Thèse

présentée pour obtenir le grade de docteur

de l'Ecole nationale supérieure
des télécommunications

Spécialité : Informatique et Réseaux

Fabien Pouget

Système Distribué de Capteurs Pots de Miel: Discrimination et Analyse Corrélative des Processus d'Attaques

soutenue le 23 01 2005 devant le jury composé de

Pr. Ludovic ME	Président
Pr. John McHUGH	Rapporteurs
Dr. Hervé DEBAR	
Pr. Pascal URIEN	Examinateurs
Pr. Marc DACIER	Directeur de thèse

Ecole nationale supérieure des télécommunications

PhD thesis

Ecole nationale supérieure des télécommunications

Communications and Electronics department

Computer Science group

Fabien Pouget

Distributed System of Honeypot Sensors: Discrimination and Correlative Analysis of Attack Processes

Defense date: 01, 23 2005

Committee in charge:

Pr. Ludovic ME	Chairman
Pr. John McHUGH	Reporters
Dr. Hervé DEBAR	
Pr. Pascal URIEN	Examiners
Pr. Marc DACIER	Advisor

Ecole nationale supérieure des télécommunications

Je dédie ce travail aux personnes qui en ont le plus souffert : ma famille et Noémie...
sans rancune? :)

Remerciements

L'aboutissement de travaux, que ce soit dans le monde du bâtiment ou de la recherche, n'est jamais l'oeuvre d'une seule et unique personne. C'est le résultat d'un ensemble favorable de facteurs concomitants.

Il en va de même pour cette thèse, effectuée à l'Institut Eurécom, qui n'aurait jamais pu aboutir sans la collaboration avec le Professeur Marc Dacier. Je lui dois bien plus qu'il ne serait possible d'écrire en une page de remerciements. Ses conseils, son attention, sa curiosité et son impressionnante lucidité sont autant de facteurs qui m'ont été très favorables, tant sur le plan professionnel que personnel. Je n'ai jamais eu à regretter cette aventure académique, bien au contraire. Un tout grand merci !

Merci également aux membres du jury de cette thèse qui ont pris soin de lire le document et qui, par leurs conseils avisés, ont permis son aboutissement. Je remercie tout particulièrement Monsieur John McHugh, Professeur à l'Université Dalhousie au Canada, ainsi que Monsieur Hervé Debar, travaillant au sein du département Sécurité des Services et des Réseaux (SSR) de France Télécom R&D à Caen, qui ont accepté de juger ce travail et d'en être les rapporteurs. Je tiens également à adresser mes remerciements à Monsieur Ludovic Mé, Professeur à Supelec Rennes, et à Monsieur Pascal Urien, Professeur à Télécom Paris (ENST), pour leur participation au jury de cette thèse.

Je suis fier dans tous les cas du projet *Leurré.com*, et des différents contacts que celui-ci m'a permis de créer. J'ai eu le privilège et l'honneur de côtoyer des personnes de grandes valeurs : merci à Hervé Debar pour ses différents soutiens au projet, merci à George Mohay et Andrew Clark, de m'avoir fait découvrir les richesses australiennes, merci à Mohammed Kaaniche, Vincent Nicomette et Eric Alata pour toutes les discussions que nous avons pu avoir à Toulouse, et un grand merci à tous les partenaires du projet, qui ont, grâce à leur confiance, permis de construire un tel outil. Je remercie également toutes les personnes ayant contribuées à mes travaux, sous forme de publications ou de discussions, et en particulier Guillaume Urvoy-Keller, qui a passé beaucoup de son temps à me supporter. Ses multiples apparitions dans le bureau C022 ont ensoleillé nombre de mes journées.

Parmi les facteurs favorables, il faut aussi noter l'environnement d'Eurécom, cadre propice abritant une multitude de personnes charmantes et compétentes, sans oublier la vue plongeante sur la mer Méditerranée. Je tiens à saluer la sympathie du personnel

administratif et technique, que je n'ai eu cesse d'harceler. A noter la sérénité (sûrement apparente) du service informatique, vis-à-vis de mes requêtes incessantes, mes ennuis et mes besoins. Patrick Petitmengin aura particulièrement souffert de ma présence à Eurécom, et je lui suis reconnaissant de sa patience et de son sérieux. Je remercie également Gwenaelle le Stir, qui a eu l'amabilité de se charger des nombreuses démarches relatives à cette thèse.

A tous, amis, collègues, merci.

Il y a ceux qui sont déjà partis, Anwar Alhamra, Laurent Bussard, Raphael Chand. Il y a ceux qui restent et qui ont guidé cette fin de thèse, Walid Bagga, Jérome Haerri, Matti Siekkinen et Melek Onen. Il y a enfin ceux qui ne sont pas directement en relation avec Eurécom et ces travaux. Ils sont résidents de la Côte d'Azur, comme Karine et Stephan (quelles belles soirées Trivial Poursuite et Brainstorm!) et la petite famille Courtel: Gérard, Sophie et Chantal. D'autres sont un peu plus éloignés, mais je garde toujours pour eux une immense estime: Laurent Perpète, Cedric Lochon, Francois Pitie, Sebastian Hirschler, Benoit Huet, Francois Ferrand, Nicolas Kiefer. Et à tous ceux que j'oublie, pardonnez-moi. L'âge commence déjà à peser... :)

Et puis, j'adresse une dédicace particulière à mes parents, à ma grand-mère et à ma soeur, pour m'avoir en permanence soutenu, même si mes choix n'ont jamais été simples. De la même façon qu'un arc-en-ciel, je leur en ai fait voir de toutes les couleurs, mais leur soutien et leur confiance ont été imperturbables tout au long de ces années d'études. Sans oublier pour autant le reste de la famille qui s'est très agréablement élargie ces dernières années.

Enfin Noémie, toi qui partage ma vie. Que dire que tu ne saches déjà? Cette thèse est à toi, pour toi, et avec toi... Je ne pourrai pas finir non plus cette page de remerciements sans laisser une empreinte de mon humour, s'il n'est glorieux, du moins populaire. Donc Noémie, voici une phrase que je te souffle: "You are such a honey!.... from your favorite honey buzzard."

Merci à tous, et bonne lecture!

Résumé

Il est difficilement concevable de construire les systèmes de sécurité sans avoir une bonne connaissance préalable des activités malveillantes pouvant survenir dans le réseau, ni une bonne compréhension des processus d'attaques. Malheureusement, il apparaît que ce savoir n'est pas aisément disponible, ou du moins il reste anecdotique et souvent biaisé par des suppositions injustifiées, des sources d'information partiales ou des bruits de couloir.

Cette thèse a pour objectif principal de faire progresser l'acquisition de ce savoir sur les activités malveillantes par une solide méthodologie.

Dans un premier temps, il convient de travailler sur un ensemble intéressant de données. Malheureusement, les données sont rarement publiques, ou alors, elles mélangent à la fois du trafic normal dit de *production* et du trafic malveillant, comme par exemple les échantillons fournis par la métrologie des réseaux. Dans cette situation, il est difficile d'établir un distinguo entre les deux formes de trafic; ce problème est au cœur des soucis de la communauté de recherche travaillant sur la détection d'intrusions, et ce depuis plusieurs années. Pour contourner ceci, nous avons déployé un réseau distribué de sondes, aussi appelées *pots de miel*, à travers le monde. Les pots de miel sont des machines sans activité particulière, ce qui implique que toute connexion les ciblant est potentiellement malveillante. Ce réseau de sondes nous a donc permis de capturer un volume important de données suspectes sur plusieurs mois. Il est important de noter que cette architecture particulière nous fournit une surveillance très locale de ce genre de trafic.

Dans le cadre de cette thèse, nous présentons une méthodologie appelée *HoRaSis* (pour Honeypot tRaffic analySis), qui a pour but d'extraire automatiquement des informations originales et intéressantes à partir de cet ensemble remarquable de données. Elle est formée de deux étapes distinctes: *i)* la discrimination puis *ii)* l'analyse corrélative du trafic collecté. Plus précisément, nous discriminons d'abord les activités observées qui partagent une *empreinte* similaire sur les sondes. Cette étape doit tenir également compte des diverses influences du réseau. La solution proposée s'appuie sur des techniques de classification et de regroupement. Puis, dans une seconde phase, nous cherchons à identifier les précédentes empreintes qui manifestent des caractéristiques communes. Ceci est effectué sur les bases d'une technique de graphes et de recherche de cliques. De multiples exemples illustrent les intérêts respectifs de ces deux phases.

Plus qu'une technique, l'approche *HoRaSis* que nous proposons témoigne de la richesse des informations pouvant être récupérées à partir de cette vision originale du trafic malicieux de l'Internet. Elle montre également la nécessité d'une analyse rigoureuse et ordonnée du trafic pour parvenir à l'obtention de cette base de connaissances susmentionnée.

Abstract

Security systems cannot be efficiently designed without i) a good preliminary understanding of malicious activities which might occur in the wild and ii) a good comprehension of attack processes. Unfortunately, it seems that this knowledge is either not available or remains anecdotal and often biased by unclear assumptions, partial information sources and rumors.

The goal of this thesis is primarily to better understand the malicious activities that occur and to provide a methodology that would help to acquire this knowledge. It is necessary in a first step to work on a valuable dataset. However, public data is not easily available, or it frequently mixes production and malicious traffic, like with network measurement datasets. In this scenario, the distinction between production and malicious traffic is a complex problem that has occupied the Intrusion Detection community for several years. To address this issue, we have deployed a worldwide distributed network of sensors, also called *Honeypots*. Honeypots are machines that are not publicly advertised. Hence, any connection targeting such a machine is potentially malicious. This network of sensors has thus contributed to capture a huge amount of suspicious data over several months. In addition, this particular sensor architecture enables us to obtain a local monitoring of malicious traffic.

In the scope of this thesis, we propose a framework, called *HoRaSis* (for Honeypot tRaffic analySis), which aims at automatically extracting meaningful information out of this remarkable dataset. It basically consists in two major stages: *i)* the discrimination and *ii)* the correlative analysis of the collected traffic. More precisely, we first discriminate collected activities according to the fingerprints they let on each sensor. This stage must also consider the potential disturbances introduced by the network. The proposed solution relies on dedicated clustering and classification techniques. We then identify all previous fingerprints which share strong common characteristics. This task is performed thanks to a graph-theory approach, and, in particular, thanks to the search of maximal weighted cliques within graphs. Different characteristics based on our preliminary experiments have been considered. Several cases exemplify the value of combining these two stages.

Thanks to the proposed *HoRaSis* framework, we show that a rigorous and methodical analysis of honeypot traffic clearly helps to get a better understanding of malicious activities.

Contents

List of Figures

List of Tables

Acronyms

We group in this section most of the acronyms or notations which have been used in the different chapters of this thesis.

$CERT$	Computer Emergency Response Team
CR	Compression Ratio (SAX method)
$CSIRT$	Computer Security Incident Response Team
DNS	Domain Name System
DFT	Discrete Fourier Transform
DoS	Denial of Service
DS	Dominant Set
DWT	Discrete Wavelet Transform
FHP	French Honeynet Project
$FIRST$	Forum of Incident Response and Security Teams
GCI	Global Consistency Index
$HoRaSis$	Honeypot Traffic Analysis framework
$ICMP$	Internet Control Message Protocol
IDS	Intrusion Detection System
IGR	Information Gain Ratio
IP	Internet Protocol
$IPID$	IP Identifier
IPS	Intrusion Prevention System
IRC	Internet Relay Chat
$ISAC$	Information Sharing and Analysis Center
NAT	Network Address Translation
PAA	Piecewise Aggregate Approximation
RCA	Root-Cause Analysis
SAX	Symbolic Aggregate Approximation
SR	Splitting Ratio
TLD	Top-Level Domain
TCP	Transmission Control Protocol
$WARP$	Warning, Advice and Reporting Point

Synthèse en français

Introduction

La sécurité est le souci d'un grand nombre de domaines d'activité. Internet a la particularité de connecter les gens de façon plus ou moins anonyme, et sans grand contrôle du trafic. Cet atout, qui fait le succès de la toile, présente aussi des inconvénients majeurs : des activités malveillantes peuvent prendre aisément une grande amplitude et produire des catastrophes. A valeur illustrative, l'équipe américaine de Staniford montre dans [215] qu'il est possible, en théorie, pour un ver, de saturer un million de machines vulnérables en l'espace de 510 milli-secondes. Dans la même idée, il faut noter la recrudescence des fraudes électroniques, qui peuvent se chiffrer à plusieurs millions de dollars par an. Internet facilite le banditisme et les crimes à grande échelle. Il semble alors très important, si ce n'est vital, d'acquérir de solides connaissances sur les menaces et les stratégies d'attaques. Une méthode pour obtenir ce savoir réside dans l'observation et l'analyse à grande échelle d'activités malveillantes.

Plusieurs techniques existent actuellement, dont certaines appartiennent à la catégorie nommée *pot de miel* (ou *honeypot* en anglais). Ce terme est récent, quand bien même le concept existe depuis de nombreuses années. Dans les années 1980, Clifford Stoll a eu l'idée de placer des données en apparence confidentielles afin de tromper et mettre en évidence les voleurs. L'idée a été reprise sous le terme anglais *honeytoken* par Lance Spitzner dans [214]. Ce dernier a proposé dans ce même ouvrage la définition suivante d'un *pot de miel* :

Un **pot de miel** est une partie ou l'ensemble d'un système d'information dont la valeur ajoutée est d'être compromise ou utilisée de manière illicite.

Nous garderons cette définition tout au long de la thèse.

Le grand avantage de ces *pots de miel* réside dans leur capacité à collecter du trafic suspect uniquement. Depuis plusieurs années, ces traces particulières se mélangeaient avec celles dites de production, ce qui n'aidait pas les personnes en charge de la sécurité à déterminer les activités malveillantes. Ce problème est maintenant révolu grâce aux *pots de miel*. Ces trois dernières années, un effort certain a été effectué par diverses communautés pour construire des architectures *pots de miel* sûres et utiles, i.e. des systèmes capables de récupérer de l'information, allant de simples paquets de balayage de ports à une communication IRC complète, sans mettre en danger le réseau hébergeur. Les solutions sont donc nombreuses, et les technologies de type *pots de miel* sont largement

utilisées, présentant un intérêt aussi bien pour les grands groupes antivirus que pour les organisations internationales et gouvernementales, telles les CSIRTs, l'ENISA (European Network and Information Security Agency ou les centres d'analyse ISAC (Information Sharing and Analysis Center).

Malheureusement, nous constatons que très peu d'efforts sont faits pour partager les informations collectées au moyen des *pots de miel*. *A contrario*, des données publiques existent, grâce à des initiatives comme Dhshield, MyNetWatchman ou le Computer Network Defense Operational Picture [34]. Ces projets présentent sur des pages Internet attractives des statistiques, mais la source de ces informations n'est pas toujours claire. Ils invitent tout un chacun à envoyer les archives de pare-feux ou de systèmes de détection d'intrusions (IDS) pour extraire des valeurs statistiques relativement simples. L'information est ainsi limitée (comptage par port), et les chiffres intègrent le biais lié au trafic de production.

En résumé, les *pots de miel* sont une source d'information de grande valeur. Cependant, comme il a était brièvement mentionné ci-dessus, le plus gros effort est fait pour optimiser leur architecture, et peu d'initiatives ont émergé pour organiser et tirer tous les bénéfices de la richesse des données qu'ils fournissent.

En parallèle, les solutions d'analyse existantes se limitent fréquemment à résoudre un problème en particulier. Il peut s'agir de techniques pour surveiller les attaques par déni de service, les balayages de ports, ou certains scénarios d'attaques bien précis. La plupart de l'existant a d'ailleurs été développé par les personnes de la communauté de la Détection d'Intrusion. Cependant, l'approche est sensiblement différente avec les *pots de miel*, comme tout le trafic capturé reste par définition suspect. Les faux-positifs ne sont donc plus le souci principal, comme cela est encore le cas pour la majorité des techniques de détection d'intrusions. Ceci nous amène à établir le constat suivant :

Constat : Les *pots de miel* sont largement déployés, et ils sont techniquement matures. Les techniques d'analyse, en revanche, sont mal adaptées pour profiter de la qualité de l'information offerte.

Sur la base de ce constat, nous avons décidé de construire notre propre environnement de type *pot de miel*, dans le but de collecter d'indispensables données ; la motivation première étant de travailler sur des données de trafic malveillant accessibles et utiles pour l'analyse. Ceci a été fait dans le cadre d'un projet nommé *Leurré.com*, qui regroupe des partenaires de nombreux pays. Grâce à cette communauté, nous avons réussi à collecter un volume de données considérable à partir de plusieurs environnements *pots de miel*. Il est important de comprendre que cet ensemble de données est unique, et qu'il est accessible pour chaque partenaire. Nous ne connaissons pas d'équivalent pour le moment. L'hypothèse de notre problème est donc la suivante :

Hypothèse : Nous travaillons sur un ensemble de données unique, constitué d'activités malveillantes observées dans différents endroits du globe et dans des réseaux très divers.

Que pouvons-nous faire avec de telles données ? Nombreuses sont les techniques

d'analyse de trafic : il existe les outils traditionnels comme netflow ([75]) ou tcpdump ([17]), ou des méthodes théoriques plus complexes. En revanche, aucune n'est spécifique au type de données fournies par les pots de miel. Plus important encore, rien de très constructif n'a été proposé, jusqu'à présent, pour échanger aisément de l'information à partir de ces analyses. Il est légitime à ce stade de se demander si ce nouvel ensemble de données apporte de l'information nouvelle et originale. Si la réponse devait être positive, serait-il possible de l'extraire automatiquement ? Pour simplifier, nous cherchons donc à savoir s'il existe une méthode pour faire cela. Si elle existe, nous la nommerons *HoRaSis* (pour *Ho*neypot t*Ra*ffic analy*Sis*), comme étant une base pour l'analyse de trafic des pots de miel. La thèse présentée dans ce document se résume ainsi :

Positionnement de la thèse : Nous voulons montrer dans cette thèse que

1. un réseau distribué de simples sondes pots de miel fournit des données intéressantes pour l'analyse et la compréhension des menaces et stratégies d'attaques.

2. il existe une méthode automatique pour extraire de l'information intéressante à partir de ces données. Celle-ci sera nommée *HoRaSis* (pour *Ho*neypot t*Ra*ffic analy*Sis*).

Les pots de miel permettent de recueillir des données très singulières, qui peuvent nécessiter une technique d'analyse dédiée. Cette remarque sera plus amplement justifiée dans les premiers chapitres de ce document, par les expérimentations préliminaires prometteuses des données. A partir de notre savoir-faire construit au fil des données, il est apparu comme vital de créer une méthode (appelée *Honeypot Traffic Analysis* ou *HoRaSis*) afin de rendre mécanique l'extraction d'information à partir des données collectées.

L'analyse de traces issues des pots de miel est à la jonction de plusieurs espaces de recherche, et la méthode *HoRaSis* que nous cherchons ne peut prétendre les surclasser tous. Par voie de conséquence, la méthode doit être ouverte à de futures améliorations, en offrant une structure modulaire. De manière plus générale, nous listons ci-dessous les critères que la méthode *HoRaSis* doit préserver :

- *Validité :* Un ensemble d'analyses a été effectué de façon empirique, en tirant peu à peu le fil d'Ariane. Cette tâche, bien que peu efficace, a fourni des résultats préliminaires prometteurs. La méthode automatique que nous cherchons ne doit pas contredire ces expériences, et *a contrario* devrait enrichir les observations, comme le critère suivant indique.

- *Découverte d'information :* La méthode *HoRaSis* doit être une nouvelle source de connaissances.

- *Modularité :* La méthode *HoRaSis* se trouve à la croisée de plusieurs domaines de recherche. Nous pouvons d'ores et déjà citer ceux des Réseaux, de la Sécurité et de l'Analyse de Données. Il existe aussi une multitude de sous-domaines, prenant diverses directions théoriques et techniques. De nouveaux apparaissent régulièrement, et il est capital que la méthode présente une structure modulaire afin de pouvoir bénéficier des dernières avancées dans chacun de ces domaines.

- *Généralisation :* Les données collectées peuvent changer de manière drastique selon l'apparition de nouvelles activités et de nouveaux procédés d'attaque. La méthode *HoRaSis* doit donc être suffisamment indépendante des données, ou du moins, être adaptable à des ensembles de données aux caractéristiques très différentes.

- *Simplicité :* La méthode *HoRaSis* doit extraire de l'information à partir d'un ensemble de données (dans notre cas, fourni par le projet *Leurré.com*). Le destinataire de cette information est l'analyste; ce dernier doit comprendre le cheminement qui a conduit à l'extraction de ces nouvelles connaissances. La méthode ne doit pas se présenter comme une boîte noire aux résultats obscurs.

Nous prouvons dans ce rapport qu'une telle méthode existe, et qu'elle nous permet de trouver des résultats prometteurs sur les activités malveillantes observables. *HoRaSis* est un moyen automatique de valider (ou de rejeter) nombre de suppositions.

Les contributions de cette thèse sont :

- Le déploiement et administration d'un système distribué de pots de miel pour collecter des données.

- La conception d'une méthode appelée *HoRaSis* pour analyser les données.

- La création de nouvelles techniques pour tirer profit des propriétés de données issues de pots de miel.

- La validation de la méthode *HoRaSis* grâce aux analyses préliminaires effectuées.

- L'amélioration de la compréhension des activités observées. Certaines de ces activités ont pu être clairement identifiées, les autres sont de nouvelles questions offertes à la communauté Sécurité.

Ainsi, la thèse peut se résumer au schéma suivant :

- *HYPOTHÈSES :* Une architecture de pots de miel déployée pour collecter des données.

- *DONNÉES INITIALES :* Un grand volume de traces réseaux, chacune étant malveillante, ou du moins suspecte.

- *PROBLÈME :* Est-ce une nouvelle source d'information d'intérêt ? Si tel est le cas, comment bâtir une solide méthode analytique à partir ce celle-ci ?

D'une manière concrète, *HoRaSis* est une méthode articulée autour de quelques étapes majeures, qui sont symbolisées sur la figure 1. Les étapes 1 et 2 concernent le déploiement et la collecte de données à partir de pots de miel. Ce travail a été rendu possible par l'intermédiaire du projet académique appelé *Leurré.com*. Afin de faciliter la compréhension d'*HoRaSis* et des problématiques existantes, nous décrivons ce projet dans le chapitre 3, qui complète le chapitre 2 dédié à l'état de l'art. Le lecteur trouvera les détails des

étapes 3 et 4 dans les chapitres respectifs 4 et 5. L'étape 3 consiste à grouper les activités présentant des caractéristiques identiques, ou, en d'autres termes, toutes les adresses IPs ayant laissé une empreinte *équivalente* sur les différentes sondes pots de miel. Dans l'étape 4, nous analysons les relations émergentes qui peuvent apparaître suite à ce premier groupement. Toutes les empreintes observées sur les sondes pots de miel qui partagent de mêmes singularités sont détectées puis analysées. Le chapitre 6 décrit l'information obtenue suite à l'application d'*HoRaSis*, celle-ci étant alors exploitable et partageable au sein de la communauté Sécurité. Chaque chapitre reprend des résultats obtenus à partir de l'ensemble de données *Leurré.com*.

Les étapes de la méthode *HoRaSis* se trouvent sur la figure 1, ainsi que le numéro des chapitres correspondants. Elles sont brièvement résumées dans les paragraphes qui suivent en français.

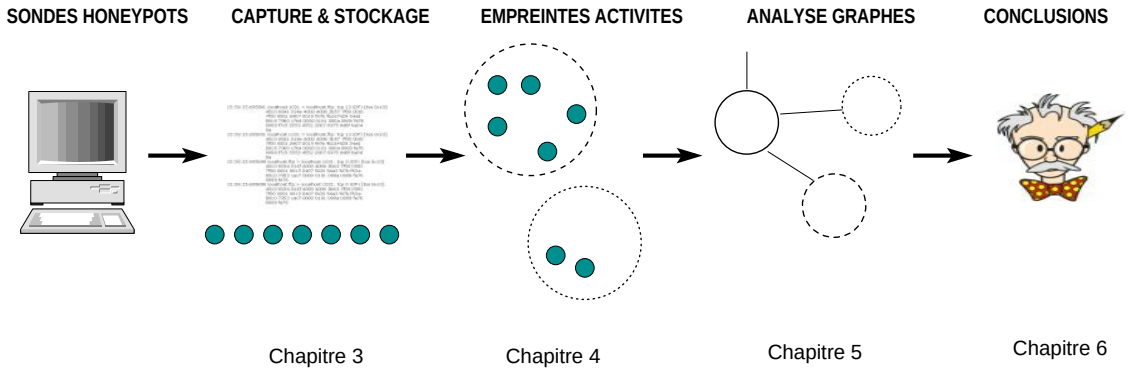


Figure 1: Méthode d'analyse via un réseau distribué de pots de miel

Motivations et terminologie d'usage

Introduction de cet état de l'art

La compréhension des activités malveillantes est un pré requis à l'élaboration d'une technique défensive efficace. Par analogie, construire un mur ne protègera pas d'une agression aérienne. Ainsi, connaître la menace ne peut être un facteur à ignorer, particulièrement quand les vendeurs sont friands de technologies aguichantes, mais qui peuvent s'avérer inutiles, ou tout simplement inadaptées.

HoRaSis est la méthode que nous cherchons pour analyser des données issues des pots de miel, afin de bénéficier de leurs propriétés intrinsèques. De nombreux projets ont récemment émergé pour capturer des traces malveillantes. En parallèle, une multitude de solutions provenant de divers axes de recherche sont apparues pour effectuer des analyses sur des données afin d'en identifier des activités anormales. Pour ces raisons, et pour garder une certaine clarté dans l'état de l'art, nous choisissons de distinguer de catégories, discutées par la suite :

- Les techniques dont la finalité est d'offrir la possibilité de capturer du trafic malveillant.

- Les techniques dont la finalité est d'extraire de l'information à partir d'un trafic donné.

