, ce fichier est le fichier `/etc/passwd` sur les systèmes UNIX.

La figure 25 présente graphiquement l'expression régulière de détection de ces motifs.

- `..\..\..\..\windows\win.ini`
- `../../../../windows/win.ini`

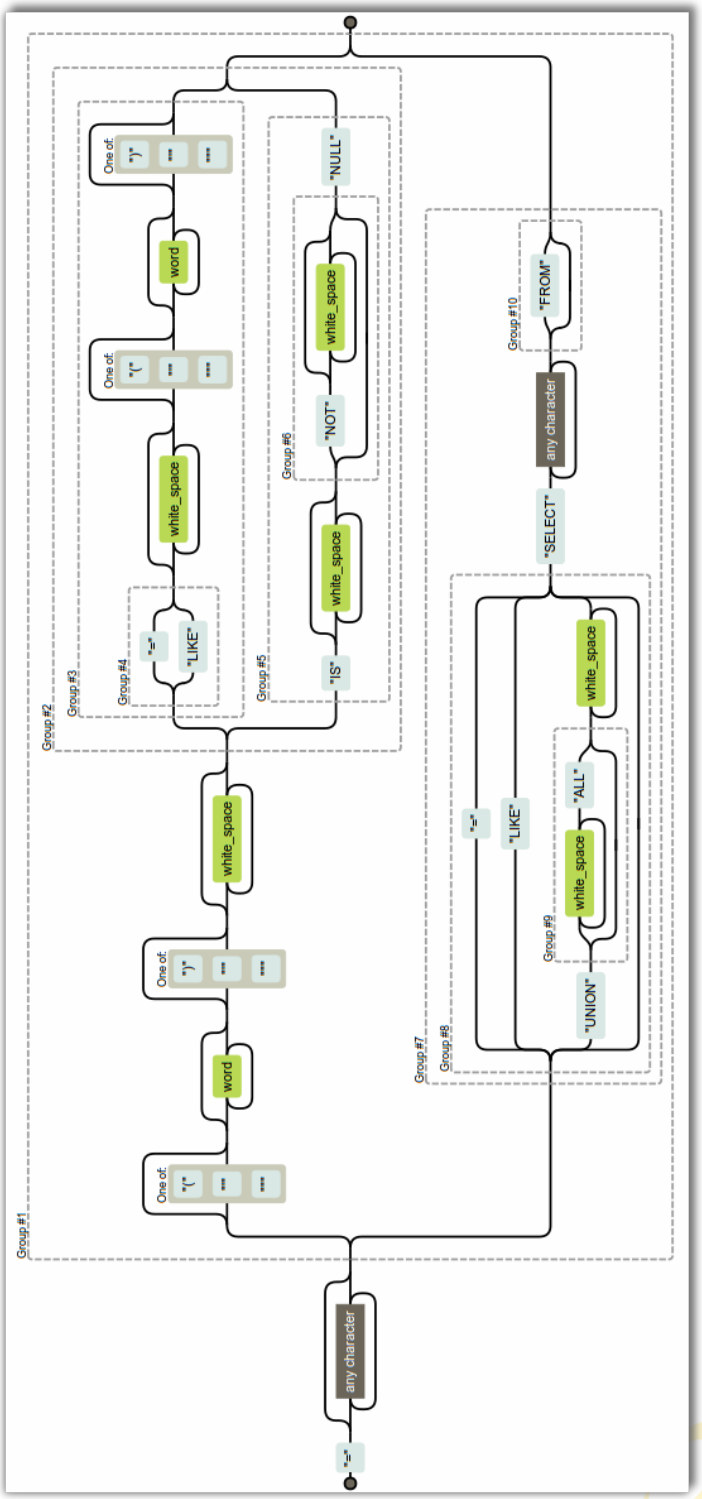


FIGURE 22 – Visualisation graphique de l'expression régulière de détection générale d'attaques de type injection SQL.