Capturer des traces originales

Il existe plusieurs projets récents qui ont pour finalité de bâtir des environnements de capture efficaces, souvent dans l'idée de capturer de nouvelles activités malveillantes (les activités malveillantes qui n'ont pas encore été observées sont appelées *0-jour*). Nous détaillons en particulier dans le document original :

- Les outils en logiciel libre associés aux termes anglais *honeypots, honeynet, honeytokens.*
- Le télescope réseau (ou *Network Telescope* développé par CAIDA (Cooperative Association for Internet Data Analysis), qui consiste à l'observation au niveau d'un équipement réseau d'une très large plage (préfixe /8 par exemple) d'adresses IP non utilisées (ou très peu utilisées).
- Le projet *Darknet* de Team Cymru, proche dans l'esprit du projet précédent. Leur site offre quelques graphes représentant une estimation quantitative du *bruit de fond* (ou *background radiation* observé.
- iSink de l'université de Wisconsin-Madison, qui était d'avantage un outil d'analyse de performance réseau à l'origine.
- IMS (ou *Internet Motion Sensor* proposé par l'Université Michigan, qui propose l'utilisation de sondes. L'information, par contre, est extraite de chaque sonde, sans analyse corrélative entre les informations trouvées.
- MINOS de l'université UC Davis, dont le principe fondamental est de marquer le trafic suspect afin de pouvoir le suivre plus aisément.
- Lobster (anciennement SCAMPI), projet européen cherchant à faciliter la surveillance des réseaux au niveau matériel.
- Mwcollect, outil très récent, ayant fusionné avec un autre projet nommé Nepenthes, dont l'objectif consiste à capturer des activités malveillantes cherchant à exploiter des vulnérabilités bien précises (DCOM, Local Security Authority Service LSASS, NetBIOS, SQL Server, etc).
- Le partage d'archivage, proposé par de nombreux sites, tels WormRadar, Internet Storm Center de l'Institut SANS, Dshield, MyNetWatchman, etc. Les résultats se basent malheureusement sur des données incertaines à la source.

Techniques d'analyse

Parmi les techniques ayant vocation à analyser le trafic brut collecté par des méthodes comme celles précédemment citées, le document détaille :

- NetFlow, le format d'agrégation en flux utilisé dans des appareils de type routeurs pour limiter le volume de données stocké. Quelques analyses s'appuient sur ces flux, bien qu'ils présentent des limitations : un flux NetFlow n'a pas d'équivalent clair au niveau protocolaire (TCP), et se limite à un ensemble restrictif de champs.

- Le projet Billy Goat proposé par Duponchel et al. d'IBM, où un effort est fait pour archiver les données collectées de manière pratique. L'extraction d'information reste cependant limité à ce stade.

- honeyStat, ainsi que d'autres techniques issues du monde de la détection d'intrusion (projet Collapsar de l'université de Purdue). Ces techniques se résument souvent à une innovation théorique testée dans des conditions particulières. L'information extraite reste donc d'autant limitée, même si ces techniques peuvent s'appliquer dans des analyses bien précises.

- Les consoles de surveillance sont nombreuses et variées. Une analyse que nous avons faite montre l'étendue des solutions existantes. Nous sommes malheureusement arrivés à la conclusion que la plupart se limitent à des techniques pragmatiques simples, comme des expressions régulières ou des requêtes SQL dans une base de données à la structure assez standard.

- La modélisation est un sujet actif de recherche. Le manque de données librement utilisable empêche néanmoins la validation des modèles proposés. Ceux-ci se limitent par ailleurs à quelques stratégies de propagation de vers connus *a priori*.

Conclusions concernant l'état de l'art

Certaines techniques de capture et d'analyses sont prometteuses. Cependant, elles restent cloisonnées et s'adaptent mal au contexte des pots de miel. Ainsi, les nouvelles solutions pour capturer du trafic via un pot de miel ne bénéficient pas vraiment de méthodes d'analyse propres et efficaces. C'est ici la contribution de cette thèse, qui constitue à apporter un élément de réponse à ce problème.

Projet *Leurré.com*

Brève introduction au projet

Au sein de l'Institut Eurécom (www.eurecom.fr), nous avons utilisé la technologie des pots de miel afin d'arriver à une meilleure compréhension des processus d'attaques. Nous avons implémenté une plateforme de test qui a été ensuite installée dans un réseau comprenant actuellement une quarantaine de partenaires provenant des cinq continents. Les données collectées depuis deux ans sont enrichies puis étudiées au moyen de techniques diverses et variées, qui sont détaillées dans les trois derniers chapitres du document (analyse en séries temporelles, techniques de regroupement, règles associatives, graphes).

Le choix de la plateforme s'est porté sur un système ayant une interaction faible (*honeyd*), afin de limiter les risques de compromission. Cette plateforme émule trois

machines différentes (Windows NT Server, WIndows 98 et Linux Red Hat 7.3), avec les ports de l'installation par défaut ouverts, ainsi que quelques scripts correspondant à des services choisis (serveurs ftp et web par exemple). Une comparaison des données collectées est par ailleurs maintenue avec un système plus complexe (i.e. des services réels, non émulés) correspondant à une configuration équivalente. Celle-ci a pour but de vérifier qu'aucun biais n'est introduit par l'utilisation de ce système à faible interaction. Il est important de comprendre les limitations de la capture avant une quelconque analyse.

Leurré.com est un projet ouvert à tout partenaire curieux et désireux de mieux comprendre l'activité malveillante ciblant ses ressources. Il lui suffit pour cela d'installer une plateforme pot de miel décrite ci-dessus à l'extérieur de son réseau. L'installation et la maintenance sont totalement prises en charge par Eurécom et ne nécessitent pas d'investissement particulier : un simple ordinateur et quatre adresses IP routables (une pour la machine d'accueil, et trois pour les machines émulées par le pot de miel) sont suffisants pour sa mise en place. En contrepartie, Eurécom offre l'accès à l'analyse des informations collectées et étudiées par le groupe de recherche sur les attaques de la plateforme partenaire. Nous proposons une interface intégrant des résultats simplifiés répondant à des requêtes fréquentes, ou un accès direct aux données par le moyen d'une base intégrant différents degrés d'information. Un rapport d'activité personnalisé de la plateforme est également émis sur demande pour chaque partenaire.

Archivage des données

Nous récupérons chaque jour les traces réseau (format tcpdump) sur les plateformes, correspondant au trafic échangé entre les machines virtuelles et d'autres machines de l'Internet. Elle contient actuellement des données à partir de février 2003, et le nombre de partenaires ne cesse de croître. Pour stocker un si gros volume de données, nous avons construit la base de données dans l'idée de pouvoir :

- chercher tout type d'information rapidement, que ce soit de l'information générale ou pointue (champs protocolaires).

- ajouter rapidement une nouvelle source d'analyse, en relation avec les informations déjà stockées.

.

Sans rentrer dans les détails de l'architecture, nous avons décidé de la bâtir autour de quatre définitions, décrites ci-dessous :

Source : Une *Source* correspond à une adresse IP observée sur une ou plusieurs plateformes, et pour laquelle le temps d'arrivée entre deux paquets consécutifs reçus reste inférieur à un certain seuil (25 heures). La différence de temps se calcule en convertissant toutes les dates au format GMT.

Global_Session : Une *Global_Session* est l'ensemble de paquets qui ont été échangés entre une Source et toutes les plateformes pots de miel du projet *Leurré.com*.

Large_Session : Une *Large_Session* est l'ensemble de paquets qui ont été échangés entre une Source et une plateforme pot de miel particulière (sonde).

Tiny_Session : Une *Tiny_Session* est l'ensemble de paquets qui ont été échangés entre une Source et une machine virtuelle donnée. Comme chaque plateforme pot de miel émule trois machines virtuelles, une *Large_Session* est composée d'au plus 3 *Tiny_Sessions*.

Les données sont introduites dans la base, mais nous appliquons également un ensemble d'applications pour enrichir ces données primaires. Par exemple, pour chaque Source, nous voulons associer une position géographique, ou du moins un pays (Maxmind, Netgeo, IP2location). De même, pour chaque Global_Session, nous voulons déterminer (de manière passive) quel système d'exploitation est utilisé par la Source (p0f, ettercap, disco). Le processus global pour archiver l'ensemble des données est symbolisé par la figure 2.

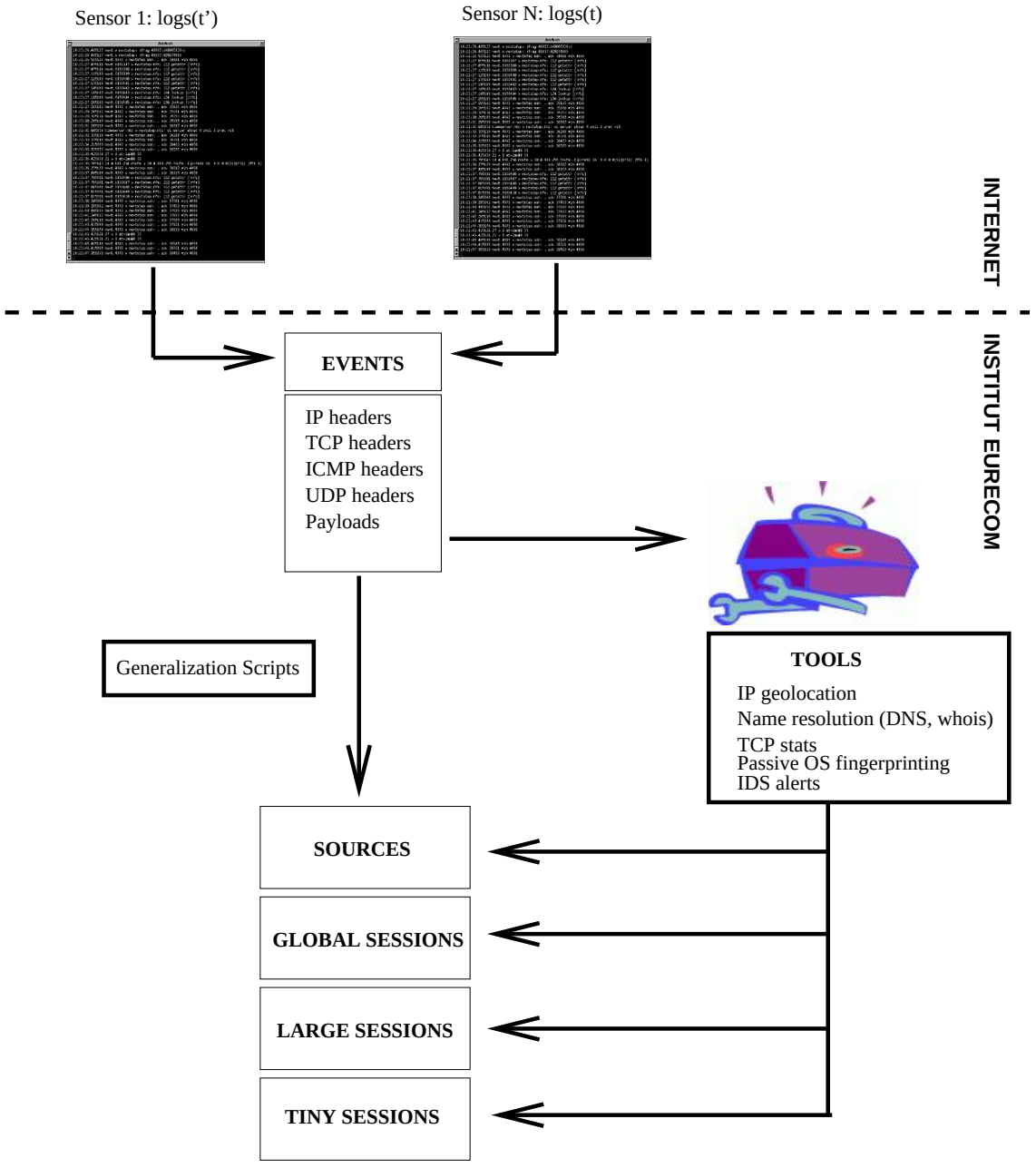


Figure 2: Des journaux Tcpdump à une base de donnée structurée

Ce travail préliminaire de collecte et de stockage évolue à travers le projet *Leurré.com*. L'ensemble de données collectées au cours des trois dernières années est unique à ce jour. Mais quelles valeurs apportent-elles réellement ? Dans quelles mesures peut-on tirer partie de ces données pour améliorer notre compréhension des processus d'attaques ? Ces questions qui découlent naturellement de ce paragraphe, sont à l'origine de la méthode d'analyse proposée dans les chapitres 4 et 5 du présent document, et dans la continuité des motivations à développer une méthode *HoRaSis*.

Empreintes d'Activités

Concepts

L'analyse doit profiter de la propriété suivante : les sondes (ou "plateformes pots de miel" : nous utiliserons les deux termes indifféremment dans la suite du document) sont distribuées dans différents pays, différents réseaux et différentes organisations. Il faut alors chercher un moyen de comparer ce qui survient sur chaque sonde, afin d'être à même d'en déduire ce qui est commun ou pas sur un ensemble de sondes. Cette étape est primordiale pour appréhender les caractéristiques des processus observés.

Nous avons défini dans un paragraphe précédent une *Large_Session* comme étant l'ensemble des paquets envoyés par une Source sur une sonde. Une *Large_Session* est donc la manifestation d'une *activité*, celle-ci étant :

Activité : Une *activité* est l'ensemble des actions exécutées par une Source sur une plateforme pot de miel.

Il est possible de caractériser une activité par l'empreinte qu'elle peut laisser sur une sonde pot de miel. Ceci conduit à définir une *empreinte d'activité* comme :

Empreinte d'activité : Une *empreinte d'activité* est un ensemble de paramètres (non nécessairement uniques) qui caractérise une même activité sur différentes sondes pots de miel.

Il est important à ce stade de comprendre qu'une activité se caractérise par une empreinte, mais que cette empreinte peut caractériser différents outils (si jamais les activités de ces outils ne sont pas distinguables à partir de la configuration actuelle des sondes). Nous supposons donc, d'une certaine mesure, que si les outils d'attaque ont un comportement suffisamment déterministe, nous devrions observer des *empreintes d'activités* semblables sur toutes les sondes ayant été la cible de mêmes outils d'attaque.

Paramètres caractérisant une empreinte

Nous décidons, dans la continuité des remarques précédentes, de regrouper toutes les activités observées (stockées sous la forme de *Large_Sessions* dans des groupes, appelés *clusters*. Les paramètres choisis sont basés sur l'expérience que nous avons acquise pour distinguer manuellement les activités, après lecture directe de fichiers tcpdump. Les paramètres initiaux sont ainsi :

1. Le nombre de machines virtuelles ciblées sur la plateforme pot de miel.

2. Les séquences de ports : à partir des paquets ordonnés par temps d'arrivée, nous pouvons extraire la séquence de ports distincts ciblés sur chaque machine virtuelle.

3. Le nombre total de paquets envoyés par la Source à l'attention d'une plateforme pot de miel.

4. Le nombre de paquets envoyés par la Source vers chaque machine virtuelle.

5. La durée totale pendant laquelle la source a été observée sur la plateforme (différence entre la date d'arrivée de son dernier paquet envoyé et de son premier paquet émis).

6. Ordonnancement de l'activité. Les paquets ont-ils été envoyés vers toutes les machines virtuelles en parallèle, ou vers l'une puis les autres ?

7. Le contenu des paquets (s'il existe) envoyé par la Source.

Malheureusement, ces paramètres peuvent varier d'une instance d'attaque à l'autre, du simple fait de certaines perturbations dans le réseau Internet. Parmi les perturbations envisageables, il peut y avoir :

- du réordonnancement : quand les paquets ne sont pas reçus dans leur ordre d'émission.
- de pertes : quand des éléments actifs du réseau (routeurs) décident de jeter des paquets.
- des retransmissions : quand l'émetteur ne reçoit pas dans les temps un accusé de réception.
- du retard : quand les éléments du réseau introduisent des latences et délais de traitement difficilement prévisibles.
- etc

Nous présentons dans le document une technique, qui s'appuie sur une propriété du champ IPID des entêtes IPs. Dans la plupart des systèmes d'exploitation actuels, ce champ n'est pas utilisé, mais s'incrémente de 1 à chaque envoi d'un nouveau paquet IP. En s'appuyant sur cette propriété, il est possible de limiter les impacts du réordonnancement et d'estimer les pertes. Cette technique ne peut malheureusement prétendre à corriger toutes les perturbations du réseau. Ceux-ci sont autant de fluctuations dans certains paramètres décrits ci-dessus pour définir une empreinte d'activité.

A partir de ce constat, nous avons choisi de classer les paramètres en deux groupes distincts :

- Les paramètres discrets : nous estimons que ces paramètres sont peu sensibles aux perturbations du réseau, et leurs valeurs doivent être considérées de manière exacte. Parmi ceux-ci, il peut y avoir les séquences de ports, ou le nombre de machines ciblées.

- Les paramètres modaux : il s'agit de paramètres présentant une distribution modale forte. Dans ce cas, leurs valeurs peuvent se généraliser par des intervalles, dont la largeur correspond à l'incertitude liée aux perturbations du réseau. Le nombre total de paquets envoyés par une Source, ou la durée pendant laquelle cette Source a été observée, font partie de ces paramètres aux valeurs généralisées.

La contribution respective de chaque paramètre dans la formation de *clusters* peut être évaluée au moyen d'indicateurs utilisés en théorie de l'information, comme l'IGR (pour *Information Gain Ratio*). Cet indice nous permet de réaliser, par exemple, que le choix du paramètre n'est (ou n'est pas) discriminant.

Nous regroupons donc à ce stade toutes les manifestations d'activité (Large_Sessions), ayant les mêmes valeurs discrètes, et ayant les valeurs des paramètres modaux dans les mêmes intervalles, en *clusters*.

L'étape suivante consiste à vérifier que les *clusters* ainsi obtenus sont bien valides. La démarche que nous avons entreprise consiste à vérifier que les Large_Sessions ainsi regroupées restent cohérentes en terme de contenu (ou *payload*) de paquets. L'algorithme proposé s'appuie sur la concaténation des différents contenus de paquets au sein d'une même Large_Session sous forme de phrase. La distance de Levenshtein et utilisée pour évaluer la distance entre les différentes phrases au sein d'un *cluster*. Une trop grande disparité en terme de distance peut amener à diviser le *cluster* en de nouveaux *clusters* plus homogènes.

Remarques générales et résultats

Cette méthode a permis de regrouper 1431000 Large_Sessions dans 52159 *clusters*, dont 8382 contiennent plus de 5 Large_Sessions. Ce regroupement en activité distincte offre plusieurs résultats détaillés dans le document, dont :

- Une étude de l'évolution des activités ciblant systématiquement les trois machines virtuelles, ce qui peut être associé à un balayage linéaire dans une plage d'adresses donnée. Ce scénario, fréquent parmi les activités observées au début de l'expérience, s'est raréfié au cours de l'année 2004, pour s'accentuer de nouveau en 2005. Cet exemple témoigne de l'importance d'une surveillance des codes malveillants, car leur comportement change rapidement au fil des mois.

- Une étude relationnelle entre trois types d'analyses, s'appuyant respectivement sur 1) les ports ciblés, 2) les séquences de ports ciblés et 3) les activités associées à un port et une séquence de ports donnés. Cette analyse montre clairement qu'il est peu significatif de produire uniquement des statistiques sur un port donné, voire même de se limiter à la séquence de ports.

- L'observation de l'apparition (ou disparition) de certaines activités au cours des mois. Nous montrons dans le document, à valeur illustrative, l'observation de la mort d'un ver. Ce ver, nommé Deloder, a fait grand bruit dans les médias au moment de sa diffusion, mais sa mort serait restée inaperçue, sans l'effort de personnes pour

faire de la rétro-ingénierie de code (tâche non triviale), ou sans une surveillance et une distinction des activités comme nous venons de le présenter.

Identification des outils

Nous rappelons ici que le terme *outil* représente tout code à l'origine de l'activité observée sur l'une des plateformes. Chaque activité est associée à un ensemble de valeurs de paramètres (discrètes ou modales). En ce qui concerne le contenu des paquets, il est possible d'extraire une phrase résumant ceux associés à une même activité. Les phrases sont estimées proches selon la distance de Levenshtein. Nous nous sommes appuyés sur le calcul de cette distance pour proposer une méthode simple de généralisation. Des solutions plus complexes existent (comme par exemple l'algorithme teiresias, l'ećhantillonneur ELPH Gibbs, etc).

Regroupant ainsi toutes ses valeurs, il est possible de créer une fiche signalétique, ou d'identification, des outils. Une telle fiche est présentée par la figure 3.

CLUSTER ID:	**IDENTIFICATION:**
2145	W32/Agobot−GM (sophos), also known as: Backdoor.Agobot.Id W32/Gaobot.worm.gen.k

FINGERPRINT:

* Number Targeted Virtual Machines: 1
* Ports Sequence: 2745,2082,135,1025,445,3127,6129,139,1433,5000,80
* Number Packets sent VM: 33
* Global Duration: 7s < t < 11s
* Avg Inter Arrival Time: < 1s
* Payloads: yes (DCOM, Netbios, WebDav)

Figure 3: Exemple de fiche signalétique obtenue

L'étape suivante consiste à associer un nom commun à chaque fiche. Cette tâche n'est cependant pas aisée, pour plusieurs raisons :

- Les outils en activité ne sont pas parfaitement connus. Certains font l'objet d'une certaine popularité, mais ne constituent pas nécessairement la majorité du trafic malveillant collecté. Ce besoin d'une meilleure compréhension est la motivation première du projet *Leurré.com*.

- Suite à la remarque précédente, nous notons aussi une déconcertante uniformité de l'information, quand celle-ci semble disponible. Les sites tendent à répandre de l'information, non validée, et dont la source reste obscure.

- En s'appuyant sur les résultats préliminaires de notre analyse d'empreintes, nous obtenons un ordre de grandeur du nombre d'outils observables à partir des sondes installées, et ce, depuis quelques mois. Il s'agit de quelques milliers de clusters (8392

ont été observés comme provenant d'au minimum 5 sources distinctes). L'association entre fiche signalétique et nom commun ne peut donc pas être résolue de manière simple.

- Quelques outils ne sont que des variantes (différentes configurations et implémentations) d'un même outil générique. Il correspondra donc à plusieurs fiches signalétiques, telles que nous les concevons.

Discussion

Cette classification des activités observées conduit à des résultats intéressants, et certains d'entre eux ont fait l'objet de publications. Il faut aussi avoir conscience que celle-ci n'est pas insensible à des techniques malveillantes pour la contourner. Nous décrivons de tels scénarii dans la section 4.6 du document. Les outils peuvent changer de comportement pour tromper cette classification, mais ce changement ne sera visible que par une observation de leurs activités. Il faut alors contrôler certains indicateurs (nombre de nouvelles activités enregistrées, fréquences de leur apparitions, etc), afin de détecter tout changement comportemental. Ceci est une direction propre du projet que nous n'aborderons pas dans la suite, car elle n'est pas directement liée à la problématique posée par ce document.

L'étude des empreintes d'activités nous renseigne pour conclure sur plusieurs aspects. Parmi ceux-ci, nous pouvons citer :

- L'évolution temporelle des activités d'un même outil sur une échelle de temps de plusieurs mois (années).
- La détermination d'activités propres à une unique plateforme, ou un un ensemble (voire la totalité) de plateformes.
- L'évaluation statistique de la représentation d'une activité donnée sur une plateforme donnée.
- La mise en garde annonçant l'observation de nouvelles activités.
- La corrélation qui peut exister entre les activités observées et les alertes émises par les systèmes de détection d'intrusions insérés dans le réseau hébergeur.

Chacun de ses aspects est abordé dans le projet *Leurré.com*, et ils restent ouverts à l'application de nouvelles solutions et innovations.

La méthode que nous proposons, prńommée *HoRaSis* pourrait s'en tenir à cette classification par empreinte d'activité, car elle est l'élément fondateur pour de nouvelles études. Il apparaît néanmoins des questions récurrentes, à chacune de ces études sur les empreintes : « Peut-on extrapôler la propriété de cette empreinte à un ensemble d'autres empreintes ?», ou «Est-ce que la propriété observée pour ces empreintes peut être mise en relation avec les propriétés précédemment annotées ?»

En d'autres termes, les empreintes d'activité suscitent en permanence une étude approfondie. Celle-ci conduit à déterminer ou vérifier une propriété propre à l'empreinte, mais

qui n'est pas obligatoirement partagée par l'ensemble. Ainsi, certains outils implémentent une couche protocolaire TCP propre, contenant des erreurs, ou du moins certaines caractéristiques, qui forment un moyen supplémentaire d'identification. Il est bon de savoir si plusieurs empreintes possèdent les mêmes caractéristiques, afin de savoir si les codes à l'origine de ces traces s'appuient sur la même couche protocolaire imparfaite. Dans un souci d'automatisation, nous sommes alors confronter au problème suivant :

- Comment *marquer* toutes les empreintes d'activités qui possèdent de mêmes propriétés ?
- Comment trouver rapidement *toutes* les empreintes qui partagent les mêmes ensembles de propriétés ?
- Comment ajouter de manière rapide et aisée une nouvelle analyse (étude d'une nouvelle propriété) aux résultats déjà établis par les deux questions précédentes ?

C'est dans le but de répondre à ces trois questions que nous proposons dans la section suivante une méthode complémentaire pour corréler toutes les analyses bâties ou à bâtir à partir des empreintes. Il s'agit de l'`analyse corrélative`.

Analyse Corrélative

Cette étape répond à la problématique précédente. Elle vise à automatiser la recherche de relations entre des propriétés partagées par un ensemble limité d'activités. Elle permet de conduire indifféremment deux catégories d'analyse :

- *Analyse intra-activité :* Au sein d'un même cluster (associé à une activité), ce type d'analyse cherche à extraire des propriétés qui sont plus spécifiques à celui-ci qu'aux autres, afin d'enrichir nos connaissances sur le phénomène à l'origine de ces traces.
- *Analyse inter-activité :* La seconde analyse cherche à trouver des propriétés communes à certaines activités, puis à les regrouper. Dans l'exemple cité dans les lignes précédentes, ce type d'analyse permet de regrouper *toutes* les activités qui ont des empreintes présentant la même caractéristique au niveau protocolaire.

Nous cherchons donc ici à trouver tous les ensembles d'activités partageant plusieurs propriétés. Nous voulons bien sûr que ces ensembles n'oublient aucune empreinte. Dans le cas d'une analyse intra-activité, les ensembles ne contiendront au plus qu'un élément, à la différence d'une *analyse intra-empreinte*.

Pour parvenir à ce résultat, nous profitons d'une technique extraite de la théorie des graphes. Nous ramenons le problème à celui plus connu de la recherche de sous-graphes complets (cliques) de poids maximum (*dominant set* dans un graphe. De manière simplifiée, il est nécessaire pour chaque analyse considérée de suivre un algorithme en 5 étapes, décrites ci-dessous dans le tableau 1. Le tableau 2 illustre chaque étape par un exemple concret qui a été implémenté. Il s'agit de chercher toutes les activités qui ont été lancées à partir d'un même groupe de pays :

Table 1: Les étapes de chaque analyse corrélative

Etape	**Description**
1	Définir une propriété à étudier
2	Représenter la propriété pour chaque activité
3	Quantifier sa représentation
4	Définir une distance pour comparer les activités
5	Construire la matrice de similarité entre activités pour cette propriété

Table 2: Les étapes de chaque analyse corrélative : un exemple

Etape	**Description**
1	Distribution des pays à l'origine de chaque activité
2	Distribution vectorielle
3	Pourcentage des empreintes provenant du pays X pour une même activité
4	Distance euclidienne vectorielle (ou technique du *peak picking*
5	Matrice nommée ici $\mathcal{A}\_Geo$

Nous avons suivi cet algorithme pour différentes analyses. Dans le cadre de ce rapport, nous avons pu ainsi construire un ensemble de matrices, chacune représentant l'étude d'une propriété particulière, liée à une finalité donnée :

Table 3: Matrices d'analyses utilisées dans cette thèse

Nom de la matrice	**Propriétés étudiées**
$\mathcal{A}\_Geo$	Distribution des pays d'où chaque activité est observée
$\mathcal{A}\_Env$	Distribution des plateformes ciblées par chaque activité
$\mathcal{A}\_OSs$	Distribution des OSs associés à chaque activité
$\mathcal{A}\_IPprox$	Proximités des adresses IPs attaquantes
$\mathcal{A}\_TLDs$	Distribution des TLDs (Top-Level Domains)
$\mathcal{A}\_Hostnames$	Catégories des machines attaquantes (noms de machines)
$\mathcal{A}\_CommonIPs$	Activités lancées par des adresses IPs attaquantes communes
$\mathcal{A}\_SAX$	Evolution des empreintes de chaque activité (par semaine)

Pour chaque matrice, nous extrayons alors les ensembles de *clusters* (ou activités) de taille et de similarité maximales. Afin d'effectuer chaque analyse dans un intervalle de temps raisonnable, nous avons eu recours à une méthode proposée par Pellilo et Pavin dans **??**. Elle s'appuie sur l'itération de fonctions particulières, issues de la théorie des

jeux, pour accélérer la convergence vers les solutions (l'extraction des ensembles de taille et de similarité maximales).

Une fois que ceci est appliqué à chaque matrice, il est alors possible de marquer les activités par un label, indiquant leur attachement à la propriété étudiée. Un exemple est fournie en figure 4.

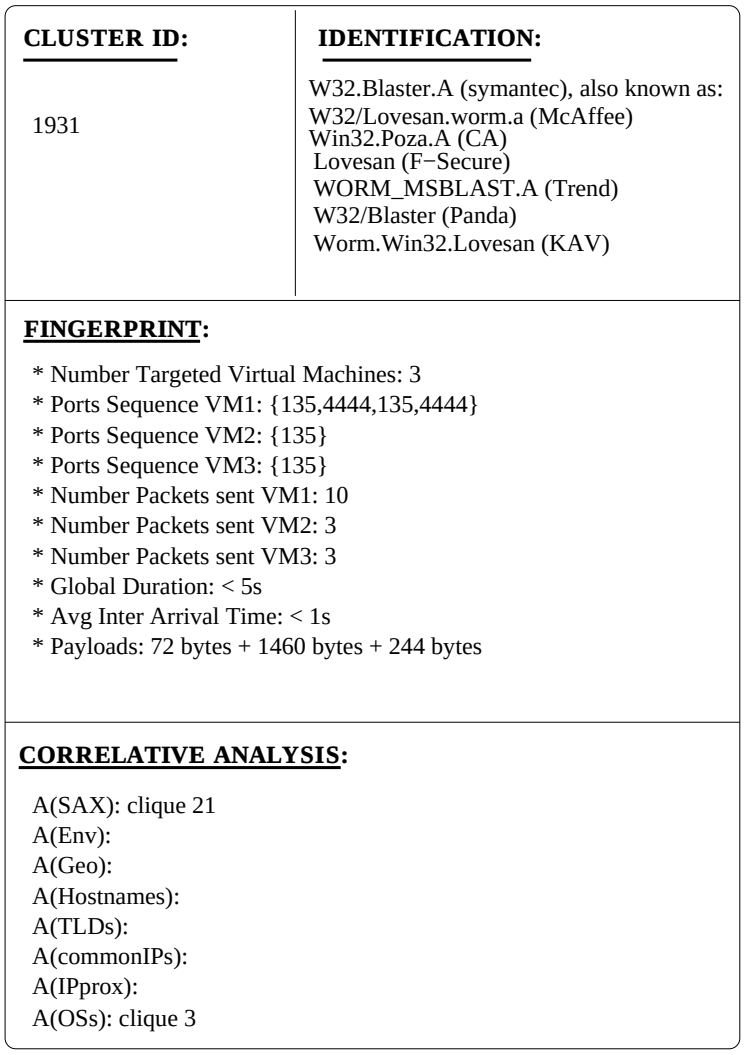


Figure 4: Exemple de fiche signalétique obtenue

L'intersection des ensembles obtenus pour chaque matrice permet également de récupérer les sous-ensembles vérifiant non plus une mais plusieurs propriétés fortes.

Découverte Automatique d'Information

Nous détaillons dans la section 6 de ce document des résultats obtenus à partir de certaines analyses (matrices) crées ci-dessus. En particulier, nous étudions :

- $\mathcal{A}\_Geo$

- $\mathcal{A}\_Env$
- L'intersection de $\mathcal{A}\_Geo$ et de $\mathcal{A}\_Env$
- $\mathcal{A}\_SAX$
- L'intersection de $\mathcal{A}\_SAX$ avec $\mathcal{A}\_commonIPs$, $\mathcal{A}\_Hostnames$ et $\mathcal{A}\_OSs$

$\mathcal{A}\_SAX$ est intéressante, car elle s'appuie sur une méthode innovante (SAX, pour *Symbolic Aggregate approXimation*) pour comparer les évolutions temporelles des différentes activités. Elle s'intègre facilement dans l'architecture de la base de données.

Les intersections révèlent aussi la pertinence de certaines analyses. Ainsi, les ensembles obtenus en croisant les deux matrices $\mathcal{A}\_Env$ et $\mathcal{A}\_Geo$ regroupent des activités venant de mêmes pays et ayant ciblées les mêmes plateformes. Ces activités peuvent être par ailleurs très différentes en terme d'attaques (services visés, contenus des paquets, etc). On peut y voir plusieurs raisons :

- Certaines machines mal configurées ciblent régulièrement un même réseau.
- Il s'agit de la même origine, ou organisation, pour toutes ces activités.

D'autres scénarii sont aussi envisageables. Il est possible de continuer l'analyse en croisant ces informations avec d'autres complémentaires (les noms des machines indiquent-ils des serveurs ? etc) afin d'affiner la compréhension de ce phénomène.

Chacune de ces matrices peut être également remodelée par de nouvelles techniques (séries temporelles, distances entre adresses IPs, etc) et de nouveaux outils (techniques de détection passive de systèmes d'exploitation, etc). D'autres, enfin, peuvent s'ajouter aisément dans cette architecture.

Conclusion

Nous avons présenté dans ce document une méthode, nommée *HoRaSis* qui peut se résumer par la figure 5.

Des capteurs de type pots de miel ont été déployés dans divers réseaux et divers pays dans le monde. Ils ont une configuration suffisamment légère pour permettre leur déploiement, et un contrôle par des capteurs étalons plus complexes est requis pour vérifier périodiquement qu'aucun biais n'est introduit pas le choix des capteurs. Les données brutes, ou paquets capturés, sont collectés grâce au projet *Leurré.com*, et stockés dans une base de données dédiée. Les paquets liant une Source (terme qui qualifie une adresse IP dans une fenêtre de temps court) à un capteur forment une *Large_Session*. Les Large_Sessions sont les représentations de différentes activités. Nous regroupons alors toutes les Larges_Sessions partageant des paramètres similaires qui caractérisent l'empreinte d'une activité. A ce stade, sur la première ligne de la figure 5, nous avons regroupé toutes les activités qui partagent une même empreinte sur au moins un capteur. Ensuite, nous appliquons différentes analyses, qui se construisent, chacune, sous la forme

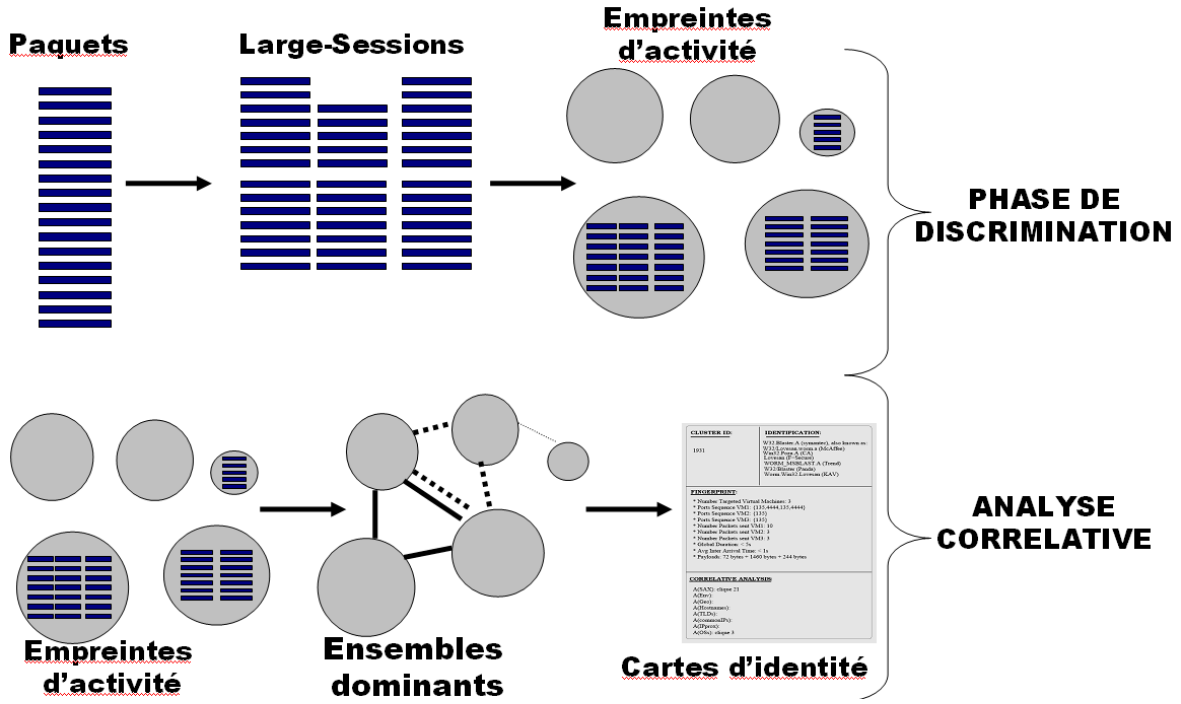


Figure 5: Schéma de la méthode *HoRaSis*

d'un graphe pondéré entre les différentes activités identifiées. La méthode est automatisée grâce à une technique de la théorie de graphes, nommée "extraction de graphes dominants". Nous obtenons in fine une cartographie des différentes activités observées sur chaque capteur, ainsi que l'ensemble des propriétés les liant (ou les différenciant) des autres. Cette méthode répond aux critères initiaux d'une méthode *HoRaSis*.

Nous avons montré au moyen d'un ensemble conséquent de données que cette méthode apporte une solide fondation pour accroître les connaissances des activités observables sur Internet.

Cette approche ouvre aussi différents axes de recherche, notamment :

- Quelles relations peuvent exister entre des observations très générales (télescopes) et locales comme celles du projet *Leurré.com* ? Sont-elles modélisables ?

- La configuration actuelle des plateformes est statique. Existe-t-il un moyen d'intégrer un certain dynamisme des configurations dans la méthode présentée ? Cela est d'autant plus important que les applications et les systèmes d'exploitation ont des versions changeant plus rapidement que la durée totale de notre analyse, qui peut s'étaler sur plusieurs années.

- La récupération d'information contextuelle est utile, mais pas suffisamment considérée par la méthode *HoRaSis* décrite dans ce document. Elle peut se formuler, cependant, aux moyens de matrices ou graphes de similarité, et s'intégrer dans l'analyse corrélative choisie.

Il n'est pas extraordinaire de finir une thèse par une ouverture vers plusieurs axes de recherche. Au contraire, cela nous conforte dans l'idée qu'il existe un besoin évident pour mieux comprendre les activités qui surviennent, et que la méthode proposée, nommée *HoRaSis*, offre une bonne fondation pour continuer sur cette voie. Elle permet déjà de répondre à un certain nombre de questions, et d'offrir de solides bases pour essayer de répondre à d'autres. Nous invitons maintenant le lecteur à se reporter directement au document, si l'anglais ne l'effraie pas, pour de plus amples détails concernant les techniques de la méthode et les résultats obtenus.

Chapter 1

Introduction

Security is a global concern in many domains of activity. Internet has the particular property of connecting people in quite an anonymous way and without strong traffic control. This advantage has also major drawbacks: malicious activities can take large amplitudes and have catastrophic consequences. As an illustration, it has been shown by Staniford et al. in [215] that a worm could saturate, in theory, 95% of one million vulnerable hosts on the Internet in 510 milliseconds. Another example is the increasing threat of electronic fraud that can result in losses reaching several millions of dollars per year (a cost of $150 million has been reported by the Commonwealth Government in 2001 in [106]). Internet makes large-scale crimes and devastating damages possible. It is thus really important to acquire a good understanding of threats and attack strategies. One method to obtain this knowledge is the monitoring and analysis of malicious activities, and it must be performed at a large scale to gain a global understanding of those phenomena.

This method is currently tried by means of numerous techniques, some of which belong to the category of so-called *honeypots*. Honeypots, honeytokens and honeynets have been used for some time in computing systems even if the use of this terminology is recent. In the late 80's, Clifford Stoll [218] had the idea of placing *interesting* data in appropriate places to lure hackers. This idea is now formalized as a *honeytoken* by Lance Spitzner [214]. In the 90's, Cheswik implemented and deployed a real *honeypot* [72]. Bellovin discussed the very same year the advantages and problems related to its usage [50]. In 1998, Grundschober and Dacier introduced in [107] the notion of *sniffer detector*, one of the various forms of what is also called today a *honeytoken*. As an attempt to clarify the terminology, Lance Spitzner has proposed the following definition for a honeypot [214]:

A **honeypot** is an information system resource whose value lies in unauthorized or illicit use of that resource.

This definition will be used throughout this thesis. The main advantage of so-called

honeypots is their intrinsic capacity to collect suspicious traffic only. In the last decades, all logs were mixed with production traffic, which made it difficult for security administrators to determine the malicious activities. This issue is now bypassed by honeypots. During the last three years, a lot of work has been done to design safe and useful honeypot architectures, that is, systems which are able to capture relevant information, from simple scan packets to IRC communication of the hacker without any danger for the network. Solutions are diverse and honeypot technologies are already used in a large variety of domains, from the major antivirus companies to governmental and international organizations. Monitoring malicious activities and reporting anomalies can be directly linked to some governmental initiatives and organizations such as the various Computer Emergency Response Teams (CERT™ou CSIRSTs) in many countries including France, US, Japan, Australia, Korea, Malaysia, Germany, etc[1]. Alternatively, governments promote privately-funded information sharing agencies, both for work related to overall network concerns and for specific sector-based needs. For instance, the UK is handling the Warning, Advice and Reporting Point (WARP) work to establish an interdisciplinary network to share critical security information. Other countries (the US, Canada, Japan, Germany, and Netherlands) have established industry-specific Information Sharing and Analysis Centers (ISAC) to serve a similar purpose.

Unfortunately, very few efforts are made to share information collected from these honeypots, except for some emerging challenges between security experts [35]. On the other hand, collective data exist from other initiatives like Dshield, MyNetWatchman or the Computer Network Defense Operational Picture [34]. These initiatives present informative statistics on attractive web pages, but the original source of information is not always clear. They invite any and all Internet users to send their firewall or Intrusion Detection Systems (IDS) logs in order to extract basic statistics out of them. The information is thus quite limited (hits per port), and contain statistics of not only malicious but also production traffic.

Thus, honeypots are a more valuable source of information than other existing techniques. Unfortunately, as we will explain in detail in the following chapter, a large effort is made toward optimizing their architecture, but few initiatives have emerged to organize and take benefits of the richness of the data. It is important to note that all the traffic collected by honeypots is by definition suspicious, so the analysis should not be biased by high false positive rates. In addition, the existing analyses are often limited to solving a particular problem. They can be techniques to monitor distributed Denial-of-Service attacks, fast and large scans, or typical attack scenarios. Most of the monitoring techniques have been implemented by the members of the Intrusion Detection community. However, the approach is slightly different with honeypots, as all collected traffic remains suspicious. False positives are therefore not the major worries anymore, whereas it is still the case with the majority of Intrusion Detection techniques.

[1]The European Commission recently established the European Network and Information Security Agency, ENISA, to coordinate the national efforts on cybersecurity and to serve as an advisory unit to the Commission and its component parts.

Observation: Honeypots are now widely deployed and technically mature. Analysis techniques, on the other hand, are really immature and targeted for solving particular problems.

From this observation, we have decided to build our own honeypot environment in order to collect such valuable data. The major motivation is that there is no trace of malicious data from honeypots that is publicly available and opened for analysis. This has been done through a project called *Leurré.com* [2], which has involved many partners from various countries and generated great enthusiasm from the security community [185]. Thanks to this informal consortium, we have managed to collect a huge amount of data from various honeypot sensors. It is important to note here that it represents a unique dataset of information, which is public[3] and that contain many months (years) of collected traffic. We are not aware of any other datasets like this one at this time writing.

Hypothesis: We have a unique set of data at our disposal, that represents malicious activities collected in various places in the world and from different network types.

What can we do with such valuable data? People have worked on lots of traffic analysis techniques. We can cite standard tools for network traffic monitoring like netflow ([75]), tcpdump ([17]), simple stats, or more complex but too specific theoretical methods. Nothing specific to honeypot data has emerged, and, more importantly, nothing constructive has been suggested as a basis to work on and exchange information. We can wonder at this stage if such dataset contain useful and original information, that would be hardly found by other approaches. If the answer is positive, could we identify such information in an automatic way? In short, we look for a potential framework, that would automatize the adequate analysis of such data. This framework, if it exists, will be called *HoRaSis*, as the bases for *Ho*neypot t*Ra*ffic analy*Sis*.

Thesis Statement: In this thesis, we want to show that:

- a distributed network of simple honeypots provides valuable data for the analysis and evaluation of threats and attack strategies.
- there exists an automatic way to extract information which seems relevant. Such framework will be called *HoRaSis*, for *Ho*neypot t*Ra*ffic analy*Sis*.

Honeypots offer a particular set of data, that might require a dedicated analysis technique. This claim will be justified in the first chapters through preliminary and promising experiments carried out on the data. Based on the acquired experience, it becomes clear that we need to develop a framework (called Honeypot Traffic Analysis or *HoRaSis*) to do automated analysis by means of a distributed network of honeypots. This framework should be an open one, that is, available for other potential analyses not yet performed.

[2] leurré = 'to lure' in French; leurré.com = allusion to the Eurecom Institute.

[3] For any partner that agrees to host one of our platforms, see Chapter 3.

HoRaSis must be an efficient framework that has to be validated through a dataset of honeypot traffic. The analysis is at the junction of many research domains, and the potential framework will not pretend to test all the research directions. From another point of view, the hypothetical *HoRaSis* should be open to several improvements by means of a modular structure. In a more general manner, the *HoRaSis* framework must fulfill at least the following requirements:

- *Validity:* A few analyses tasks have been performed manually, by digging into the data and pulling Ariadne's clew. This task, even tedious, has provided interesting observations. We thus impose any *HoRaSis* framework to show up these preliminary observations. There should not be any contradiction with manual analysis, and we expect, *a contrario*, an enrichment of these preliminary observations, as detailed by the next property.

- *Knowledge Discovery:* This property is clearly related to the previous one. We assume that manual observations might not be the unique ones. The *HoRaSis* framework must bring new and original knowledge. This new knowledge could be validated of course by complementary manual analysis and would justify the automation of analysis processes over the dataset.

- *Modularity:* The *HoRaSis* framework should cover several research domains. Indeed, we can easily identify several major distinct research branches such as Networking, Security and Data Analysis, and dozens of subdomains from which the framework might borrow techniques and algorithms. New techniques are constantly emerging in all of these fields. As a consequence, it is important that the framework presents a modular architecture. It will be easier, in this situation, to take advantage of new technologies by integrating them into the existing framework, and eventually, compare them with the ones already implemented.

- *Generality:* Collected data can drastically change with the appearance of new threats and new attack processes. The *HoRaSis* framework should consider that the analyzed data is evolutive. We should thus avoid building the framework on parameters which are related to particular data characteristics. In other words, the framework should be adapted to other types of dataset.

- *Simplicity and Intuitiveness:* The *HoRaSis* framework must extract information out of a given honeypot dataset (in this case from the *Leurré.com* project). The recipient of the information remains the analyst, who relies on the analysis to derive observations and decisions. It is thus important for the analyst to precisely understand the different steps which have led to this information discovery. The *HoRaSis* framework must avoid including opaque techniques, or *magic* black boxes.

It is proved in this report that such a framework exists and it has helped us finding other valuable results on malware activities. The proposed *HoRaSis* offers an automatic way to validate (or invalidate) these assumptions.

Thesis Contributions: The thesis contains the following contributions:

- Deployment and administration of a distributed honeypot architecture to collect data.
- Conception of a framework called *HoRaSis* to analyze honeypot data.
- Creation of new techniques to deal with honeypot data properties.
- Validation of the *HoRaSis* framework based on results obtained in a preliminary step by digging into the data.
- Presentation of new findings about monitored activities. Some of them have been clearly identified, others are new questions addressed to the security community.

To summarize, the thesis can be mathematically symbolized by the following problem statement:

- *HYPOTHESIS:* A honeypot architecture deployed to collect data.
- *INPUT:* A large volume of network traffic, all packets being malicious or at least suspicious.
- *PROBLEM:* Is this new source of information valuable? If so, how to build a reproductive, useful, valid and open analysis framework (*HoRaSis*) out of this?

In a concrete manner, *HoRaSis* is made of several important steps, that are illustrated in Figure 2.2. Steps 1 and 2 are the deployment and collection of information from honeypots. This work has been built on top of an academic initiative called the *Leurré.com* project. For a better understanding of the *HoRaSis* method, this project is detailed in Chapter 3, after an overview of existing approaches, both for monitoring and analyzing malicious traffic, presented in Chapter 2. The reader can find the details of the proposed *HoRaSis* that consists of steps 3 and 4 in Figure 2.2 in Chapters 4 and 5 respectively. Step 3 aims at grouping activities sharing some identical patterns, that is, all IPs sharing similar fingerprints on the honeypot sensors are identified and clustered. This is described in Chapter 4. In Step 4, we aim at analyzing the relationship between such groups based on a dedicated graph method. All fingerprints observed on the sensors which present common characteristics are automatically detected and analyzed. The method is carefully described in Chapter 5. Finally, Step 5 is the outcome of the proposed *HoRaSis*, the valuable and concise information the expert can exploit and share with the community. Each chapter is illustrated by results obtained from the *Leurré.com* dataset.

Steps of the *HoRaSis* framework are represented in Figure 2.2, together with the corresponding Sections.

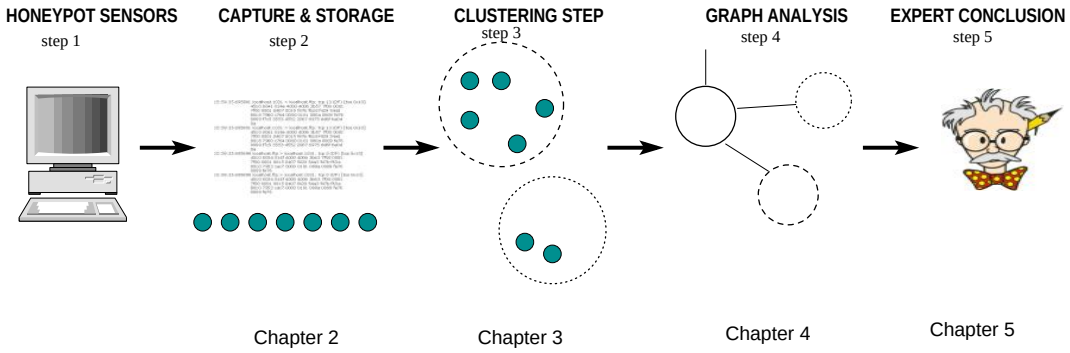


Figure 1.1: Distributed Honeypot Sensor Analysis

Chapter 2

Background and Related Work

An deus immensi venias maris, ne tua nautae
Numina sola colant, tibi serviat ultima Thule.
(Virg., Georg.,i,29.)
(Explanation: The Ultima Thule was, in ancient times, the
northernmost region of the habitable world -
hence, any distant, unknown or mysterious land.)

2.1 Background

2.1.1 Introduction

Monitoring malicious traffic is an important step to build efficient defensive techniques. Building a very high wall will take time but will be totally inefficient against underground intrusions. *Knowing the enemy and his strategies* is an important step that seems to be underestimated by many companies today [213, 144, 168]. Some vendors produce more and more complex boxes, integrating brand new technologies. However, it seems important, as a preliminary risk assessment, to get a very good feeling of the current threats.