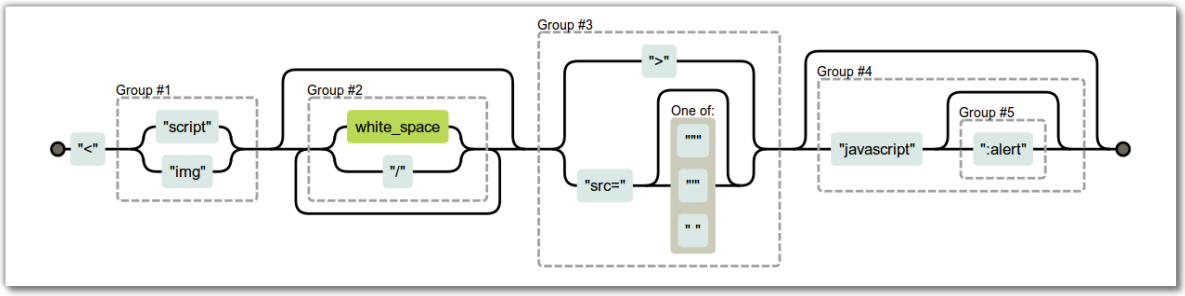


FIGURE 23 – Visualisation graphique de l'expression régulière de détection générale d'attaques de type XSS.

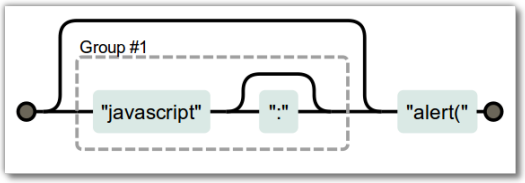


FIGURE 24 – Visualisation graphique de l'expression régulière de détection générale d'attaques de type XSS. *Version raccourcie*

- `../../../../etc/passwd`
- `./etc/passwd`
- `./cmd.exe`
- `./C:/`
- `.\cmd.exe`
- `.\C:\`
- `.\cmd.exe`
- `../etc/passwd`
- `../etc%2Fpasswd`
- `%2E%2E/etc/passwd`
- `../cmd.exe`
- `../C:/`
- `..\cmd.exe`
- `..\..\C:\`
- `..\cmd.exe`

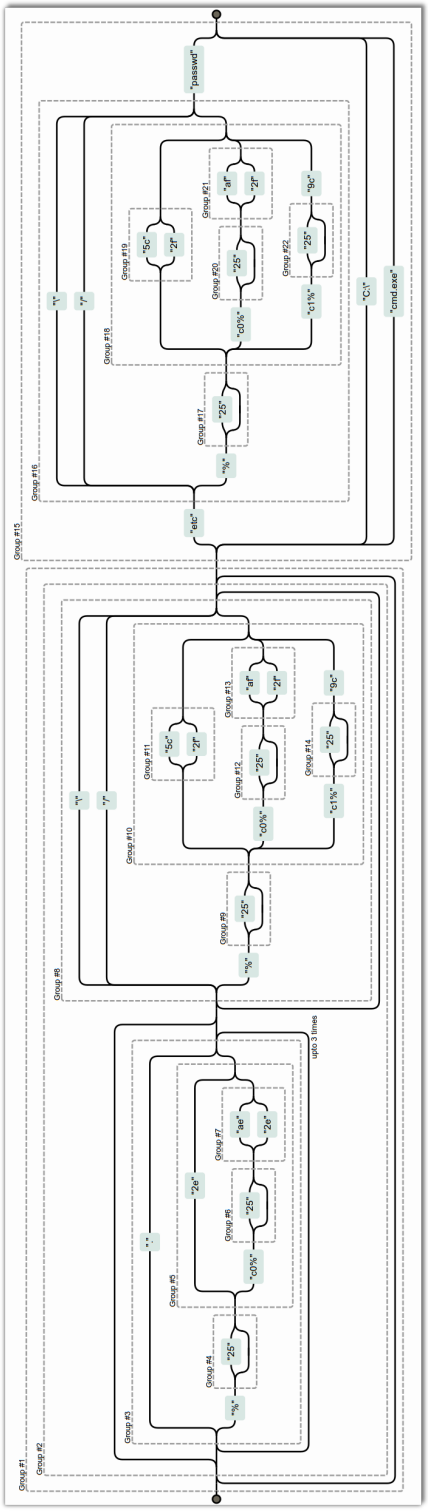


FIGURE 25 – Visualisation graphique de l'expression régulière de détection générale d'attaques de type LFI.