HoRaSis is a framework we are looking for to analyze data obtained from honeypots in order to take benefit of the data intrinsic properties. Many techniques and projects are currently focusing on capturing malicious traffic. In other words, they intend, like honeypots, to capture meaningful data of suspicious activities. In parallel, many existing solutions from various research domains have already been applied to perform analysis of abnormal activities occurring in the wild. Thus, for a better understanding of this work

motivations, we briefly introduce these techniques in this section. However, as they are not clearly related, we prefer to distinguish the two categories separately:

1. Techniques which aim at capturing original traffic
2. Techniques which aim at performing original but specific analysis

"I keep six honest serving-men (They taught me all I knew); Their names are What and Why and When and How and Where and Who." ("The Elephant's Child", Rudyard Kipling, Just Stories 1902).
These six questions are often tools we as humans use in an attempt to gain knowledge. The same principle will be followed to better understand malicious Internet activities. *"What"* is what we call *"monitoring malware activities"* and will be defined in Section 2.1.2.
"Why" and *"When"* and *"Where"* are discussed in Section 2.1.3.
"Who" is discussed in the Related Work Sections 2.2 and 2.3.
"How" is the major contribution of the thesis which presents a new manner to operate.

2.1.2 Monitoring Malware Activities

It has been heard that "knowledge will set you free". When it comes to real-world network security, this phrase takes all its meaning. One can make a short experiment. If someone asks her friends or her network administrator the following question: "do you know what kind of attacks your machines are facing?", she will normally get the usual responses, including "well, the traditional worms, you know, Blaster, etc.", or "I do not know exactly. We have set up firewalls, antivirus, intrusion detection systems. They are doing their job quite well". The knowledge that can be acquired on pernicious activities normally comes by trusting the information spread over the Internet and the defense tools designed by specialists.

To handle this problem, many people are using network monitoring approaches[1], as defined by Bejtlich et al. in [47, 46] with the NSM acronym (Network Security Monitoring). He defines this activity as: *the collection, analysis, and escalation of indications and warnings to detect and respond to intrusions.*
Machines are interconnected within networks. It is thus possible, by looking at the connection level, in theory, to see everything. Unfortunately, there also exist many problems to monitor all activities, like encrypted packets, a large complexity in terms of protocols, standards and implementation bugs. It is also not sufficient to collect the data, without understanding what is under scrutiny. The analysis is a straightforward step to acquire useful knowledge of collected data.

[1] Historically applied for the Network Management tasks and the Performance Evaluations.

In the following, the focus is put on existing methods that aim at capturing and analyzing useful data, with the admitted goal of improving the knowledge of existing network threats.

2.1.3 The Purposes

There are many reasons why one would wish to monitor malicious activities, as stated in the previous paragraph. Among other things, we distinguish five major domains which may benefit from such a knowledge:

1. To build early warning systems: it is important to *react* fast against new threats, and at least, limit their overall impact.
2. To ease the alert correlation task: the administrator receives too many alerts from different systems in real time. A review of the state of the art of the correlation techniques has been described in [189, 188], however, most of them remain very basic or too theoretical.
3. To enforce the security policies toward the new threat trends: a concrete example can be found with the *PacketScore* project [127]. The authors prioritize packets based on a per packet score which estimates the legitimacy of a packet given the attribute values it carries. This is based on the distinction of nominal attribute value.
4. To perform tracebacks and forensics, in order to determine the root causes of the attacks and find the culprits: this is an important step for law enforcement, even if this technique might present social and legal problems in some cases, as reported in [137].
5. To confirm or reject some assumptions: for instance, the author explains in their report for the SANS GIAC Institute ([219]) that *an army of more than 100,000 machines* exist, but he does not bring any concrete proof for this claim that would let the reader check its validity.

There already exist several and various sources of information which intend to collect malware activities. They often differ in the way the system is positioned and the type of information it is collecting. Traditionally, approaches to threat monitoring fall into two broad categories, *host based monitoring* and *network based monitoring*. Host based techniques fall into two basic approaches, forensics and host based honeypots. Antivirus software and host based intrusion detection systems seek to alert users of malicious code execution on the target machine by watching for patterns of behavior or signatures of known attacks. Host based honeypots track threats by providing an exploitable resource and monitoring it for abnormal behavior. A major goal of host-based honeypots is to

provide insight into the motivation and techniques behind these threats. The second monitoring approach is to observe threats from the network viewpoint. Passive network techniques are characterized by the fact that they do little to intrude on the existing operation of the network. By far the most common technique is the passive measurement of live networks. They fall into three main categories: data from security or policy enforcement devices, data from traffic characterization mechanisms, and direct sensing or sniffing infrastructure. By either watching firewall logs, looking for policy violations, or by aggregating IDS alerts across multiple enterprises, one can infer information regarding a worm's spread. Other policy enforcement mechanisms, such as logs from router ACLs provide coarse-grained information about blocked packets. Data collection techniques from traffic planning tools offer another rich area of pre-existing network instrumentation useful in characterizing threats. Coarse-grained interface counters and more fine-grained flow analysis tools such as NetFlow offer another readily available source of information.

We propose in the following to give a short overview of all existing techniques. As previously written, most of them tend to focus on one direction only, either monitoring activities or analyzing collected information. It seems thus easier to split the related work description into two major branches:

- Techniques which aim mainly at collecting traffic. They are presented in Section 2.2.
- Techniques which aim mainly at analyzing collected traffic. They are presented in Section 2.3.

A summary of the state of the art is offered in Section 2.4 and justifies the thesis position.

2.2 On the Capture of Relevant Traffic

2.2.1 Honeypots, Honeynets, Honeytokens

Many projects derive from the honeypots and other honeypot-based architectures. It seems thus important, as a preliminary step, to clarify the terminology used. The interested reader can report to the two technical reports we have written at the beginning of this work, and that aimed at clarifying the terminology in use in the literature [190, 191].

A methodology has been proposed for students' practical works in [202]. We will consider indifferently, in the following, these three terms, and will employ the only *honeypot* word, following the definition suggested by L. Spitzner in [212], which is:

*A **honeypot** is an information system resource whose value lies in unauthorized or illicit use of that resource.*

How to architect a honeypot depends on the objectives it has to fulfill. A complex honeypot can be built to give the attacker a complete operating system with which he interacts. However, for detecting any unauthorized activity such as scanning, a simpler honeypot which merely emulates a variety of services in operation can be built. However, when capturing the latest worm for analysis is the main requirement, then a customized honeypot with the intelligence to interact with the worm and capture its activity is more appropriate. A honeypot can offer many different functionalities and the level of interaction they offer to attackers is important. It is supposed to give a granular scale with which to measure and compare honeypots. The more a honeypot can do and the more an attacker can do to a honeypot, the greater the information that can be derived from it. However, by the same token, the more an attacker can do to the honeypot, the more potential damage an attacker can do. We distinguish two different levels of interactions: respectively *low* and *high*.

A low-interaction honeypot is a computer system that provides certain fake services [44]. In a basic form, these services can only be implemented by having somebody listening on a specific port. Services are limited to listening ports. For example a simple Unix command like: $netcat - l - p80 > /log/honeypot/port\_80.log$ could be used to listen on port 80 (HTTP) and log all incoming traffic to a log file. In such a way all incoming traffic can easily be recognized and stored. However, with such a simple solution it is not possible to catch communication of complex protocols. The honeypot cannot trace TCP connections for instance as it logs only the first connection requests without answering. Another solution consists in building simple fake services that emulate a machine behavior. Generally speaking the attacker gets a better illusion that a real operating system exists and he has more possibilities to interact and probe the system. Special care has to be taken for security checks as all developed fake daemons need to be as secure as possible (buffer overflow risk, etc). Furthermore the knowledge for developing such a system is very high as each protocol and service has to be understood with expert details. In existing implementations though, fake services are often limited to simple scripts. Honeyd, Specter and LaBrea are honeypot solutions that can be classified as low-interaction honeypots (see [190, 191] for more information on these tools).

On the contrary, a high-interaction honeypot has a real underlying operating system to offer to the attacker. This leads to much higher risk as the complexity increases. On the other hand, the possibilities to gather information, the possible attacks and the attractiveness increase a lot. The goal of an intruder will most likely be to get as many privileges as possible on the target machine. By providing a full operating system to the attacker we offer him the possibility to upload and install new services/applications. This implies that the system must continuously be under surveillance. All actions can, and must, be recorded and analyzed to gather more information about the blackhat community.

Lance Spitzner explains that: "(his) perception of low interaction vs. high interaction is *intent*". With low interaction, we intend on limiting the attacker to only emulated services. With high interaction honeypots, we intend on giving attacker access to the full operating

system. Both deployments require a real operating system. With low interaction, the emulated services do run on a real operating system, as in Tiny Honeypot, Specter, and Honeyd [25, 9, 190]. However, the goal is to limit the attacker to interact with just the emulated services and not give them access to the operating system.

Honeypots are often generic names to define such architecture that follow the definition of L. Spitzner, and which remain, actually, very close to Intrusion Detection Systems, as they incorporate one or many of the following functionalities:

- Data collection
- Malicious activities Detection
- Logging capabilities
- Analysis techniques
- Decision and Reaction

Their originality mainly lies on the particular traffic they are monitoring, as it consists, echoing previous paragraphs, in data representing suspicious traffic only.

2.2.2 Darknets, Telescopes, Blackholes

CAIDA's Network Telescope

The Cooperative Association for Internet Data Analysis, also called CAIDA, has developed particular *honeypots*, called *network telescopes*. They define a *network telescope* as a portion of routed IP address space in which little or no legitimate traffic exists. Thus, monitoring unexpected traffic arriving at a network telescope provides the opportunity to view remote network security events such as various forms of flooding denial-of-service attacks, infection of hosts by Internet worms, or network scanning. As the authors mention in [158], network telescopes were named as an analogy to astronomical telescopes, and in both of these cases, have a large *size* (fraction of address space or telescope aperture). The ranges can be quite large (e.g. a /8 prefix corresponds to 16 million addresses), and telescopes cannot be used by anyone in practice. The telescopes used by CAIDA also assume that the telescope is monitoring contiguous range of address space. In terms of probability, they can extrapolate this way the monitored activity, based on the fact that for an IPv4 network of size /x, the probability of monitoring a chosen target is given by: $p_x = \frac{1}{2^x}$. This unique observation has lead to interesting worm studies, like:

- the Saphire/Slammer worm [159, 156]

- the Witty worm [157]
- the Code Red and Code Red II worms [22]

Telescopes are also very useful to detect and observe large Denial of Service attacks, as spoofed Source IP addresses involve a large range of IPs, and result in an important number of packet residues called *backscatters*. Such studies have been presented in [160, 162].

The extrapolation task is meaningful for all attacks that are involving the whole IPv4 address space, but this global approach might present a few limitations:

1. The probabilistic model does not sometimes hold, for the reason that some IP space regions are inexistent or not routed. Such IP spaces are commonly called the bogons [97]. These addresses, and other unallocated blocks are not taken into account in the probabilistic model.
2. Many attack tools are source-dependent and target nearby addresses. This has been shown in the Cod Red II propagation strategies [243], and it has also been confirmed by some experiments in [67] that a few attacks are specific to particular networks.
3. The coding of the worm might contain some bugs, especially the pseudo-random number generator. This can introduce a bias in the probabilistic model.
4. The amount of collected information is colossal, especially if the analysis needs to be performed over several months. It is not explicitly mentioned how telescope-based systems manage to store such data, but it either requires huge memory capacities and lookup resources or it is based on the NetFlow aggregated information. In this last case, some information might reveal to be missing during punctual experiments (see Section 2.3.2).

The CAIDA's Network Telescope is a very high-level monitoring system. This approach has led to noticeable results and we are convinced it has opened large avenues for investigation [76, 161]. This technique has also been deployed in other research projects that we detail in the next sections. However, despite these benefits, the huge amount of data is an important limitation for performing lower-level analyses. Furthermore, determining how the CAIDA members organize data is difficult. We can imagine that there also exists potential privacy issues, as the telescope should monitor some production traffic despite the fact that the range of IPs is theoretically unused. These two major drawbacks have motivated the creation of the Leurré.com project described in Section 3 and the need to organize efficiently the collected data and gather experience on the traffic over a long period of time (in terms of months, and even years).

Finally, we want to mention the interesting initiative launched by the CAIDA members and called the *Internet Measurement Data Catalog* (IMDC) [77]. This initiative is not

directly related to the Telescope approach, but it aims at building a system that would facilitate access, archiving, and long-term storage of Internet data as well as sharing the data among Internet researchers. CUrrently, though, they only share *meta-data*, that is, outcome of analysis (which makes the work on it not always simple). Concerning malware analysis, they offer at this time an access to backscatter[2] data from the UCSD network telescope. Unfortunately, they currently offer researchers only three weeks of DoS attacks data from January 2001. Despite this, the authors mention in [77] the same underlying motivations for network measurement research than those we defend in this thesis: a lack of meaningful and organized data to be shared with the community. This problem of data availability is recurrent for all considered solutions in the following.

The Internet Motion Sensor: IMS

The Internet Motion Sensor is a project from the Michigan University that utilizes a distributed sensing network based on the monitoring of globally routable but unused address space. The concept is similar to the *Network Telescopes*. To make things shorter, this technique has a variety of other names including network telescopes, darknets, and blackholes. Each blackhole sensor monitors a dedicated range of unused IP address space [27, 82]. As the architecture was deployed to monitor quite large IP Address spaces (/8, /16 and /24 networks), the authors do not have the capacity to log all packet information. In their defense, they have developed an efficient technique to manage the overhead of storing payloads. However, the project remains limited to traffic collection and storage. The analysis consists in some statistics and, as presented in [82], the analysis of a worm is restricted to the statistics of a single port and to the counting of packet payloads matching the signature of that particular worm. It is hard at this stage to make stronger comparisons between the sensors than the ones they have deployed.

However, Cooke et al. have presented an interesting study based on simple criteria in [81]. They use 10 large IMS sensors in different networks, belonging to major service providers, large enterprises and academic networks. Then, the authors have compared the observed traffic on each IMS according to three dimensions:

- Over all protocols and services (ports): they have concluded that *the amount of traffic (between sensors) varies dramatically and can differ by more than two orders of magnitude between sensors*
- Over a specific protocol (TCP) and port (135): this time, they have concluded that *there are large disparities between the number of unique source IPs seen across platforms, and that these differences correlate with differences in overall traffic.*
- Over a worm (Blaster) signature: they finally have concluded that *there are still significant differences in the number of unique sources between sensors.*

[2]Backscatters are residues of Denial of Service attacks.

This work has highlighted the fact that the CAIDA's approach, which extrapolates a large IP traffic observation into the whole Internet, cannot properly hold. First, there might be inhomogeneous activities within the monitored IP range. Second, there can be dissimilarities between IP ranges. This is another interesting justification of the deployment of smaller and more local sensors. The *Leurré.com* Project we have developed over the last years in order to collect valuable data is based on the same principle. Local positioning of honeypot sensors might bring different, or at least, complementary and valuable information. This partially explains the decision to develop a distributed architecture of sensors, instead of a global monitoring system like the telescopes presented in the previous section.

The information is not centralized. Each IMS sensor is responsible for gathering and archiving data, performing queries on its local data store, and generating alerts that are sent to a third element called aggregator. In this distributed storage environment, it seems hard to make cross sensors analysis, except on a few global statistics.

Team CYMRU's Darknet Project

A group of researchers founded in 2004 the Team CYMRU, Inc. One of their apparent objectives was to share their security experience with the community. They have launched the TEAM CYMRU's Darknet Project and have provided all the directions in their web site to settle such an environment [87]. The principle is also very close to the Network Telescope[3]. The authors' recommendations can be found on their web page: *At a minimum we recommend a /24, though smaller address space - even up to a /32 - will work.* This unclear requirement is completed by the directions to configure the router and to monitor the traffic. They give poor information, however, on the techniques to analyze the data. They only suggest to create simple *garbage meters* based on the traffic entering the Darknets. Figure 2.1 found on their web page illustrates such a metric. It shows simple statistics on the number of packets received per second (pps) on each sensor (noted DARKXX in the Figure). It is worth noting that values vary a lot among Darknets. It remains coherent with the remarks made in the previous section. However, one more time, a prevalent effort is made to build a useful architecture for monitoring malicious traffic, but no real effort is made towards the analysis.

iSink

Among systems that monitor large unused address spaces like the ones previously described, we can also mention iSink, developed by the University of Wisconsin-Madison [235, 236]. A large effort has been made on the architecture interaction provided by the *Active Sink* built on the Click elements [12]. The monitoring and analysis parts are based on measurement tools, like MRTG [167], FlowScan [1], and Argus [13]. These tools

[3]Team Cymru is also a collaborative member of CAIDA.

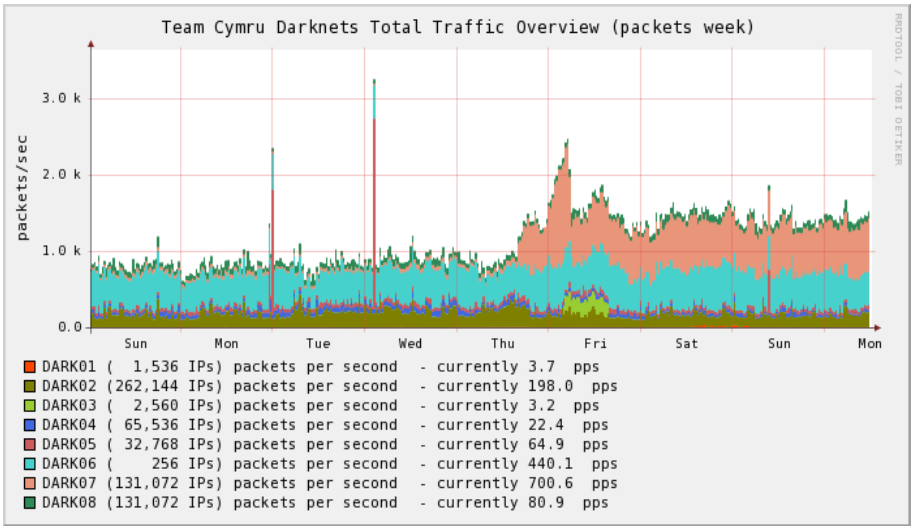


Figure 2.1: Darknet Traffic Garbage Meter from [87]

provide valuable flow level information and can deal with large packets traffic (iSink collects unsolicited traffic destined for approximately 100,000 unused IPv4 addresses within 4 class-B networks). The authors gather statistics concerning inbound traffic (bits or packets per seconds), TCP flows and other traffic measurement metrics. This provides interesting remarks on the traffic, like *the packet sizes, which are relatively constant, and the number of bytes and packets which follow a predictable ratio, one reason being the important volume of TCP traffic*. These measurement tools cannot be considered *stricto sensu* as malware analysis tools. They provide interesting pieces of information but are intrinsically not sufficient to identify, and thus to understand, malware activities.

MINOS

Minos is an emulation tool proposed by the University of UC Davis that presents the advantage of tagging network data considered as *low priority*, and then propagating these tags through filesystem operations and the processor pipeline to potentially raise an alert, whenever low integrity data is used as control data in a control flow transfer [84]. It implies a tuning at the processor level and the operating system.

Thus, Minos is by definition a host-based intrusion detection system. In [85], the authors have deployed it in machines plugged in the wild, to use its features as a honeypot. However, this architecture is very specific to monitor attacks that intend to hijack the control flow of the CPU. The authors provide in [85] few examples of buffer overflow exploits against services like SQL Server 2000, Linux wu-ftpd, etc. The monitoring of the activities is restricted to Minos alerting system and assembly codes. The capture process is here very close to a detection mechanism.

Lobster

Lobster (for Large Scale Monitoring of Broadband Internet Infrastructure) is a European project with a research Consortium (including among others the greek FORTH foundation and the University of Amsterdam) that aims at deploying a pilot European infrastructure for accurate Internet traffic monitoring [60]. In order to support collaborative passive network monitoring across a large number of geographically distributed uniform access platform sensors, Lobster is based on a distributed uniform access platform, which provides a common interface for applications to interact with the distributed monitoring sensors. This interface is build onto the Monitoring Application Programming Interface (MAPI), while the network monitoring system is called SCAMPI (SCAlable Monitoring Platform for the Internet [83, 166, 86, 55]). A great effort has been made during the project on the technical architecture of the traffic capture, as it can handle high-speed network cards (10 Gbits/s) and has been directly implemented in some Network Processors. The architecture can also measure network performance and behavior at high-speeds, in order to feed billing components. First, this architecture has not been designed to collect malicious traffic only. This is thus not a distributed *honeypot* architecture, by definition. Second, the analysis of malicious traffic is handled by a dedicated Network Intrusion System (IDS) using converted Snort rules. As a consequence, and despite its title, this project remains quite far from the initial problem statement we have defined, as the monitoring of malicious activities remains limited to the observation of the snort alerts.

2.2.3 Logs Sharing

There are some other approaches very similar to monitoring consoles, which differ from the origin of the data. Unlike monitoring consoles that aim at representing logs and alerts issued by the active security elements of the supervised network, these approaches collect data from... everywhere. Anyone who is willing to send data registers, installs a small software that periodically sends data. Then, some scripts parse these datasets, grabbing a small number of elements, like the targeted services, the number of observed different IPs, etc. Among these approaches, we note the interesting projects called WormRadar, Internet Storm Center from the SANS Institute, Dshield, MyNetWatchman [36, 33, 5, 163]. They give a very good overview coming from different profiles, and are thus, useful to get a first hint on the malicious activities happening in the Internet. However, they collect logs from any network. Thus, all these projects make the assumption that what they receive in the logs is a good representation of global activities. Such an approach presents a few inconveniences:

- They trust all entities that send their logs.
- They mix security policies filtering and malware activities.

- We have no information on the exact source of the information (academic, military, industry network?)
- They do not give explanations on how they extract information from these various log formats.
- They make simplistic links between activities on a given port and worm names. However, many malware can take benefits of the very same port; the same malware can be characterized by activities on many ports ... and vice-versa.
- There is no way to dig into the data. The analysis remains limited to the plotted statistics and reports.

WormRadar is slightly different, as it relies more on a visualization principle [36]. The software to install emulates a couple of services, (IIS, FTP, Telnet), for a brief period before it cuts the connection with a FIN allowing such things as a login to the FTP server as anonymous or to see what the GET/Head etc. It then allows listening on a user-defined series of ports (both TCP and UDP). On these ports, WormRadar seems to accept any data sent. It is an interesting little toy but the docs on its site are sadly lacking so there is a need to experiment to find out what it does. In any cases, these tools are practical synthetic elements and visualization methods of Internet activities. However, they cannot be considered as valid and trustworthy providers to build on a certain knowledge of malware activities. As an illustration, Wormradar can be perceived as an *aggregated port hit statistic site*: the offered data is the web interface. However, data is sent by anybody willing to participate. Conclusions and available data are thus limited to exposed graphs in the web interface.

2.2.4 Others

In Large

Other tools can be used as well to monitor malicious traffic. They are not dedicated to this task only, or, so to say, they do not pretend to collect malicious traffic only. This includes all approaches hidden behind the expressions *Intrusion Detection Systems* (IDSs), or *Intrusion Prevention Systems* (IPSs) [18]. Concerning IDSs, it is important to note that they still tend today to trigger an abundance of mostly false alerts. In [122, 123], Julisch et al. explain this fact by a few reasons, including the under and over specified signatures, or the lack of abstraction. The major difficulty consists in classifying traffic into two categories: normal and abnormal. This issue does not exist anymore, however, with respect to honeypot technologies. IPSs normally collect logs of the traffic they block, so, they are not monitoring malicious traffic only, but instead, all packets that do not match the security policy chosen in the considered network. This is not the exact expectations, as each traffic is then tightly correlated to the network security policy within which it is captured.

Mwcollect and Nepenthes

Among other collection tools, we can mention the new *mwcollect* tool [31]. Mwcollect is somehow different from typical honeypots because it was originally designed to collect bot software, but the current version collects worms and other forms of malware that take advantage of vulnerabilities that mwcollect exposes. According to the mwcollect Web site, systems that run the tool cannot be infected with malware due to the way mwcollect operates internally. It binds to specified ports, waits for an exploit attempt, scans for shell code, and tries to download any related malware. Captured malware can then be added to a database at the mwcollect Web site.

The second collection tool *Nepenthes* is similar to mwcollect [32][4]. It also presents known vulnerabilities to the network and waits for intrusion attempts. Current modules for Nepenthes enable it to emulate vulnerabilities with DCOM, Local Security Authority Service (LSASS), WINS, ASN1, NetBIOS, SQL Server, and a lot more Microsoft services. Since Nepenthes runs on Linux systems, none of those services would actually be available, which means exploits against them would have little or no effect on the underlying OS. Just like mwcollect, when Nepenthes detects intrusion attempts, it tries to download any related malware through a variety of methods including FTP, Trivial FTP (TFTP), and HTTP. Captured malware is then sent to a center server hosted by the developers of the tool. These tools are very interesting to collect the whole malware, including payloads and binaries. They are thus very interesting to capture meaningful information. Unfortunately, there are dedicated to a small number of activities, the ones that take benefit of offered vulnerabilities. Furthermore, it remains to the analyst, once the data has been collected, to perform a non-obvious task for dissecting captured data.

HoneyTank

HoneyTank can be seen as a good complement of large telescopes and darknets. It has been designed with the idea that such architectures might have difficulties in logging all activities for thousands of unused IPs. The authors suggest to tag packets in a particular way, exploiting the timestamp option of TCP. This avoids monopolizing lots of memory and cpu resources to keep connection states. This technique, however, is restricted and can only work to emulate TCP services. In addition, it requires the attacker to correctly reply to TCP timestamps (TSecr field must echo the last received Tval value). One more time, a noticeable effort is made to participate in the building of a honeypot architecture, but very few to monitor and analyze the output.

[4] In February 2006, the two projects merged operations into a single malware collection tool also called *Nepenthes*.

2.3 On the Analysis of Traffic

2.3.1 Positioning

There are many techniques which aim at analyzing the traffic. All active techniques are not considered in this document. We identify two major categories: some tools aim at capturing packets, while others bring add-ons on the traffic they collect by somehow interpreting them. The first category is out of the scope of this section, as it has been briefly introduced in Section 2.2. We detail in the following some techniques that could compete as candidates to analyze malicious traffic.