C.4 Burp *Scanner*

Burp est un *reverse proxy* qui est capable d'intercepter les requêtes envoyées aux serveurs Web audités, de les modifier, et surtout de réutilisant ces requêtes comme des « modèles », afin de lancer des *scans* de vulnérabilités sur ces applications, en utilisant des requêtes paraissant légitimes.

- `alert(document.cookie)`
- `benchmark(20000000%2csha1(1))`
- `%20and%20'6676'%3d'6676'`
- `%20and%20'6676'%3d'6677'`
- `%20and%20'3425'%3d'3426'`
- `waitfor%20delay'`
- `<script>alert(1)</script>`
- `(select%201)=1`
- `(select%201%2c2)`
- `..\windows\win.ini`
- `..\winnt\win.ini`
- `../etc/passwd`

ANNEXE D

ARCHITECTURE EN COUCHES DU MODÈLE TCP/IP

La figure 26 donne l'architecture des différentes couches du modèle TCP/IP, dont la définition complète est détaillée dans le glossaire.

Modèle TCP/IP			
	Type de donnée	**Couche**	**Fonction**
Couches Hautes	Donnée	5. Application	Transmission des données propres à l'application.
	Segment	4. Transport	Connexion bout-à-bout, contrôle des flux, connectabilité. Introduction de la notion de ports.
Couches Basses	Datagramme	3. Réseau	Détermine le parcours des données et l'adressage logique (adresses IP).
	Trame	2. Liaison	Adressage physique, et+ liaison point-à-point (adresses MAC).
	Bit	1. Physique	Transmission des données sous forme binaire.

FIGURE 26 – Architecture en couches du modèle TCP.

ANNEXE E

MODÈLE DE FICHE D'INCIDENT « BRUTE »

La levée de doutes (voir page 64) effectuée dans les activités d'un SOC nécessite de recueillir de nombreuses informations et surtout de les stocker.

Afin de réaliser ces investigations, j'ai écrit un modèle de fichier permettant de prendre ces notes de façon exhaustive afin d'écrire de façon efficace les fiches d'incidents liées aux investigations et aux intrusions en cours.

```
# {{Incident}} - ALA{{XX}}

{{criticité des event/alarmes OSSIM}}
{{criticité reelle}}
{{vulnerabilite exploitee}}

{{Periode d'apparition -- date debut/fin -- continu/ponctuel}}

## Events

- {{sid}} -- {{rule name}} ({{nb}} events a {{heure}} le {{date}})

## Source(s)

{{ip}}
{{ports}}

## Destination(s)

{{ip}}
{{ports}}

## Utilisateurs mis en cause

## Sondes / Branches
```

```
## Plan d'action

## Liens

## Exemple de Requete HTTP / Flux reseau
```

Listing 61 – Modèle de fichier permettant la prise d'informations concernant des intrusions.

ANNEXE F

CONFIGURATION PAR DÉFAUT OSSEC

J'ai présenté dans la partie 4.4.1, page 67 mes travaux afin d'écrire une configuration générique pour l'HIDS OSSEC sur des postes Windows.

Voici les règles que j'ai ajouté au fichier de configuration standard d'OSSEC.

```
<syscheck>
    <directories check_all="yes" realtime="yes">C:\Users\Administrator\
    Documents</directories>
    <!-- Stratup Programs -->
    <directories check_all="yes" realtime="yes">%SystemDrive%\ProgramData\
    Microsoft\Windows\Start Menu\Programs\Startup</directories>
    <!-- Installed Programs -->
    <windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
    CurrentVersion\Uninstall</windows_registry>
    <!-- Windows Update Servers -->
    <windows_registry>HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\
    Windows\WindowsUpdate\WUServer</windows_registry>
    <windows_registry>HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\
    Windows\WindowsUpdate\WUStatusServer</windows_registry>
    <!-- Last Windows Update -->
    <windows_registry>HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\
    CurrentVersion\WindowsUpdate\Auto Update\Results\Install\
    LastSuccessTime</windows_registry>
    <!-- DNS -->
    <directories check_all="yes">%WINDIR%\System32\dns</directories>
</syscheck>
```

Listing 62 – Règle à ajouter afin d'obtenir une configuration générique pour OSSEC