2.3.2 Netflow

name	Description
srcaddr	Source address
dstaddr	Destination address
input	Input interface
output	Output interface
dPkts	Number of packets
dBytes	Number of bytes
First	Start of NetFlow
Last	End of NetFlow
srcport	Source port
dstport	Destination port
tcp_flags	TCP flags
tos	IP type-of-service

Table 2.1: Some Relevant NetFlow Fields (v5)

Many router manufacturers implement interesting logging capabilities that offer to summarize and analyze the traffic. One famous data is the CISCO's NetFlow[5]. The concept of flow has been proposed by Claffy et al. [78] as: *a flow is active as long as observed packets that are meeting the flow specification are observed separated in time by less than a specified timeout value*. Flows have proven to be a very useful tool for measurements and traffic characterization. This is also reflected in the efforts to standardize flow data measurement and collection architectures [59, 180, 91]. CISCO's flow level aggregation technique [75] almost fit the model by Claffy et al. According to CISCO documentation, the NetFlow

[5]We use the term NetFlow for both the concept as well as individual records

implementation identifies a flow by the tuple *(srcaddr, srcport, input, dstaddr, destport, tos)*. Table 2.1 summarizes some of the relevant fields related to NetFlow records.

NetFlow presents a few limitations. For instance, it may aggregate packets from several TCP connections into one NetFlow, e.g. if the same socket is used for several connection attempts, as done by file-sharing applications or some attack tools [207]. A contrario, a TCP connection can be split into many NetFlows, if the TCP connection is longer than the flow timeout. As a consequence, it is very hard from the aggregated NetFlows layer to deduce the TCP connection layer. The flow model is very interesting though, and has been proved useful in quite a few studies. For example, it has been used by Thompson et al. [224] for traffic measurement and characterization. Lin et al [143] evaluate the effect of different flow classifiers on switching performance, while Feldmann et al. [98] examine the impact of application-layer aspects on the flow characteristics. Newman et al. [164] propose IP switching based on flows.

More in the focus of this document, NetFlows have also been used to monitor malicious network traffic. CISCO itself offers a product called The Cisco Security Monitoring, Analysis and Response System (Cisco Security MARS) that models the flows in the network, and make periodic comparisons based on expert rules and network topology information. Many reports have been applied on NetFlow traffic from high-speed networks. As an illustration, Duebendorfer et al. present in [93] a study of the Blaster.A and Sobig.F worms in a Swiss backbone network called SWITCH. We have extracted from [93] the details of Blaster's infection and presented it in Figure 2.2. Blaster infection mechanism is currently well-known, and is split into activities against two distinct TCP ports, 135 and 4444 on the victim size.

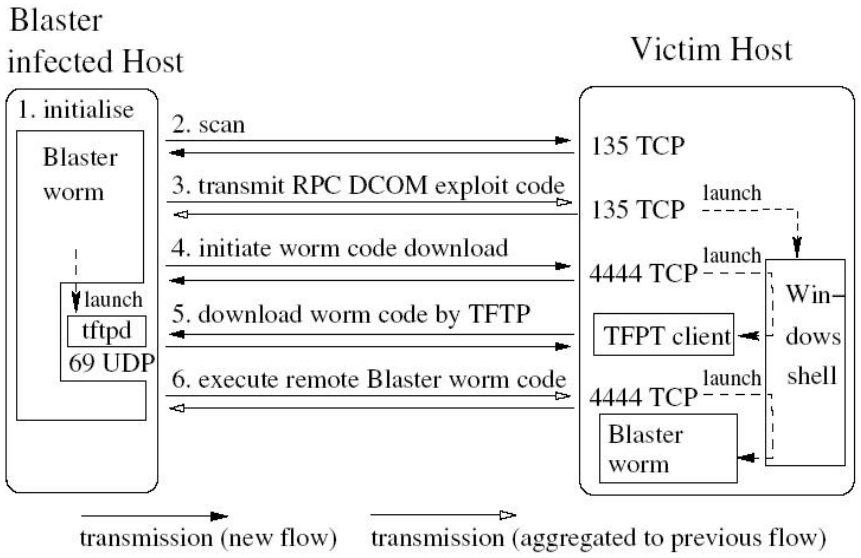


Figure 2.2: Blaster's Infection Steps [93]

The authors have then defined four different infection stages that classify to which extent a Blaster infection attempt on a victim machine is successful. They have asso-

ciated each infection stage to its corresponding NetFlows. Finally, they have extracted flow-level plots for each infection stage, in order to estimate the number of stage cases. Unfortunately, it is not completely sure that they are seeing Blaster.A activity by observing activities on port 135 or on port 4444, and by imposing a few constraints on the flows (ranges for flow byte sizes). This limitation comes from the intrinsic definition of flows. Another important limitation is that the correlation between NetFlows and Blaster's infection stages has been possible thanks to the diagram presented in Figure 2.2. However, the knowledge about attack tools is either kept secret or simply not acquired. Thus, this technique can only be applied *a posteriori*, when the attack tools have been well-studied. The solution we propose will be inspired from the NetFlow fields to classify our data, but, at the same time, will be designed to avoid the same pitfalls and limitations.

It is also worth noting that other protocols and standards similar to Netflow exist. One is *sFlow*. *sFlow* is an open standard defined in RFC 3176 [179]. It is based on packet sampling, and while NetFlow only captures information about IP packets, *sFlow* can be used to analyze other protocols like Ethernet, IPX and Appletalk. IPFIX and PSAMP are two IETF working groups that are working on standardizing IP flow export and sampling. PSAMP concentrates on defining methods for sampling based passive measurement of flows, while IPFIX is a new effort to define what information flow records should contain and how they are exported to collectors. At the moment, nothing concrete has emerged, and none of the tools based on these solutions are dedicated, as far as we know, to monitor malicious activities in particular [147]. The interested reader can have a look at [15] where an interesting state-of-the-art of many dozens of netflow-based tools is presented.

2.3.3 Billy Goat

Duponchel et al. introduce in [118] "Billy Goat", a honeypot-like system dedicated to worm detection. The capture tool, at first glance, is similar to products like WormScout, or even honeyd ([195]), but is built on top of iptables in a clever way. With the system basing all its feigned services on an infrastructure for virtualization, a single Billy Goat machine can appear as many addresses on the network. At the same time, the virtualization infrastructure allows the use of standard programming tools and interfaces to create new feigned services for Billy Goat. The mountains of data that these virtual services create are stored and cataloged (that is, logged) in a relational database, which is summarized in Figure 2.3. An interesting effort has been made to help organizing the data. Data is aggregated for each observed source address over a specific time frame (not specified in [118]). The aggregation results from five main parameters:

- The Source IP address
- The time period covered by the data in the model
- The IP addresses of the Billy Goat sensors that have reported the activity

- the description of network/transport level activities (Source and Destination IP addresses/ports, Protocol, flags)
- Description at the application level (XML fields: REQUEST and HOST, see Figure 2.3)

Unfortunately, it is not really clear how the authors can map each application level data into a XML format. The document remains very vague on that field. Furthermore, the authors apparently avoid completely other meaningful packet exchange formats like TCP, ICMP or UDP. However, the impact is different in the case the packet is flagged RST/ACK with payload (crafted packet for instance), or if it is SYN-flagged. Finally, the real purpose of the database is not clear, as most of the (reduced) collected information is *in fine* generalized during the analysis by means of a simple summarization of the data (orders of magnitude, statistics). We note here a clear effort to organize the data. Unfortunately, the extraction of information remains too restricted to build upon a strong analysis as we intend to.

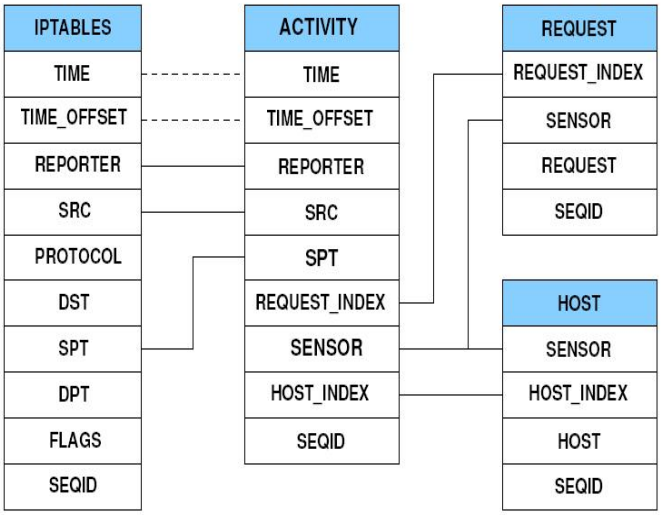


Figure 2.3: Database Structure used in Billy Goat [118]

HoneyStat

It seems also relevant to present HoneyStat [197]. This original approach considers Destination/Source Correlation (DSC) to match the same port being used for source and destination scans to identify compromised machines. The authors demonstrate in [197] that this approach works fine for analyzing scan-based, fast spreading worms. The general idea behind DSC consists in keeping a sliding window of local network traffic. Two general items are tracked: for each port witnessed in this traffic, the authors record the address of the inside destination host and the scanning source from the monitored network. If a

source scan originates from a host that previously received a scan on an identical port, i.e., they observe a worm behavior-like infection pattern, and treat this local host as a suspicious victim. In other words, if a host gets a packet on port i, and then starts sending packets destined for port i, it becomes suspect. Then, if the immediate outgoing scan rate for the suspect hosts deviates from a normal profile, the suspicious victim is considered to be infected.

This analysis technique is quite specific to worm detection. We can also cite a few other techniques, having similar approaches, like Collapsar from Purdue University [121]. Researchers have also proposed statistical models, e.g. Kalman Filter [244], analyzing repeated outgoing connections [232], and victim counter-based detection algorithms [233]. All these approaches, however, remain quite worm oriented. We are interested in the following to classify traffic more than to find a detection technique that would be applicable in very specific conditions or to very specific classes of attacking activities.

2.3.4 Monitoring Consoles

The topic has been widely studied in [188, 189], in which we have described many techniques and existing tools. The monitoring consoles are often relying on the alerts and logs issued by security systems (firewalls, IDSs). We have shown the gap that exists, as of today, between sophisticated techniques presented in research papers and actual implementations that are readily available. We have presented solutions not only from the Intrusion Detection community but also from the Network Management community, which has tried to solve similar problems for many years. Solutions exist that come from various research domains and that have proven their efficiency in many cases. However, we have reached a deceiving observation: among tools and products that have been proposed so far in Alert Correlation, very few implement such approaches. Most of them are limited to down-to-earth, pragmatic techniques, such as pattern matching or database queries. As a consequence, we do not detail more this field in the following, but we invite the interested reader to have a look at [188, 189] for a more complete state-of-the-art.

2.3.5 Vizualization Techniques

Network security visualization is an emergent field and a number of systems exist that focus on event visualization with an eye for relatively dense data display. Their goal is both to facilitate awareness of the global network status and to subsequently explore the dataset. Some solutions can be also seen as extensions of the category presented in the previous section called *Monitored Consoles*. Among the solutions, we note NvisionIP, which shows activities between pairs of IP addresses and the similar but more abstract SeeNet [132, 45]. Colombe et al. present an interesting data representation for visualization. The idea of the display is to tap into the user visuospatial pattern recognition skills (rainbow

palette coloration, position on the screen, etc)[79]. PortVis ([153]) is also an interesting visualization tool which aims at displaying network flows. The authors characterize the traffic by a tuple, each tuple representing *the activity on a given port during a given hour, through a given protocol.* This is a first step to organize information, but this choice was made to build the tool, more than to classify the data for other investigations. In all cases, the analysis they provide is based on the expert's view of the offered graphs. They give a few possibilities to traceback information from the graph to the packets[6], and are often limited to plotting useful but non sufficient statistics, like the number of alerts, the quantity of data transfers, the ports activities, etc. It is thus hard to understand the real malicious activities. These tools are designed to detect, or at least, show up anomalies, more than to understand the occurring threats.

2.3.6 Modeling

Modeling malware activities is an active research domain. Very interesting approaches have been observed in recent security conferences like the Workshop on Rapid Malcode (WORM) or the Conference on Recent Advances in Intrusion Detection (RAID). It is really interesting to analyze the malware activities by reproducing in a theoretical environment its behavior, either by applying mathematical formula (most of them are currently based on the epidemiologic domain [69, 204, 126, 245, 246, 205, 140, 68]) or dedicated simulation testbeds ([227, 96, 170, 239, 139]). Unfortunately, these models can be validated as *correct* if and only if they match (during a certain period of time) the propagation and evolution of existing threats. This directly implies to have full access to some particular dataset where this information can be easily retrieved. Such dataset does not exist however, or some very specific logs are applied as references without numerous details regarding their relevance. Some other tests have a questionable validity, as the ones based on famous datasets provided by the Lincoln Lab of the MIT in 1999 [146, 150]. This basic observation has motivated the decision to offer access to the whole dataset to all partners of the *Leurré.com* project. The modeling techniques, in addition, are limited to a few propagation strategies implemented by popular worms (see for instance the analysis of CodeRed II in [157]). As a conclusion, these techniques will really show their values when being compared to large and valid datasets from various places. They cannot replace the monitoring and analyzing steps we are considering.

2.3.7 Challenges and Personal Accomplishments

Among the interesting analysis techniques, we can also mention the personal accomplishments of individuals, who design their own environments, and share their information within a community. Many security researchers report such analysis and tools in incident

[6] It is all the more true that most of the tools presented in this section are directly working on Netflow logs.

and forensics mailing lists [74, 113]. Another approach is to organize challenges. As an illustration, the Honeynet Alliance organize a monthly challenge, called *Forensic Challenge*, which consists for incident handlers around the world in all looking at the same data (an image reproduction of the same compromised system) [35]. The jury determines who has dig the most out of that system and has managed to communicate her findings in a concise manner. This is an expert study of tools, techniques, and procedures applied to postcompromise incident handling. As the organizers say in [35], "the challenge is to have fun, to solve a common real world problem, and for everyone to learn from the process". This is a good experimental analysis, but it remains anecdotal, not perfectly rigorous in many cases, and might not reflect the current threats networks are facing. This can give, however, another hint at some malicious activities.

2.3.8 Others

Sets, Bags and Rock 'N Roll

McHugh has presented in [151] interesting concepts that have led to the creation of a series of tools called *SiLKtools* (also detailed in [105]), designed in the context of the CERT Coordination Center from Carnegie Mellon University. The author shows that there is value to use sets, in order to provide a compact way of describing and reasoning about the Internet and about traffic observed at various points on it. For example, this might be useful to consider such things as the set of external hosts that are observed performing a scanning activity during a time interval. Similarly, one might want to identify the set of users of some service provided by the local network (e.g. web services) to the outside world during the interval. The use of sets and bags allows abstracting from individual behaviors to clusters of activities. It is also important to mention that this clustering is performed on the data monitored by the CERT, that is a very large amount of packets per second. As a consequence, the clustering tools have been built on Netflows (see Section 2.3.2 for more details). This can lead to the same drawbacks as the ones previously described with other Netflow-based applications. The author however mentions a future packet to Netflow code that should be included within the *SiLKtools* to limit the Netflow perturbations. This work is important for us as it appears to be one of the first initiatives to focus on malware data analysis. It seems relevant at this stage to cite a sentence of the document ([151]), which is also a major principle of the *HoRaSis* method presented in the next chapters.

> *"Sets and set theory are abstractions that facilitate reasoning about many classes of problems."*

2.4 Summary

2.4.1 Observations from this State-of-the-Art

From the previous sections, it seems quite clear that most of the effort has been devoted to the design of efficient architectures, which:

1. capture original and specific malware activities.

2. analyze the very same particular and specific malware activities.

Some of them are very promising, and we believe that the research should go on with the same eagerness. However, we also believe from this observation that there is a clear lack of commonly shared information. Many research studies aim at designing tools and architectures, without proving first that there is high value in doing so. In other terms, the security community fails to find concrete validation examples. Furthermore, the global understanding of malware activities is still unknown. We can cite for instance the relationships between attacks and networks, the order of magnitude of different malware propagating in the wild, their localization, etc. An illustration lies in the numerous articles and publications which aim at detecting large sweeping scans [217, 135]. The first such algorithm in the literature was that used by the Network Security Monitor (NSM) [109], which has rules to detect any source IP address connecting to more than 15 distinct destination IP addresses within a given time window. Such approaches present the same limitations, that is, once the window size is known it is easy for attackers to evade detection by simply increasing their scanning interval. Snort implements similar methods [209]. Does it seem relevant to activate this Snort feature?

The previous approaches have interesting approaches but the major problem remains that there is no available information to work on. Furthermore, once data is collected from the honeypots, there is no existing technique or framework which helps at grabbing the useful information by taking benefit of their particular property.

Finally, we want to recall an interesting event: the European IP Network RIPE, in charge of allocating IP addresses and administrating AS blocks organized in the course of year 2005 a global meeting between ISPs. One of the most important conclusions was [152]:

> *Attack fingerprint sharing and similar mechanisms need to be further researched, developed and deployed to combat the existing threats.*

2.4.2 First Conclusions

The honeypots do not, conceptually speaking, represent any major breakthrough and are certainly no rocket science. However, they bring an important set of information, only dedicated to malicious activities. The interpretation of this dataset is then different from the one performed by IDSs, as this drastically reduces *a priori* the false positive rates. The current techniques are only based on acquired technologies, one being the traffic analysis in terms of flows, the other one being simple statistics. This leads to wonder to which extent it can be interesting to develop a specific analysis for that type of information. In addition, this implies not to stay in a global perspective, like telescopes, but to have a nearer, more refined view of the attacks from a local perspective. This complementary approach might reveal original threats which cannot be observed otherwise. And vice-versa.

The next chapter aims at describing the distributed environment which has been deployed in order to gather more local information and at presenting the corresponding dataset which has been collected. We feel it is important to let the reader understand where the information is coming from, and how it has been stored during the three first years of the *leurré.com* project. We will also show how we designed, built and deployed this environment, and how it complements and addresses the weaknesses of the previously described approaches.

Chapter 3

The Information Generation

An expert problem solver must be endowed with
two incompatible qualities,
a restless imagination and a patient pertinacity.
(Howard W. Eves)

3.1 Introduction

As the state-of-art presented in Chapter 2 has highlighted it, there is a need of data and information to start acquiring a global knowledge of malicious activities that occur in the wild. Archives of traffic data over a long period of time are rare and difficult to get access to due to privacy laws or data security concerns. For those which exist, we note a lack of details concerning their origins, the challenge and costs of handling large amount of data, and a potential interference with current network operations and accounting (mix of production and unexpected traffic).

This section aims at explaining how it has been possible, over several years, to collect meaningful data from honeypot platforms placed in different networks and countries, thanks to the success of the *Leurré.com Project*. This section could have been introduced in an experimental section at the end of the document, but we think it is important for the reader to understand the richness of the data and its current uniqueness before describing the analysis theory. The reader who is already aware of the *Leurré.com Project* dataset can skip this Section and move ahead to Chapter 4.

3.2 The Leurré.com Project

3.2.1 The Objectives

The project we have launched aims at disseminating similar honeypot sensors everywhere thanks to motivated partners, on a voluntary basis. Partners are invited to join this open project and install a honeypot sensor on the premises of their own networks. We, at Institut Eurécom, take care of the installation by furnishing the sensor image and configuration files. Thus, the installation process is automatic. In exchange, we give the partners access to the centralized database and its enriched information. A dedicated web site has been developed to make research faster and more efficient. The project has started triggering interest from many academic, industrial, and governmental organizations. As of this writing, around 35 platforms are deployed in 25 different countries covering the five continents. We keep installing new ones regularly.

3.2.2 Principles

On the choice of a honeypot sensor

The deployment of honeypot sensors in a variety of places requires first to choose the most appropriate sensor types. As it has been detailed in Section 2.2.1, the security community often distinguishes two major categories. First, there are sensors running on real systems (OSs, services, users, etc). They belong to the *high interaction* category. Secondly, others exist, which interaction is limited to a few emulation scripts. They belong to the *low interaction* category. Each of them presents interesting advantages but also limitations, that are summarized in Table 3.1 extracted from [212]. In the row entitled *Work to Deploy and Maintain*, one finds the time required to run and to maintain the honeypot. In the *Knowledge to develop* one, one sees the amount of prerequired knowledge to build a honeypot environment. The *Compromise Whised* row express the expected goal of the honeypot, that is, if it aims at being compromised, or in a more restricted and mode, if it aims at collecting malware traffic without letting an intruder enter the system . The *Level of Risk* row is an indicator of the risk run when implementing a honeypot into a system.

It seems straightforward, from Table 3.1, that *Low* interaction is adapted to the requirements of the Project described in previous Section 3.2.1. Indeed, partners can be from any country, and the tasks of deployment and maintenance of the sensors must remain acceptable. Furthermore, the risks must be as low as possible to motivate partners

Table 3.1: Level Interaction and Honeypots

level of Interaction	**Low**	**High**
Information Gathering	Connection Attempts	All
Work to deploy and Maintain	Easy	Difficult
Compromise Wished	No	Yes
Knowledge to Develop	Low	High
Level of Risk	Low	High

in joining the project. It seems hard to imagine asking industry partners to join the project and introduce new vulnerabilities in their network.

Low interaction is the most sensible choice at this stage. Unfortunately, information which is collected from it might differ from real systems. More generally, it is important to qualify and quantify the amount of information that might differ when using one of the two interaction sensors. We have published a detailed comparative analysis in this direction in [187]. It is briefly summarized in the next section.

A comparison between Low and High Honeypots

We have described in [187] two distinct honeypot platforms. They have been called H_1 and H_2. H_1 is a *high interaction* honeypot, running three different OSs and various services. H_2 is a *low interaction* honeypot, based on an open tool called *honeyd* [9]. H_2 has been configured in a very particular way: we have scanned the open ports in H_1 and opened the very same ones in the honeyd configuration file for each of the three virtual machines. Some service scripts that are available in [9] have been linked to open ports, like port 80 (web server) or port 21 (ftp). As a consequence, H_2 can be seen as offering a similar yet simplified behavioral model as H_1. We connect every day to both host machines in the same way to retrieve traffic logs and check the integrity of chosen files. Data is then stored in the very same database (described in Section 3.5.3).

The paper reports a comparison over 3 months of data. The results show in particular that:

1. Both approaches provide very similar global statistics based on the collected information. High-interaction honeypots are more or less attacked the same way than low interaction ones.

2. A comparison of data collected by both types of environments leads to an interesting study of malicious activities that are hidden by the noise of less interesting ones. One example has been the discovery of scans targeting one out of two successive IPs [187]. Another example is that 3% of activities which have targeted only two virtual machines out of the three have precisely targeted two windows machines on

both environments H_1 and H_2.

3. This analysis highlights the complementarities of the two approaches: a *high interaction* honeypot offers a simple way to control the relevance of *low interaction* honeypot configurations and can be used as an effective *"etalon system"*. Thus, both interaction levels are required to build an efficient network of distributed honeypots.

Algorithms have been described to make this comparison automatically, but we report the authors to [187] for more details and illustrations of these contributions. The important result worth keeping in mind is that:

Lemma: The collect is not biased by the use of *low interaction* honeypots in a distributed sensor network. Furthermore the deployment of a few number of contiguous *high interaction* ones can help controlling the relevance of the collected information.

In the next section, the configuration which has been used to build H_2 is detailed.

3.2.3 Honeypot Sensors

The sensors which have been deployed along with the *Leurré.com* project are based on several open source utilities, which emulate operating systems and services. The basic building block used is honeyd [23]. The sensor only needs a single host station, which is carefully secured by means of access controls and integrity checks. This host implements a proxy ARP. This way, the host machine answers to requests sent to several IP addresses. Each IP is bound to a certain profile (or personality in the honeyd jargon). Thus, the emulation capacity of the sensor is limited to a configuration file and a few scripts[1]. The sensor we are using emulates three Operating Systems: Windows 98, Windows NT Server and Linux RedHat 7.3, respectively[2]. Some service scripts that are available in [23] have been linked to open ports, like port 80 (web server) or port 21 (ftp), among others. A simple sensor architecture is presented in Figure 1. Finally, we connect to the host machine to retrieve traffic logs and check the integrity of the system files every day. Next sections aim at presenting the global data which has been collected so far and its particular storage at the Institut Eurécom.

[1]New emulation scripts have appeared during the last months of this thesis: *FakeNetBIOS* emulates traffic on ports UDP 137 and 138 only [66]. Scriptgen described in [138] generates in an automated and clever way scripts derived from tcpdump traces. They were not available when we started deploying the sensors.

[2]These OSs have been chosen three years ago to mimic and thus to be compared with other high interaction honeypots already emulating these OSs.

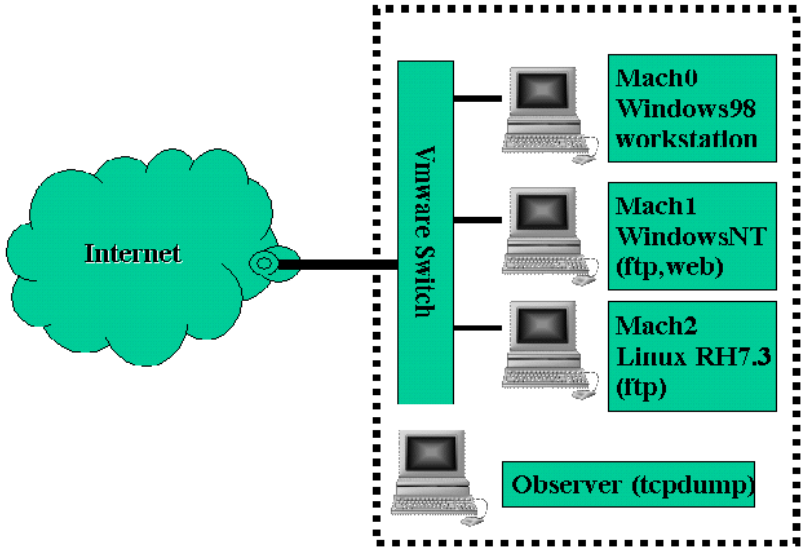


Figure 3.1: Architecture of a Honeypot Sensor

3.3 Global Picture

3.3.1 First Figures

Some platforms have started running in February 2003. Each month, new partners come and increase the volume of data. Some global statistics are listed here.

- Number of observed distinct IP Addresses: 989,712
- Number of received packets: 41,937,600
- Number of emitted packets: 39,911,933
- Total number of collected packets: 81,849,533
- Number of received TCP packets: 74,428,652, that is 90.93% of all packets
- Number of received UDP packets: 635,363, that is 0.77% of all packets
- Number of received ICMP packets: 4,218,109, that is 5.16% of all packets
- Others: (malformed packets, etc) 2,567,409, that is 3.14% of all packets

In short, with a maximum of platforms up and running at this time writing, it is possible to observe more than 5000 new IP addresses per day, and collect 100000 new packets issued by these IPs. This represents an important volume of data, and the numbers increase each day, as new partners join the project. This explains in the next chapter the need to classify the data in such a way that any kind of lookup can be efficiently performed.

3.3.2 First Analyses

This section aims at presenting global statistics, which are the historical motives of this work. Some results have been described in [88, 89]. We present here four interesting data representations which are part of the initial observations we made.

Average Number of Attacking IPs per Honeypot Environment

Figure 3.2 shows the average number of attacking IPs observed each day and per environment. Values are definitely not uniform, and it is important to notice that some environments (identifiers 12 or 5) can be attacked almost 100 times more than others (identifiers: 20 or 32).

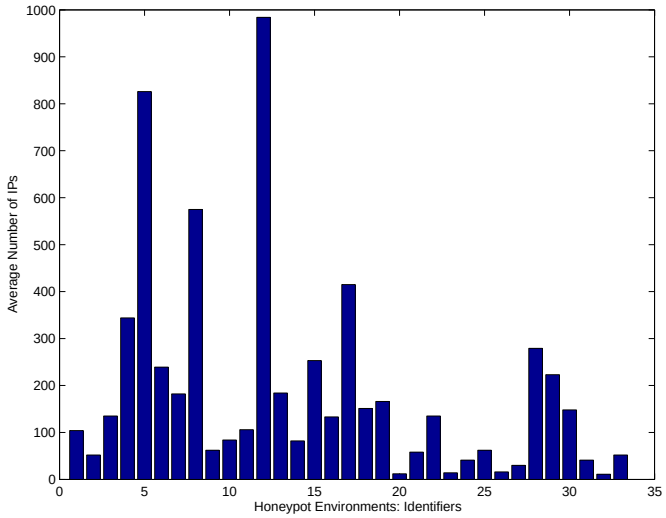


Figure 3.2: Average Number of Attacking IPs per Honeypot Environment

Number of IP Sources observed per Day and per Environment

Figure 3.3 shows the number of IP addresses observed per day and per environment. We have represented three different environments for clarity concerns, but the others present similar characteristics. Dates are comprised between February 1st 2005 and March 31st 2005. We note here that a given platform might not observe the same number of distinct IPs over days, and some high variations can occur. We also note that for a given day, three platforms can observe very different numbers of distinct IP sources.

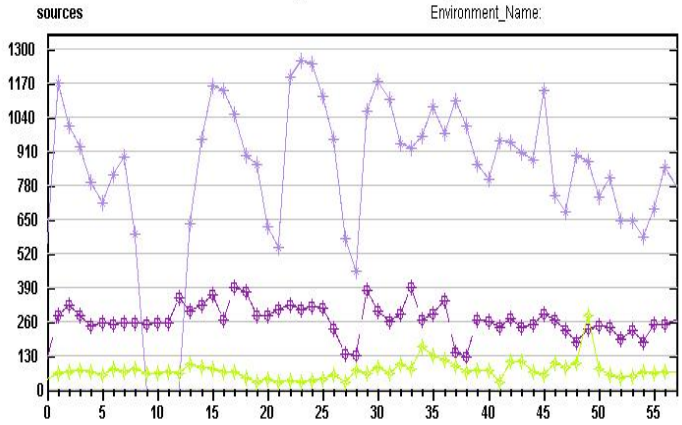


Figure 3.3: Distinct IP Sources Observed per Day on Three Sensors

Average Number of Bytes sent by Attacking IPs per platform (TCP payloads)

Figure 3.4 gives the average data payload[3] that have been observed in average on all platforms between May and July 2005, when all platforms have been up and running for several days. This is of course not completely meaningful, as it is also highly dependent to the emulation level of the honeypot sensors. It however indicates that some environments present very strong differences compared to others, in terms of received bytes.

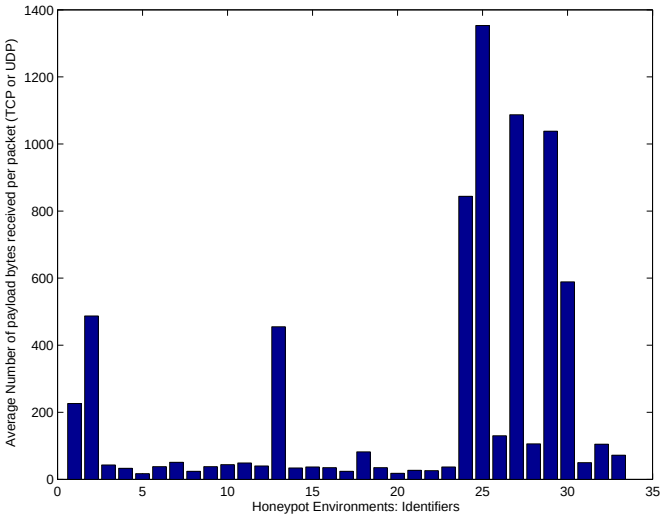


Figure 3.4: Average Number of Bytes sent by Attacking IPs per Platform (TCP payload)

[3]Data payload includes all data for layers 4+, except TCP and UDP headers (with options for TCP).

Average Number of Attacking IPs per hour (capture time)

Figure 3.5 shows the cumulative number of attacking sources that have been observed in different hours during the day on any environment. We consider here the whole dataset. For instance, the first column gives the number of distinct IPs which have sent their first packet to an environment between midnight and 1am, the second between 1am and 2am, etc. The considered time is the capture time, that is, the time on the environment observing the activity. These simple statistics show a strong temporal pattern, with about third less attacks during night hours than working hours. This statistics have also been produced for several periods during the year (over months, 2-months and 6-months), but they show the very same property than by looking at the global dataset.

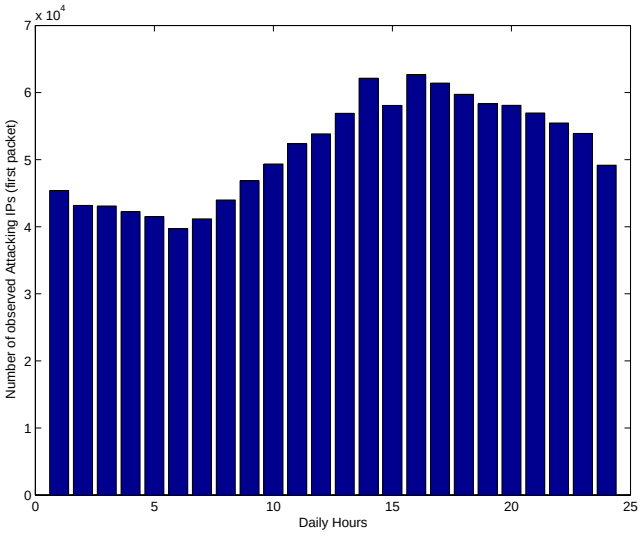


Figure 3.5: Average Number of Attacking IPs per Hour (local time)

3.3.3 On the Advantages of Local Distributed Sensors

The first graphs presented in Section 3.3.2 indicate in a clear manner that sensors collect activities which might differ from sensors to sensors. They are not facing same activities. From another point of view, Figure 3.6 presents on one side the activities on port 445 as shown by a web site (Dshield [14]) during September 2004, to be compared with the activities collected on one of our sensors during the same period in Figure 3.7.

The peak observed on September 26th does not appear in any Dshield reports or mailing list posts. The reason for this has not been investigated further by the partner. The *Leurré.com* dataset contain numerous similar examples. Nevertheless, it clearly shows that local observations might differ from global trends. This claim is also defended in [82]. The authors demonstrate differences observed in *class A* IP ranges and smaller

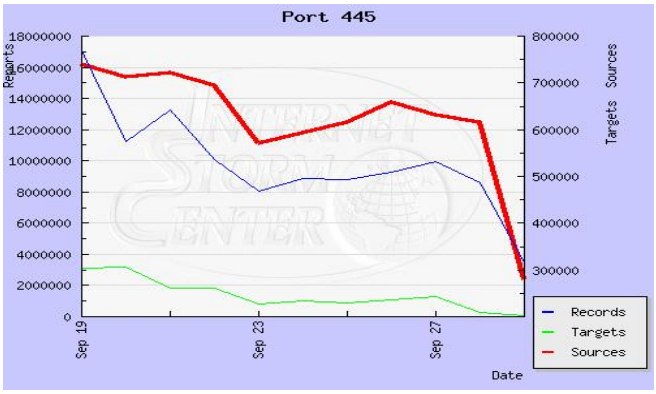


Figure 3.6: Dshield vs Leurré.com data: Dshield [14]

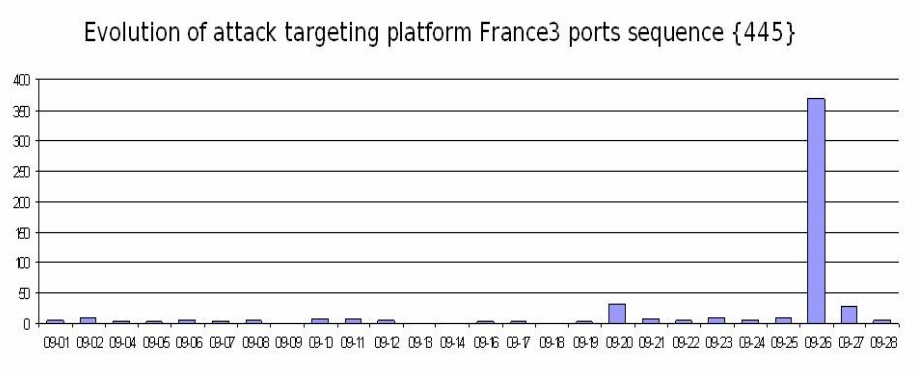


Figure 3.7: Dshield vs Leurré.com data: Leurré.com data

subnetworks along at least three dimensions: over all protocols and services, over a specific protocol and service and over a particular worm signature. This leads to the two following lemmas, considered for the different analyses we intend to perform in the next chapters.

Lemma 1: Local observations of malicious activities DO NOT bring same information as more global techniques. This architecture is thus a complementary and essential approach.

Lemma 2: Local sensors share some common similarities, but the global statistics also indicate strong dissimilarities w.r.t. monitored activities.

3.3.4 First Discussions

The analysis of *Leurré.com* data could stop here. The previous sections have demonstrated the usefulness of deploying local sensors, and the new dimension it can bring for monitoring malware activities. Other similar examples have been detailed in our earlier publications [186, 192]. However, these first results also highlight the motivations for further analyses.

It would be interesting for instance to characterize the differences between platforms. Current techniques are however quite limited to directly understand the exact meaning of these differences and the relationships between them. The *HoRaSis* framework should help improving the knowledge acquisition and the extraction of new information in an automatic way, that is, it should perform analysis steps further than simple statistics.

3.4 Observation Positioning

3.4.1 Sensors Limitations

One first remark goes to the deployment of sensors. It has been shown that their deployment brings interesting and complementary information. However, we have not presented the impact of their configuration to the quality of the data collection [54]. This is another dimension which is not taken into account in the following sections. A few preliminary experiments have been considered in the course of this work and reported in [192], but it is admitted, in the following, that the whole experiments have been applied on a unique sensor configuration. It is also worth mentioning that sensors are, and this is not a real surprise, not perfectly undetectable. None of the existing honeypot-based systems are, and honeyd, the software on which the *Leurré.com* sensors are built, follows this rule. As an illustration, an experiment has been presented by Kohno et al. in [129]. They have presented a technique to remotely fingerprint hardware devices via remote clock skew estimation, and tested it on honeyd platforms. The sensor time is not maintained via NTP or SNTP, and the fingerprint of honeyd would be possible by sending ICMP Timestamp Requests (type 13) against honeypot sensors. Therefore, we have carefully looked for such packets and have observed only 38 of them so far. They are unlikely due to this detection mechanism, as they have been observed one year before the first public reference to that problem. Another bug has been found in the early versions of honeyd (< 0.8, [9]), which has not been used. A system running honeyd can be detected as it replies to invalid TCP packets (with SYN and RST flags) -which it should not.

In a general manner, this has to be related to the work presented in [187]. Such risks can be minimized by frequently comparing the sensor captures with other "etalon systems". Attempts to fingerprint the honeypot sensors would also be interesting insofar as it would indicate that this monitoring disturbs some particular communities.

3.4.2 About Non-Observable Malicious Activities

Discussion

As it has been said in the previous section, sensors implement the very same honeyd configuration. There might be some attacks the sensors will not monitor due to their configuration. It is by nature impossible to catch every malicious activity in the Internet, but it is expected to have a very good overview of the major threats with quite standard machine configurations.

In addition, the environment will not observe a unique attack, dedicated to a target which is not a *Leurré.com* sensor. It does not pretend to be a perfect early warning system. However, such a distributed system could help identifying the common activities, also called background radiations in [171][4] and to detect new threats and activity changes monitored by the different environments.

Worm Propagation Strategies

There are now many worm species, and some books have already started building *phylogenetic* classifications. We can cite as examples the noteworthy study of Szor et al. in [223], or the one of Filiol et al. in [100]. As they both explain, *worms* are network viruses replicating on networks. They all present a large diversity of spreading strategies. Among them, we note:

- *Local-Subnet propagation*: it involves worms scanning for vulnerable hosts in a class-C or smaller subnet. It usually increases the number of infected machines more quickly, as the worm can find less protected machines and a less heteregeneous network environment once the firewall is bypassed. This technique has been used by the Code Red II [5] and Nimda worms for instance [228]. If the worm limits its propagation to a very small subnet that is not covered by a honeypot sensor, its activities will remain unobserved by the proposed architecture. On the other hand, the risk remains for the same reason limited to the uncovered subnet.

- *Hit list propagation*: This technique is applied when the worm propagates based on a list of victims. This list is given by the attacker, either by a hard-written list of IPs/networks, or by collecting information from publicly available resources. A theoretical worm, named Flash Worm, propagating this way, has been studied by Staniford et al. in [216, 215].

[4] Background radiation reflects fundamentally nonproductive traffic, either malicious (flooding backscatter, scans for vulnerabilities, worms) or benign (misconfigurations) in [171].

[5] Code Red II has three propagation strategies, one being to favor local class-C subnets [157].

Activities will never be detected by one of the *Leurré.com* sensors if the propagation is too local. This will also be the case of propagations over a restricted hitlist.

3.5 Data Storage

3.5.1 A Need

Data logged by each *Leurré.com* sensor is copied to a centralized machine. The sizes of the logs highly depend on the sensor and the activities against it. Figure 3.8 represents the cumulative size of logs collected during the considered period.

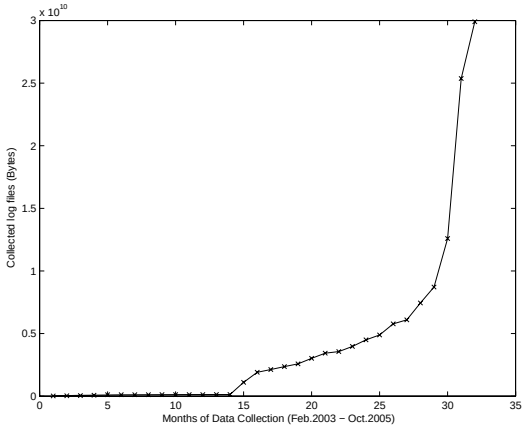


Figure 3.8: Cumulative Log Size Collected from the Sensors

The traffic is limited during the first months of 2003 to a few dozens of Megabytes per months, as the only sensors working at this time were the ones implemented at Eurecom. The regular increase in 2004 corresponds to the first phases of the *Leurré.com* project and its first partners. The steep increase in 2005 is the manifestion of the keen interest to the project from multiple communities. It seems important, from Figure 3.8, to organize data in an efficient way, in order to query it easily. The data organization is described in the next paragraphs.

3.5.2 Definitions

This data needs to be properly organized, as it will be used for further analysis and experiments. In theory, no traffic should be observed from the machines we have set up. As a matter of fact, many packets hit the different virtual machines, coming from different IP addresses. Typically, if an attacker decides to choose one of our honeypots as her next victim, she tries to establish direct TCP connections or to send UDP, or ICMP, packets

against it. She can behave differently when targeting each of the three virtual machines. As a consequence, we distinguish in the database three major classes of information:

1. Information that characterizes the attacking source. It includes its IP address, the date it has been observed, the domain and geographical location associated to this address,etc.

2. Information that characterizes the behavior of the attacking source against the global sensor. It includes the number of virtual machines it has targeted, the global duration it has been observed on it, the way it has targeted the virtual machines (sequence vs. parallel), etc.

3. Information that characterizes the behavior of the attacking source with respect to a single virtual machine. It includes the sequence of ports that have been targeted on that machine, the data sent, the number of exchanged packets, etc.

For the sake of conciseness, we do not want to describe the full database architecture here. All details are precisely described in [184]. We just want to point out that most of the comparisons that are presented in the following rely on this efficient way to organize the information. This organization leads us to frequently make use of the following four definitions that derive from the previous classification.

Definition 3.1. ***Source:*** *A* Source *corresponds to an IP address observed on one or many platforms, and for which the inter-arrival time difference between consecutive received packets does not exceed a given threshold (25 hours). The time difference is computed by converting all times to GMT.*

As an illustration, two packets observed at "2005-02-17 10:00:00 GMT" (Sensor A) and at "2005-08-05 13:00:00 GMT+5" (Sensor B) which share the same IP source address will be associated to two distinct attacking Sources.

Definition 3.2. ***Global_Session:*** A Global_Session *is the set of packets which have been exchanged between one Source and all Honeypot Environments of the* Leurré.com *distributed monitoring system.*

Definition 3.3. ***Large_Session:*** *A* Large_Session *is the set of all packets which have been exchanged between one Source and a particular Honeypot Environment (sensor).*

Definition 3.4. ***Tiny_Session:*** *A* Tiny_Session *is the set of packets which have been exchanged between one Source and a single Virtual Machine. As each honeypot Environment is made of three virtual machines, a Large_Session is associated to 1, 2 or 3 Tiny_Sessions.*

3.5.3 ER diagram

A dedicated database has been designed to store the information at different abstraction levels. The UML diagram in Figure 3.9 offers its over simplified structure. Many tools are used to enrich the data. For instance, for each Source, we look for, and include in the database, its geographical location (Maxmind, Netgeo, IP2location [11, 63, 117]), its passive OS fingerprinting attack (p0f, ettercap, disco [24, 7, 4]), its name by means of domain name lookups, etc. The details are carefully described in [184]. A more complete view of the UML diagram, including important attributes, is presented in Annexe A.

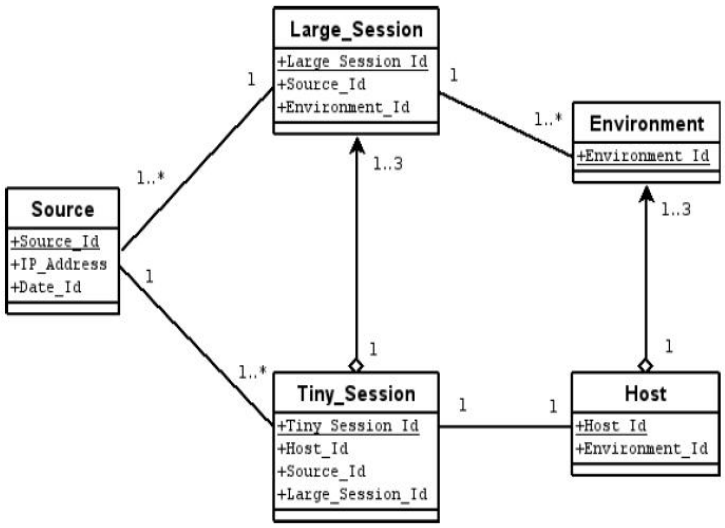


Figure 3.9: UML Diagram: Relationships between Definitions

3.5.4 Web interface

Two distinct web interfaces have been developed along with the *Leurré.com* project. One is the public project web site, www.leurrecom.org. It presents some global statistics, without mentioning any partner name nor IP address [6]. The partnership offer is described on the site and many papers are available on it. The other interface is a protected GUI to the database, with a personal access for each partner. Some useful queries are implemented to ease the task of the partner. A direct access to the database has also been made possible for more personal or complex queries. Both are briefly presented in Annexe B.

[6] A Non-Disclosure Agreement has been signed by all partners to keep such information confidential.

3.5.5 Collection Issues

Sensors Stability

Honeypot sensors are not perfectly stable. They might be down for some days for several reasons, like electrical problems, network changes or human incidents (powering off, etc). This introduces a bias in the data collection and analysis, as missing data can have two different meanings: either there was an important decrease of the attack during a period, or the platform was not working. It is important to distinguish between these two scenarios for the analysis. The missing logs are reported in a dedicated table of the database. We note that 10% of log files are globally missing. To address this issue which might impact global statistics, it has been implemented an interpolation technique called Cubic Spline. This name comes from the fact that this procedure closely approximates a technique that has long been used by draftsmen. A draftman who wishes to plot a smooth curve through a set of $n+1$ observations will place a set of weights on a thin elastic rod called a spline. The weights are placed in such a way that the rod passes over each of the observed points. The draftsman then traces the curve formed by the rod. The theoretical details can be found in [43]. As many other techniques, an interpolation cannot be perfect, except in rare cases (mathematical functions), and it is hard to estimate the error. However, the cubic spline interpolation has some interesting features, compared to other techniques:

- Splines are smooth and continuous across an interval. A polynomial, for instance, fitted to many data points, could exhibit erratic behavior.
- The spline curve interpolates the data while remaining within the range of the dataset.
- Splines are piece-wise defined functions whose individual curves meet at the points.
- The splines not only interpolate the data but match the first and second derivatives at the points.

The set of points are called the knots. The set of cubic splines on a fixed set of knots, forms a vector space for cubic spline addition and scalar multiplication. An example is shown as illustration in Figure 3.10 .This Figure represents on the light curve an erratic function ($y = (sin(x) + cos(x))^{\frac{3}{4}}$), while the bolder curve has been generated by connecting data points (the *knots*) along the above line with a cubic spline function. While the fit is not perfect, it does closely approximate the function without a great degree of divergence. Future work will consist in adapting other interpolation functions which would suit more closely the properties of the curves under study.

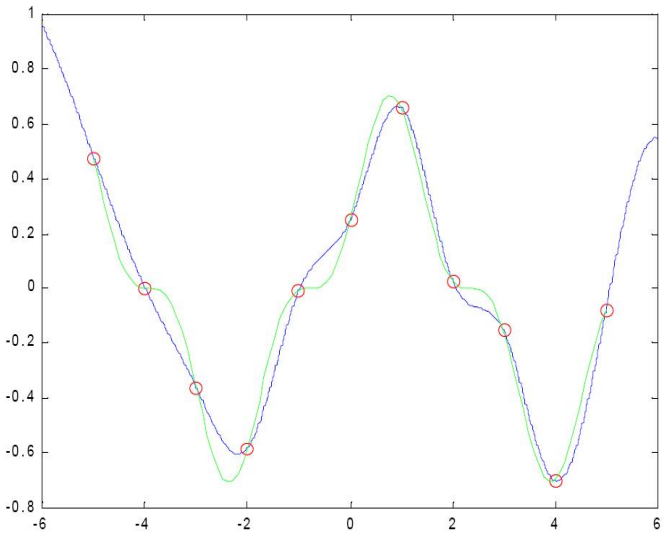


Figure 3.10: A Cubic Spline Interpolation of $y = (sin(x) + cos(x))^{\frac{3}{4}}$

Data Synchronization

Another potential issue is the Synchronization problem between sensors, as it has been reported by Lamport in [133]. For security concerns, there is no Network Time Protocol (NTP) daemon running on the platforms. Thus, comparing the activities on the sensors consists in determining the time lag between sensors. It is performed by performing periodic *date* commands which results are then stored in the database. The rate error of each sensor is quite constant over months and never exceeds a few seconds per month.

In the other chapters, we work on the stored data, and for some similarity studies, make use of the Cubic Spline interpolation technique. When done, this is explicitly mentioned. Furthermore, the synchronization is not a major issue in the results presented in this thesis, as the analysis we present is not bias significantly by the desynchronization effects.

3.5.6 Conclusion

Figure 3.11 presents in a simple diagram the different steps that are followed, from the tcpdump packets fetched on each sensor to the global database contents which are the foundations of the next chapters. The database is precisely built around a small number of information categories, called *Sources*, *Global_Sessions*, *Large_Sessions* and *Tiny_Sessions*. These definitions described in Section 3.5.2 are a first attempt to organize the collected data and start making comparisons and analyses. The generalization scripts presented in Figure 3.11 aim at deriving the attributes of these new information levels from the raw packets tables. We have also considered several tools to enrich this information, some of them being commercial solutions, others being open source software or hand-made scripts.

The database architecture is flexible enough to insert *a posteriori* results from other tools which are not considered today.

The data collection and storage were a preliminary and intuitive work, that has been enriched all along the project. The obtained dataset is quite unique and represents several years of data. It can be wondered whether valuable information can be extracted from such a dataset, and how should the analysis be performed to do it. The first statistics tend to indicate that sensors collect similar but also different traffic. As a consequence, it would be interesting at this stage to distinguish the different activities which are monitored. The next chapters precisely justify a way to analyze data and characterize the activities observed on the *Leurré.com* sensors. We are following, as expected from the Introduction, the directions to build an *HoRaSis* framework.

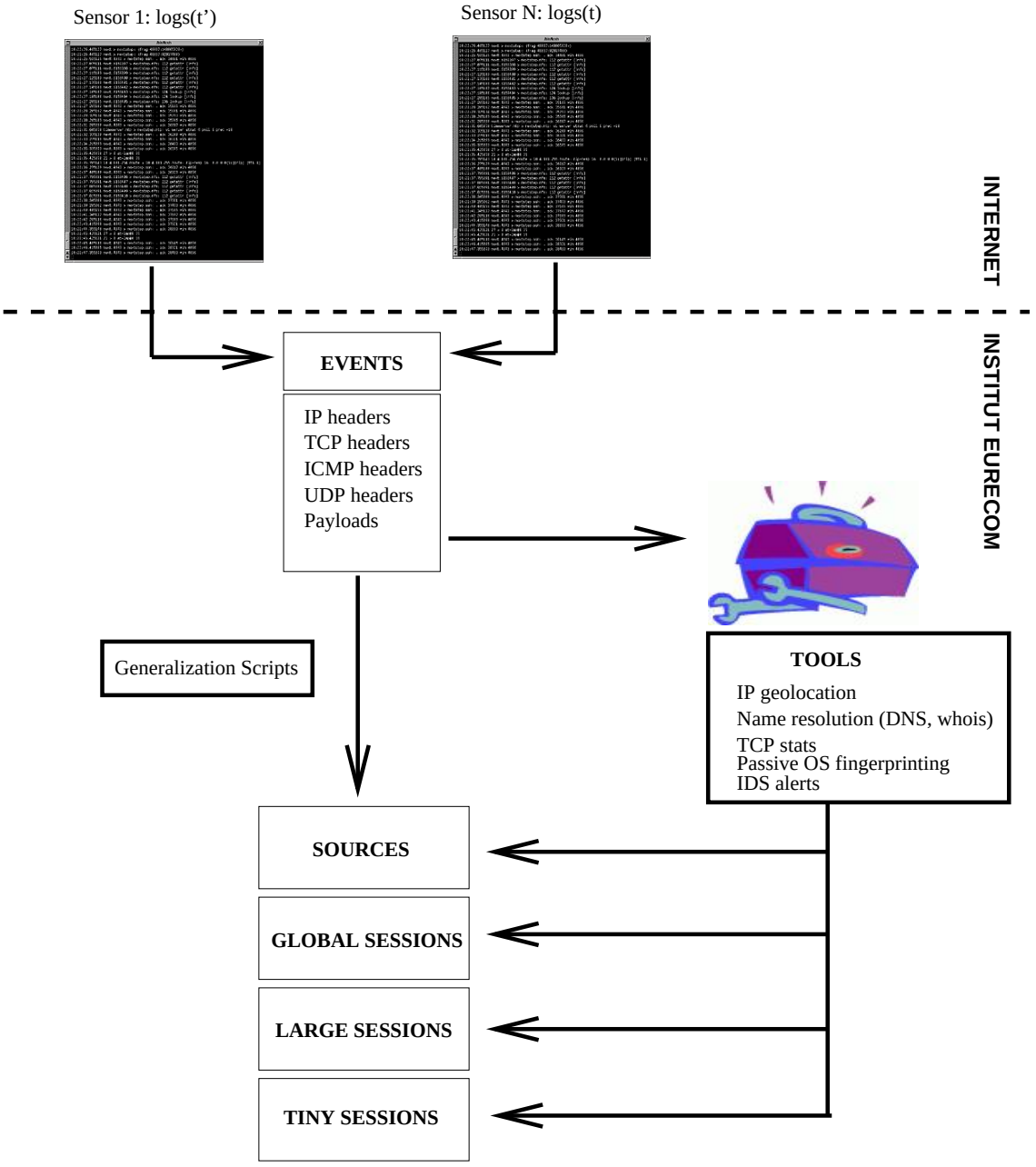


Figure 3.11: From Dump Logs to a Structured Database

Chapter 4

Discrimination Step: Fingerprinting Activities

Problems worthy of attack,
prove their worth by hitting back.
(Piet Hein)

4.1 Introduction

4.1.1 Need for Classification

In the *Leurré.com* Project, traffic is collected from each honeypot sensor with tcpdump. The dump files contain plain raw packets. Information as such is not intuitive. A first step is to store packets according to their respective protocol levels, as it has been detailed in Chapter 3. Packets by themselves are also not really meaningful, as their analysis remains fastidious. We presented in Section 3 some global statistics on packets, but the analysis at this stage remains at a too coarse level. We also observed when storing such packets into the database, that grouping packets according to their origins and destinations is helpful. This led us to create four distinct information levels, called *Sources*, *Global_Sessions*, *Large_Sessions* and *Tiny_Sessions*[1].

They represent all four different abstraction levels. By choosing different abstraction levels we can switch between levels and analyze the appropriateness of the abstraction for a specific situation.

[1] Complete definitions are in Chapter 3.

Abstraction mechanisms are often complex, and provide means for identification and design of invariant components and structures [206, 201, 210]. One of them is *classification*, often complemented by two other underlying mechanisms: *generalization* and *specialization*.

In the following, we present a generalization process, that will lead to a useful abstraction level of the data. This abstraction level generates *clusters* that will be the basis of the automatic analysis process that we present in Chapter 5.

4.1.2 Concepts and Challenges

Previous work has shown that address blocks in different networks can see different traffic traces [82, 186]. Furthermore, the global statistics extracted from the database (see Section 3.3.3) also indicate that each sensor presents unique properties compared with one another, in terms of the number of observed IP sources, targeted ports, received bytes, etc. Thus, sensors do not monitor exactly the same '*events*'. To exemplify this property, we make use of the following terminology:

Definition 4.1. *An **Activity** is the set of actions performed by an IP source on a honeypot sensor.*

An activity can be characterized by a given *Large_Session*. We remind here that a Large_Session is the set of packets exchanged between an IP source and a honeypot sensor. So, rephrasing the previous remarks, we have observed so far that:

Observation: Honeypot sensors do not monitor exactly the same activities.

The *HoRaSis* framework aims at better understanding the activities that are monitored through the distributed network of sensors. One implication would be to compare somehow the activities on each sensor, in order to determine what makes them differ and what kind of information this can bring. The *HoRaSis* framework must contain a functionality that helps at comparing *activity fingerprints*; an *activity fingerprints* being defined as:

Definition 4.2. *An* activity fingerprint *is a set of parameters that characterize an activity observed on a honeypot sensor.*

The set of parameters we chose to perform this study is detailed in the next section. Continuing previous reasoning, it seems reasonable to think that attacking tools, if they consist of purely automatized deterministic activities, should generate the very same *activity fingerprints* on all targeted sensors. This leads us to formulate the following assumption:

Assumption: If the attacking tool has a deterministic *behavior*, we must observe the very same *activity fingerprint* on all sensors which have been the target of this attacking tool.

Most of the tools have, as far as we observed, no random *behavior*, and share this deterministic property[2]. In Section 4.2 we describe the set of parameters used as *activity fingerprints*. In Section 4.3, we explain that this theoretical *activity fingerprint* might differ due to network distorsion, e.g. losses. These phenomena are not part of the fingerprint and then must be considered when comparing the *activities*. We have developed dedicated algorithms to group efficiently the IP sources sharing an identical *activity fingerprint* while considering these network distorsions.

The analysis we intend to perform requires to group malicious activities (in a general sense) that share a common fingerprint. *Clustering techniques* are natural candidates for this task. Note that the proposed solution might not be (and does not pretend to be) the unique one. We make use of techniques steering from a large variety of research domains, from knowledge discovery to data mining, and other solutions might be possible. The goal here is to present, based on the experience gained with our data, *HoRaSis*, a simple but meaningful technique to organize and classify data. This method is validated and produces, at each step, new interesting results which contribute to the final analysis.

4.2 Fingerprints of Activities

4.2.1 Definitions

We make use, all along the thesis, of the terms *activities fingerprint* and *cluster*. They must so be carefully defined:

Definition 4.3. *A* Cluster *is a set of IP sources having exhibited the same* activity fingerprint *on a honeypot sensor.*

Definition 4.4. *We copy here the definition of* fingerprint *found in [80]. A Fingerprint is:*

1. *An impression on a surface of the curves formed by the ridges on a fingertip, especially such an impression made in ink and used as a means of identification.*
2. *A distinctive or identifying mark or characteristic: "the invisible fingerprint that's used on labels and packaging to sort out genuine products from counterfeits" (Gene G. Marcial, [80]).*

[2]We discuss in Section 4.6.2 situations when this assumption might not be valid.

3. a DNA fingerprint, a chemical fingerprint.

We are interested in the second and generic definition. The fingerprint should not be an add-on. A fingerprint is a unique set of parameters that allow characterizing an object. There might be many fingerprints for a given object (just like the eye , DNA or thumb, etc for human beings), but it is important to remark that the combination of several fingerprints remain a fingerprint. Here, we propose to identify fingerprints of attack tools from the honeypot datasets, in order to analyze and correlate them, to understand attack processes and detect new threats, etc. We also show in the following sections the first advantages this classification brings to the analysis. Based on the previous remarks, we can define an *Activity Fingerprint* as:

Definition 4.5. *An* Activity Fingerprint *is an analytical evidence that characterizes a specific malicious activity on any sensor of a group of IP sources.*

4.2.2 Analytical Evidence

We want to define the fingerprints of an attack in terms of a few parameters. In a passive manner, all the info we receive constitute the fingerprint. However, we need to determine the dimension of this fingerprint. We base this step on our own experience of traffic monitoring, and on techniques commonly used for network monitoring. This leads us to define the following list of attributes:

1. *The number of targeted virtual machines on the honeypot platform*
2. *The sequences of ports:* From the ordered packets (received time) sent to one virtual machine, we extract the exact sequence of distinct targeted ports. Figure 4.1 illustrates the definition.
3. *The total number of packets sent by the attacking source*
4. *The number of packets sent by the attacking source to each honeypot virtual machine*
5. *The duration of the attack*
6. *The inter-arrival time between packets received by the targeted machine*
7. *Ordering of the attack*
8. *The packet contents (if any) sent by the attacking source*

We can also imagine to take other parameters into account, like the *packet size*. This information is often misleading, as some protocols implement padding to build normalized size packets. This whould not allow us to discriminate several attacks using such protocols.

This is the main reason why we focused on packet payload only, as delivered to the upper layer. These seven attributes are more precisely detailed and justified in the following sections.

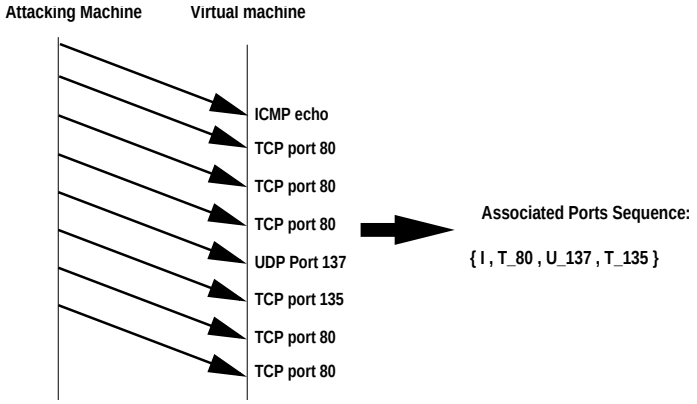


Figure 4.1: A Ports Sequence Associated to an Observed Activity

4.2.3 Classification Requirements

The analytical evidence of an *activity fingerprint* is made of all concrete attributes previously listed and observed directly from collected data. They do not require particular tools (only traffic sniffing and simple computations) to compute them. An offensive tool can have different *activities*, depending on the way it is configured. We want to find clusters, i.e. groups of IP sources that share the same *activity fingerprints*.

Preliminary analysis of the traffic we have scrutinized has revealed that network disturbance might affect in some ways the *activity fingerprint*. Obviously, the clusters we intend to obtain must not be biased by these disturbances which are not directly related to the attack activity. As a consequence, the *HoRaSis* framework must contain a classification mechanism for the IP sources which consider this problem.

Generalization process has been used to characterize DoS attacks in [115], and it is, as far as we know, the only publicized effort which is going so far in the generalization process of the traffic. The authors identify *the attack stream*, that is the sequence of attack packets created by the host machine and the attack tool. However, the attack stream is shaped by many factors: number of attackers, attack tool, operating system, host CPU, network speed, host load, and network cross-traffic. Since they define an attack scenario as a combination of the attacker and attack tool, the fingerprinting techniques should be robust to variability in host load and network cross traffic.

The classification task consists in assigning objects to classes (groups) on the basis of measures made on the objects. Classification is *unsupervised* if classes are unknown, and if we want to discover them from the data (cluster analysis) [110]. Classification

is *supervised* if classes are predefined. In this case, we can use a (training or learning) set of labeled objects to form a classifier for the classification of future observations. In our situation, we have no predefined classes. However, as we explain below, we have a good intuition based on the experience of digging into the database on possible classes, or at least attributes to build classes. Thus, the proposed classification method must be *unsupervised but controlled*.

Clustering comes into two general flavors: Partitioning or Hierarchical [61]. Partitioning usually requires to pre-specify the number k of mutually exclusive and exhaustive groups (k-means, self-organizing maps, PAM, etc). The hierarchy-based clustering methods produce a tree or dendogram. They avoid specifying how many clusters are appropriate by providing a partition for each k obtained from cutting the tree at some level. This tree can be built in two distinct ways:

- bottom-up: agglomerative clustering
- top-down: divisive clustering

Some techniques also exist to estimate the number of clusters (silhouette width in PAM [92], Gap statistics [230], etc). In our situation, we have no indication on the initial number of classes. Furthermore, the initial dataset is the whole database, that is all packets collected so far. Our clustering technique should be in this case a *hierarchy-based and top-down* approach. It is important to point out that clustering cannot *not* work. That is, every clustering methods will return clusters. Clustering helps to group information and it is a visualization (abstraction) tool for learning more about the data.

To conclude, we intend to classify attacking Sources according to their activity fingerprints on each platform. Some clustering techniques are applied to make this grouping, and the global method, presented in the next sections, is a *hierarchy-based and top-down* approach.

4.3 Clustering Algorithm

4.3.1 High Level Description

The purpose of classification here is to group all the IP sources that share common characteristics as defined in the previous section. This task is however not as simple as it appears, for at least two reasons. First, traffic in the network is subject to a few bothering effects, e.g. losses, delays or reordering. Second, the notion of similarity associated to each parameter is not clearly defined. There exists dozens of distance functions, and others can also be generated. To deal with these potential issues, we split the clustering algorithm into four steps:

1. We withdraw all network influences from the dataset;

2. We classify the data according to deterministic parameters;

3. We cluster the sources together according to non-deterministic parameters;

4. We validate the clusters and provide a consistency attribute.

Each of these steps is presented in the following subsections.

4.3.2 Network Disturbances

Introduction

We are interested in this section in estimating the impact of some network effects, and especially losses and reordering, in the analysis of attacks. As an illustration, consider the analysis of Win32.Rbot.H which is described in [40]. Win32.Rbot.H is an IRC controlled backdoor that spreads by scanning ports 139 and 445 respectively. If reordering or packet losses occur, the sequence of ports can be altered, and so will the analysis in an indirect manner. Win32.Rbot.H could then be associated to ports sequences {139}, {445} (if loss), or {445,139} (if reordering), instead of the "exact" sequence {139,445}. To the best of our knowledge, no study of the impact of packet losses and/or reordering on attack forensics has been carried out so far. We have described the whole study in [193], and we provide in the following the main results.

In addition, we are collecting packets on precise and unique locations, the Honeypot sensors. The general problem of *vantage points* has been detailed in [177]: the location where packets capture is performed can significantly skew the interpretation of the capture, in quite non-apparent ways. Some vantage-point issues cannot be corrected without additional information, and this leads to a fundamental problem in network intrusion detection of adversaries being able to exploit vantage-point ambiguities to evade security monitoring [196, 102]. In our case, data collection is made at the receiver side. Only a small fraction of the traffic from the attacking Source is observed: the sole packets targeting the *Leurré.com* sensors. As a conclusion, existing solutions from the traffic analysis field are not directly applicable to our case [71, 120, 169, 241]. This led us to devise a new solution that we detail in the next section. This solution is based on a particular IP header field. An advantage of this method is that it relies on layer 3 information and is thus applicable to TCP, UDP and ICMP traffic altogether.

Reordering and Losses

We have focused on some particular network effects, namely the packet losses, retransmission, duplicates and forward reordering. A good definition of *forward reordering* can be found in [48] and is illustrated in Figure 4.2. It refers to packets sent by a source which are not received in the correct order at the receiver side.

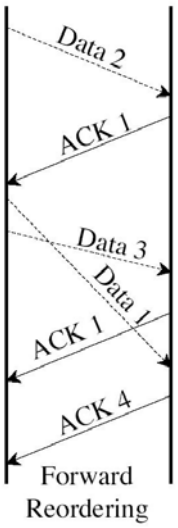


Figure 4.2: Forward Reordering from [48]

In order to detect *forward reordering*, we have developed a technique relying on a particular IP field called *the identification field, or IPID* that is normally used in fragment reassembly (see RFC 791 for more details [90]). As RFCs do not clearly specify it, this field is implemented in different ways, depending on OS flavors. Five different implementation scenarios have been observed so far ([30]):

- *Scenario 1*: IPID is a non-null (C) constant.
- *Scenario 2*: IPID is increased by a standard increment of one for each sent packet.
- *Scenario 3*: IPID is increased by an offset of 256 each time a packet is sent. This results from an unintentional error in Microsoft IP stack.
- *Scenario 4*: IPID is randomly chosen each time a packet is sent.
- *Scenario 5*: IPID is always a zero value.

The authors in [49, 62] report that the *id* field in the IP Header is generally implemented as a simple counter incremented by one each time an IP packet is sent. Bellovin also uses this particular property in [49] to detect NATs and to count the number of active

hosts behind them. We have demonstrated in [193] that most of the traffic we collect has the very same property. More precisely, around 75% of the sources having sent more than one packet share this property. It has also been demonstrated in [193] thatit is quite unlikely that ordered sequences of IPIDs could be due to random effects and, thus, that any disorder is most likely the representation of reordering effects.

We present in [193] a few algorithms to detect packet reordering which are applicable to the sources that use incremental IPIDs. In these cases, as presented in Figure 4.3, the receiver will only observe packets with IPID n and $n+2$ (resp. n and $n+512$), and not the one with $n+1$ (resp.$n+256$) in case of a packet loss and an IPID sequence $n, n+2, n+1$ (resp.$n, n+512, n+256$) in case of reordering.

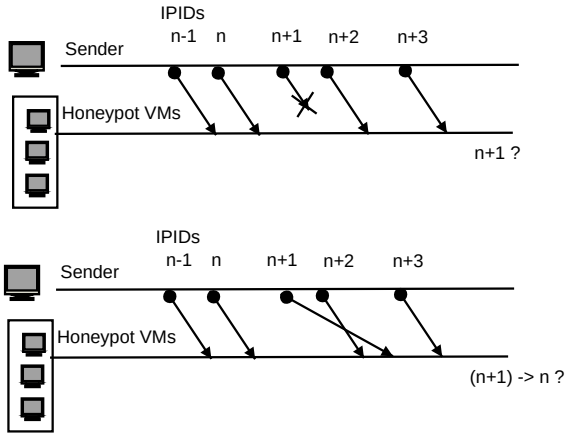


Figure 4.3: Honeypot-oriented Observations during Packet Losses or Reordering

For example, we identify reordering by looking at the sequences of received packets from each attacking source on a honeypot environment (Large_Sessions). Each mismatch between the sequence of IPIDs and the sequence of capture timestamps is labeled as a reordering. Algorithm 1 provides the pseudo-code to detect reordering for an attacking source sending packets $(Pkt_i)_{1<i<N}$ captured at time $(T_i)_{1<i<N}$ with IPIDs $(IPID_i)_{1<i<N}$. Such sessions are flagged with a *reordering flag*.

Algorithm 1 IPID Analysis: Reordering detection

```
for each sequence of packets and each attacking source
within the set of those identified as implementing scenarios 2 or 3 do
  if ∃ i ∈ [1..N − 1] verifying
  T_i < T_{i+1} AND IPID_i > IPID_{i+1} (mod 2^16) then
    detect_reordering = true
    break
  end if
end for
```

To avoid reordering effects, the easiest solution consists in ordering packets by their IPIDs when the source is labeled with a *reordering flag*.

A missing IPID can either be due to a loss or simply to packets sent by the Source to other destinations. At this stage, we can only make assumptions on the missing packets, and label the corresponding packets session with *loss labels*. The statistics method we have presented in [193] however helps providing a good confidence on the labeling. Detected losses will be taken into account in the following when comparing packet traces. There is no easy way however to interpolate the missing packets, except by statistically interpolating the traffic with the others on quite similar ports. We limit in the following the loss impacts by generalizing the parameters of the *activity fingerprint*.

Duplicates and Retransmission

It might happen that the network duplicates the original packet and generates at least two packets with the very same sequence number. The causes and impacts of these anomalies have been extensively studied in [119, 174, 155].

Brosh et al. have presented an interesting way to classify out-of-sequence packets, by comparing only two headers fields, namely the IPID and the TCP sequence numbers. Following Jaiswal et al. in [119], they have defined an out-of-sequence (OOS) packet to be a packet which TCP sequence number is smaller than previously observed sequence numbers (receiver time) in that connection. Duplicates, as well as reordering and retransmission, are classified as presented in Figure 4.4. Duplicates are easily identified as completely identical packets, including tcp sequence numbers and IPID. We remove them before any further analysis in our data. Reordering is detected and fixed as presented in the previous section.

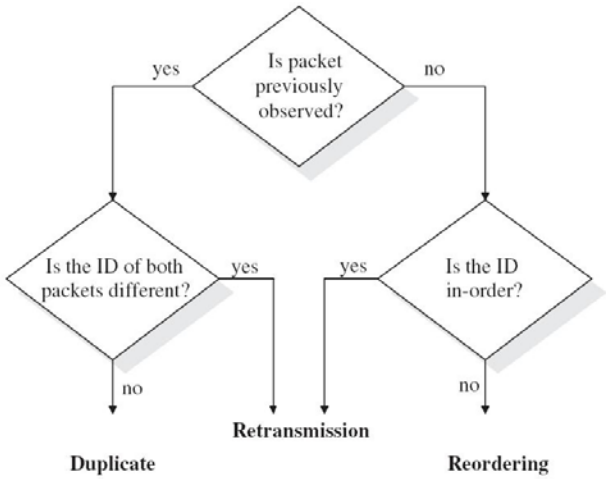
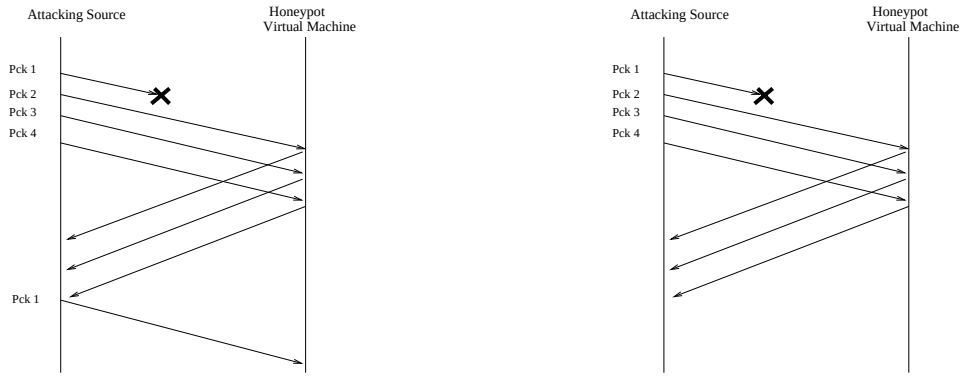


Figure 4.4: Classification Process of Out-of-Sequence Packets [57]

Retransmission corresponds to the two other outputs (see Figure 4.4). It is important to note that retransmission and loss are different, and might not be correlated. As some

attack tools might implement particular transport layer[3], a loss could not be detected and not imply a retransmission. This is illustrated in Figure 4.5 where the first figure presents a loss followed by a retransmission, while the second figure represents a loss without retransmission. Both scenarios are possible, but in the second case, we miss one packet. On the contrary, it is sufficient in the first case to reorder the out-of-sequence packet to get the initial sequence of packets.



(a) Loss with Retransmission

(b) Loss without Retransmission

Figure 4.5: Impact of Loss and Retransmission

Other biases

We have shown in the previous section how to limit the impact of packet retransmissions, duplicates and reordering. Other network disturbances exist, like the time it takes for a packet to travel from its source to its destination (delay, sometimes called latency and its variations aka jitter).

We explain in the following how to deal with the remaining effects, including:

- packet loss
- delay, jitter

Another network effect is the IP dynamic allocation. This issue has been avoided thanks to the definition of a *Source* (Section 3.5.2). We consider an IP as the same source as long as the observed IP activity does not contain more than a 25-hour period of inactivity. However, lots of other network problems might be considered. As an illustration, it has

[3] An interesting summary of the ambiguities in the true semantics of observed traffic has been presented by Paxson in a recent talk [175].

been shown in [203] that the simple task of identifying the end of a TCP connection is not easy, as many flavors are currently implemented in TCP stacks and despite the recommendations in RFCs [182, 38, 173]. It is also worth mentioning at this stage the study presented by Paxson et al. in [176], which aims at describing a few problems that might arise while conducting Internet Measurement. More precisely, the author focuses in [176] on some imperfect capture devices, which can exhibit limitations both intrinsic to their design and how we use them. This danger of *misconception* can lead to errors in equating what we are actually measuring with what we *wish* to measure. Paxson points out a few of these problems, including:

- Measuring TCP packet loss by counting retransmitted packets, leads to overlooking the problem of packets retransmitted unnecessarily, or of packets replicated by the network (see [119] for a study that explicitly acknowledges this difficulty, and [39] for a study demonstrating that differences in the two rates can be quite significant).
- *tcpdump* only produces an end-of-run summary of the total number of drops, so it is not possible to associate drops with the point in time at which they occurred.
- *tcpdump*, as a majority of tools, suffers from implementation flaws. As an illustration, major advisories appeared in 2004, as researchers found that tcpdump could crash or misbehave after parsing particular protocols like L2TP (port 1701), ISAKMP (port 500) or RADIUS (ports 1645,1646,1812,1813).

As a complementary illustration, it has been studied in a student project [65] some techniques to determine network anomalies due to attack crafted packets or to capture bugs and misbehaviors. The author focus on some TTL anomalies. This is another network influence that has not been considered at this time on the *classification* mechanism. We have, however, determined a bug in the capture of Snort, which modified particular fields (TTLs and IPIDs) due to a code error in the *TCP stream4 preprocessor*.

We can imagine to study all of the potential biases, and this must definitely be done. However, this is a huge amount of work, and not an easy task. We avoid the problem in the following by generalizing some attributes, like the number of packets, with regards to these potential network influences. This *generalization* approach remains realistic and feasible in the scope of our study.

Drawbacks

The techniques presented to address the impact of reordering, retransmission, duplicates modify the data and can be considered as an attempt to *normalize* the monitored traffic. It is worth pointing out here that such a *normalization* of the traffic might lead to some drawbacks. By withdrawing duplicates, we might miss particular types of attacks, e.g. the ones that sent very same crafted packets (all packets having the very same ip/tcp layer, in

terms of seq number, etc.). As a consequence, we limit the number of such data changes to one per Tiny_Session. This is justified as a large majority of connections concern a small number of sent packets. Thus, these network influences should remain limited. The most difficult point is to determine certain criteria, which help deciding if the observed phenomenon is only due to an artifact unrelated to the attack, or if the phenomenon itself is an additional feature of the attack. The framework must tolerate network anomalies, instead of withdrawing them all, in order not to bias attack analysis. This remark also justifies why we have not looked at other disturbances and why we leave them for future work.

4.3.3 Discrete Parameters

Different parameter categories

Influences due to the network have been accounted for in the previous section, and some of them have been cancelled. They are not related to the attack processes. The following step consists in classifying data. The fingerprint attributes have been briefly described in Section 4.2.2. Two packet traces (Large_Sessions) will be said similar if all of their fingerprint attributes (see Section 4.2.2) are similar. It implies that a similarity function must be defined for each attribute. These functions must at least consider the previously mentioned case of *losses*. We estimate that some attributes are less impacted by losses (or delays) than others. Those we believe cannot be impacted by losses are called *discrete values* hereafter and are described in the next section. Others highly fluctuate depending on losses. They are generalized in *Supervised Intervals* which are detailed hereafter.

Discrete Values

When applying machine learning in practical settings the first difficulty is raised by the attribute evaluation phase for the data at hand. The basic idea of attribute selection algorithms is searching through all possible combinations of attributes in the data to find which subset of attributes works the best for prediction. The selection is done by reducing the number of attributes of the attribute vectors, keeping the most meaningful attributes (which together convey sufficient information to make learning tractable), discriminating ones, and removing the irrelevant or redundant ones. In practice, the choice of a learning scheme (the next phase) is usually far less important than coming up with a suitable set of attributes.

We come out with three major discrete attributes out of the seven characterizing an *activity fingerprint* (see Section 4.2.2), that seem characteristic of different attack tool fingerprints and represent major semantics:

1. **Attribute A**: The *number of targeted machines*: An IP source can target either 1, 2 or 3 virtual machines in each *Leurré.com* environment.

2. **Attribute B**: The *ordering of the attack against virtual machines*. If the virtual machine has targeted several virtual machines, we give a boolean value 0 if the packets were sent in sequence and 1 otherwise. *In sequence* means that the Source sends all its packets to a virtual machine before targeting another one.

3. **Attribute C**: The *list of ports sequences* used against each virtual machine of an environment.

Figure 4.6 presents the cumulative distribution function (CDF) of the number of received packets per Virtual Machine, for each IP source (that is, each *Tiny_Session* in the *Leurré.com* terminology). 30% of the Tiny_Sessions contain at less 3 packets. This makes the number of machines quite stable against packet losses, and confirm our choice of Attribute A as a first clustering criterion. This property is also valid with ports, even if the average number of packets per port per Tiny_Sessions is smaller. Furthermore, due to the complexity of the interpolation processes, it is reasonable, in a first stage, to consider the sequences of ports as a discrete value. A refinement will be proposed in the next chapter. Following the very same idea, it would also have been possible to consider the sequences of targeted virtual machines instead of their number. The choice has been justified by the fact that whenever IP sources target several virtual machines (VMs), they follow the natural order of the VM IP addresses of a sensor in 99,7% of the cases.

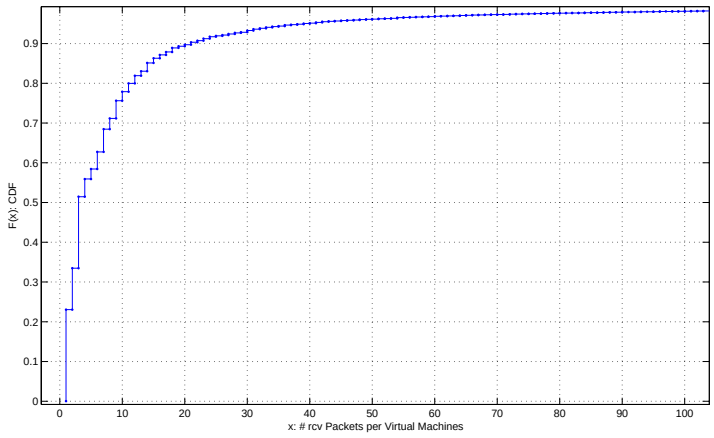


Figure 4.6: CDF: # Received Packets per Virtual Machine

The attribute evaluation can be done in different ways. They are listed hereafter, according to the *entropy* of classes. We compute the entropy H of each class probability distribution P as:

$$H(Class) = -\sum_{x \in Class} P(x).log(P(x)) \tag{4.1}$$

- *The Information Gain* evaluates the worth of an attribute by measuring the information gain with respect to the class.

$$InfoGain(Class, Attribute) = H(Class) - H(Class|Attribute) \tag{4.2}$$

- *The Information Gain Ratio* that evaluates the worth of an attribute by measuring the gain ratio with respect to the class.

$$GainR(Class, Attribute) = \frac{(H(Class) - H(Class|Attribute))}{H(Attribute)} \tag{4.3}$$

- *The Symmetrical Uncertainty* that evaluates the worth of an attribute by measuring the symmetrical uncertainty with respect to the class.

$$SymmU(Class, Attribute) = 2 * \frac{H(Class) - H(Class|Attribute)}{H(Class)} + H(Attribute) \tag{4.4}$$

- *The Chi-Squared Statistic* χ^2 (or *Pearson-Chi-squared statistic*) evaluates the worth of an attribute by computing the value of the chi-squared statistics with respect to the class. The computation is made between each pair of attributes in order to feed a *contingency table*.χ^2 is then quite easily derived from the table and express how related the two attributes are.

We intend in the following to evaluate the attributes thanks to the *Information Gain Ratio (IGR)*, also used in some decision tree algorithms like C4.5 ([198]), as it provides a fairer value than the *Information Gain* only. Indeed, this last notion tends to favor attributes that have many values. It is important to note here that we intend to keep all of the chosen parameters which depict an *activity fingerprint*. However, given a sample space of p dimensions, it is possible that some dimensions are less discriminatory than others. This measure intends to quantify this *difference*.

In Table 4.1, we give the number of clusters obtained by splitting the whole dataset (from February 1st 2003 to July 31st 2005) depending on the number of chosen attributes:

It can be observed from Table 4.1 that Attribute B is not really *discriminatory*. Its overall Information Gain Ratio remains very small. In other words, its contribution to the classification is not really significant. Actually, this is not surprising as it provides quite redundant information with Attribute A and as it discriminates the activities on several virtual machines only. This table provides an interesting estimate of the correlation among parameters, in the case this correlation is not straightforward.

Table 4.1: Classification in function of some discrete values

Splitting Attributes	Number of clusters	Info(F)
Attribute A	3	1.0940
Attribute B	2	0.0034
Attribute C	46446	4.269547
Attributes A and B	4	0.760566
Attributes A and C	46476	4.269695
Attributes B and C	46449	4.269569
Attributes A and B and C	46479	4.269717

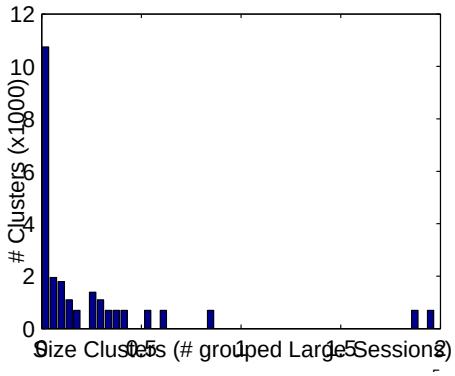


Figure 4.7: Distribution of Sizes among Clusters

Figure 4.7 gives the distribution of sizes among clusters (y-axis in thousands of clusters). There is a clear prevalence for small ones. In a more general manner, only 2702 clusters contain at least five Sources, 8759 at least 2 and 1699 at least 10. We remind here that the database contains 1431093 Large_Sessions, that is, there is a very small diversity of attacks with respect to Attributes A, B and C (in average, there is $\frac{1431093}{46479} \sim 31$ Large_Sessions per cluster). It is also important to note that more than 95% of large_Sessions are thus included within 2702 clusters. Without considering other attributes, this leads to the following corollary:

Corollary: The grouping of malicious activities by ports sequences and number of targeted virtual machines indicates that there does not exist a large variety of combinations in the wild.

4.3.4 Supervised Intervals

Attributes Description

We defined in the previous section some attributes with discrete values. Classification is simple in this case, as any combination of n values defines a new class. There are

other attributes, however, that characterize a fingerprint, but which cannot be considered as different for each different discrete values because the gain ratio would be very close to zero, as there would be too many generated classes. In other words, in such case, a very same fingerprint can have an interval of values for a given attribute. Among these parameters, we consider:

- **Attribute D** $\rightarrow$ *Duration:* The total duration during which an attacking source was observed on one honeypot environment. It is computed, for a given source, as the difference between the date the last packet has been received on the environment and the date the first packet has been received. This attribute aims at considering network delays as a simple artifact of the Internet and not as intrinsic features of the attack activities.

$$Duration = t(last\_received\_packet) - t(first\_received\_packet) \tag{4.5}$$

- **Attribute E** $\rightarrow$ *Number of packets sent to each virtual machine* by an attacking source. This attribute also varies because of packet losses. Variations might be simple artifact of the Internet and not intrinsic features of the attack activities.

- **Attribute F** $\rightarrow$ *Average inter-request time*: the average time interval between each packet received from the attacking source.

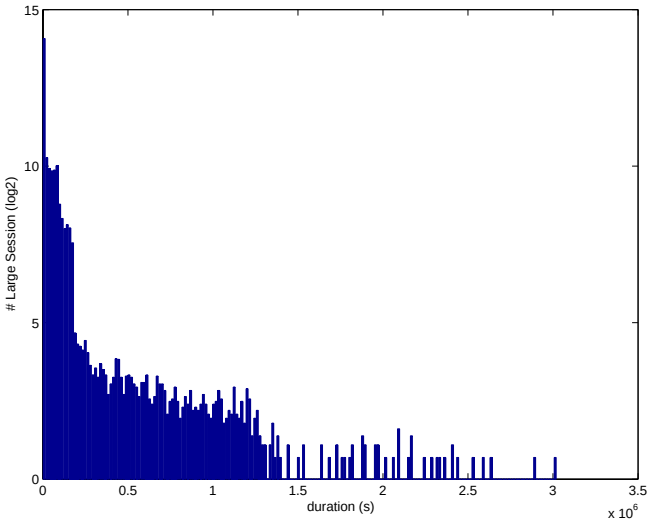
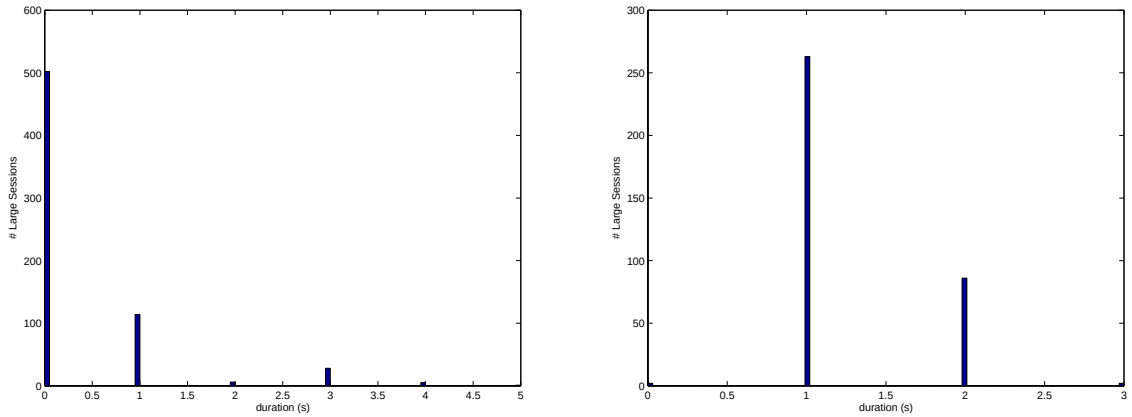


Figure 4.8: Distribution of the Duration Values over all Large_Sessions

It is not obvious to perform a good generalization with respect to different and unpredictable distributions. For example, Figure 4.8 describes the distribution of the duration values (Attribute D), in seconds, over all Large_Sessions in the database. A simple glance at the figure indicates the hard task of generalizing such an attribute. A contrario, Figure 4.9 presents the very same distribution, but limited to Large_Sessions within clusters 2404 and 1062 (obtained after having considered the discrete attributes in Section 4.3.3).

These two clusters are relatively larger than the average, and consist of a few dominant peaks. We consider for instance, that the two peaks of Cluster 2404 characterize two different fingerprints.



(a) Cluster 2404 (b) Cluster 1062

Figure 4.9: Examples of Duration Distribution among Two Clusters

Modal properties of these three parameters

As illustrated by the previous example, it seems important here to check that all clusters share the same property: there exist for Attributes D, E and F clear peaks that can be used to generalize their values. In order to prove this assumption, we make use of a so-called *peak picking* technique. Peaks of a distribution are often called *modes*. Many techniques which aim at extracting them automatically have been proposed [131, 134]. In the following, we intend to prove that there are a few of them in each cluster, and that their corresponding bins stem for the majority of their components. A bin is computed by a given baseline around the peak value. The baseline is determined according to the *tolerance* we give around the values (see Figure 4.10). The choice of a tolerance threshold is justified in the following, as it does not really impact on this demonstration.

The technique is detailed in Algorithm 2. We extract the weight of the 5 higher peaks of each cluster and we compute their global weight (relative to the cluster size, in %). These relative weights across clusters are then represented in Figure 4.11 for Attribute D[4]. The x-axis represents the indexes of the 1699 clusters gathering more than 10 large_Sessions. All values are higher than 85% (except for the 2 largest clusters, out of the 1699 considered clusters). In other words, the majority of the distribution of each attribute D, E or F can be expressed by no more than half a dozen of peaks. It seems thus relevant to apply this simple *peak picking* technique to all clusters in order to generalize the attribute values.

[4]The graph of Attributes E and F are very close to this one.

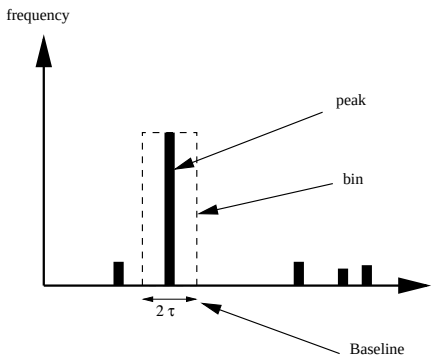


Figure 4.10: Peak Terminology of a Given Distribution

Algorithm 2 Distribution Properties of Attributes D,E and F

for all Attributes A_i with tolerance τ_i **do**
  for all Clusters C_j **do**
    $Weight(W_{ij}) = 0$
    Compute the distribution D_{ij}
    while There exists dominant peaks & $counter < 5$ **do**
      Extract Dominant Peak P_k
      Compute its baseline:
      Interval $[P_k \pm \tau_i]$
      This bin has a weight $w_{ij}(k)$
      $Weight(W_{ij}) = Weight(W_{ij}) + w_{ij}(k)$
      Remove the bin from Distribution D_{ij}
      Increment counter +1
    end while
  end for
  Plot distribution of Weight(W_{ij})
end for

Generalization Process

As another illustration, Figure 4.12 represents the distribution of attribute F, i.e. the average inter-arrival time, over all Large_Sessions. The x-axis describes the attribute values while the y-axis represents the frequency of the values in terms of Large_Sessions. Clusters also present interesting modal distributions, as it has been described in an analysis of Inter-Arrival Times (IATs) in [242] carried out in cooperation with Zimmermann et al. The analysis of some modal characteristics has led to interesting results, and the findings of two different activity anomalies:

- A strange IAT peak of value 28800s involving UDP port 38293: it turned out to be the misconfiguration of a Norton Antivirus automatic update server against a particular sensor. [242, 16].

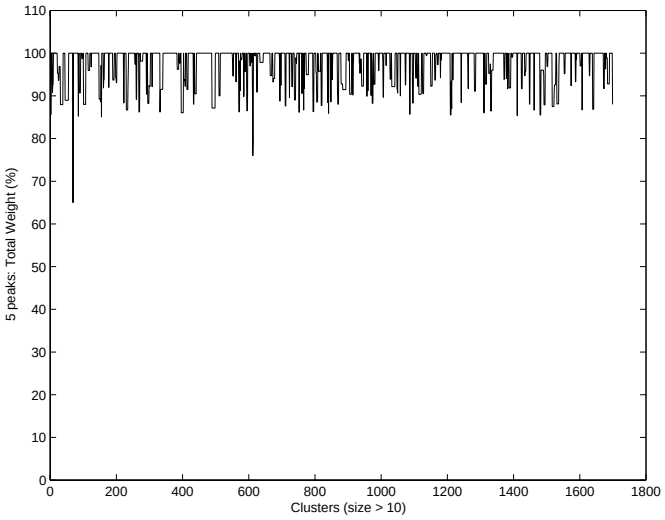


Figure 4.11: Modal Property of Attribute D: Weight of First 5 Peaks

- A strange IAT peak of value 9754s involving UDP port 1026: this port is used by Microsoft Windows operating systems for the Windows Messenger service, among other things. At least one worm is also known to propagate via a vulnerability using this service [242, 19]. This port is also known to be utilized for the distribution of spam over the Windows Messenger service [20]. The traffic has been periodically sent by two Chinese servers and monitored by several honeypot sensors.

As shown through the examples in [242], there are some clear and meaningful peaks. We use a dedicated algorithm to generalize these values according to some particular thresholds that are explained hereafter. In general, we have decided to keep on splitting the classes with certain tolerance indices (or bin baselines), where a tolerance index is defined for each attribute. The tolerance indexes are associated to probabilities of loss and delay in the network. The technique is summarized in Algorithm 3. It simply consists in considering peaks by decreasing frequency. For each peak, the baseline is computed according to the tolerance index. We also avoid the situation where there is a baseline overlap: in this case, the new baseline does not take values already included in the other baseline. An illustration of such a scenario is presented in Figure 4.13. Baseline of *Bin 2* is shorten as it partially covers baseline of *Bin 1*.

For each attribute presented above, we have used the tolerance thresholds presented in Table 4.2. Small variations of the different tolerance indexes does not make large classification changes. This is quite straightforward when looking at the two examples presented in Figure 4.9.

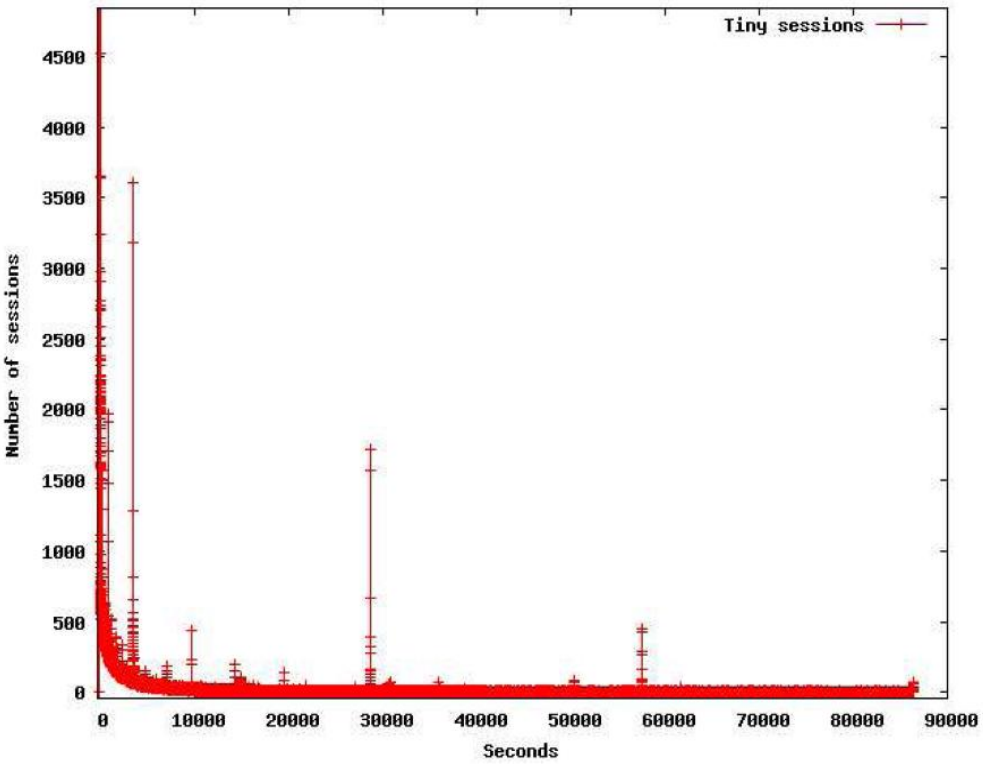


Figure 4.12: Distribution of *Average Inter-Request Time* Values

Table 4.2: Tolerance indexes τ_i

duration	2%
total number of received packets	8%
received packets per Virtual Machine	8%
average inter-packet time	2%

Information Gain and Remarks

The Information Gain summarizing this new clustering step is presented in Table 4.3. We only present the outcome of this method applied on the previous 2702 clusters, that is the ones grouping at least 5 sources (97,5% of all Large_Sessions). Globally speaking, all attributes are quite equally discriminatory. The splitting caused by Attribute D associated to F, i.e. the *observation duration* and the *average inter arrival time*) does not bring more additional information than the attributes considered alone. This is not totally surprising, as the attributes are not completely uncorrelated: the values of Attribute F are simply computed by dividing Attribute E by Attribute D for each Large_Session. This explains that the Information Gain Ratio is similar when considering Attributes (D and E), or (D, E and F).

Algorithm 3 Process of Generalization

```
for all Attributes A_i with tolerance τ_i do
  Compute the distribution D_i
  Order D_i in decreasing order
  for all Frequent Values D_i(j) do
    Take Interval with:
    α_min = min(D_i(j).(1 − τ_i), 0)
    α_max = D_i(j).(1 + τ_i)
    for all Other already-built intervals [a; b] do
      if a > α_min AND a ≤ α_max then
        α_max = a − 1
      end if
      if b ≥ α_min AND b < α_max then
        α_min = b + 1
      end if
    end for
    Add new interval [α_min; α_max] in the list
  end for
end for
```

Table 4.3: Classification with Supervised Intervals

Spliting Attributes F	Number of clusters	Info(F)
Attribute D	4109	1.0940
Attribute E	3100	0.0034
Attribute F	3085	4.2696
Attributes D and E	4813	0.7606
Attributes D and F	4703	1.0962
Attributes E and F	3112	4.3121
Attributes D and E and F	4815	4.3127

4.3.5 Validation: Unsupervised Classification

Introduction

At this stage, we have classified the data according to 6 major attributes, described in Sections 4.3.3 (discrete values) and 4.3.4 (supervised intervals). Another attribute has been mentioned but has not been considered so far: it is the payload of packets, which is also an interesting fingerprint attribute. It does not fit into the previous two categories. Taking the exact values is not relevant, as many fields in the protocol layers 5+ often include timestamps or other identifiers, which make the payload somehow unique. More generally, two major issues must be considered before comparing Large_Sessions and their associated data payloads:

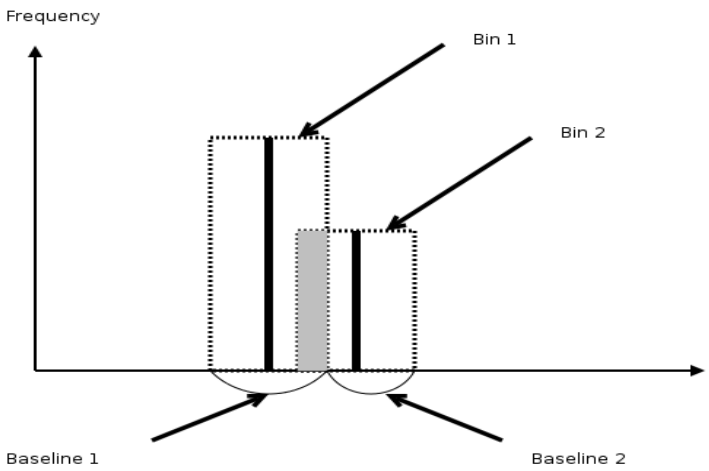


Figure 4.13: Peak Picking: Intersection btw Baselines

- First, when comparing two Large_Sessions in terms of all packets sent by each attacking source to an environment. Each Large_Session consists in several payloads to be compared with.
- Second, Payloads can include some random or changing fields that must not disturb the comparison.

A distance function is applied between payloads of each Large_Session from a given cluster. This function is called the *Levensthein distance* and is discussed in the next paragraph. The clustering top-down hierarchy is explained in the following paragraph, as well as the splitting criteria to get the new clusters.

Levenshtein-based Phrase Distance

In order to validate the consistency of clusters, we consider packet data contents. The payloads of all packets sent from the same source are concatenated to form a simple text phrase thanks to the tethereal utility [6]. Tethereal is the command line version of the popular network traffic analyzer tool ethereal. It allows examining data from a capture file and browsing detailed information for each packet in a text format. Thus, we consider each phrase as a concatenation of tethereal lines, with || separators. Figure 4.14 gives a short phrase of an ftp attack for illustration. Each cluster gathers all attack sources that are assumed to be due to a single root cause, i.e. to the use of the same attack tool against our honeypots. We define for each attack source its associated *attack phrase*. Then, we compare for each attack of one given cluster distances to all others phrases of the same cluster. This technique is based on the Levenshtein edit distance which is explained below.

The Levenshtein distance (LD) algorithm has been used in many domains, such as

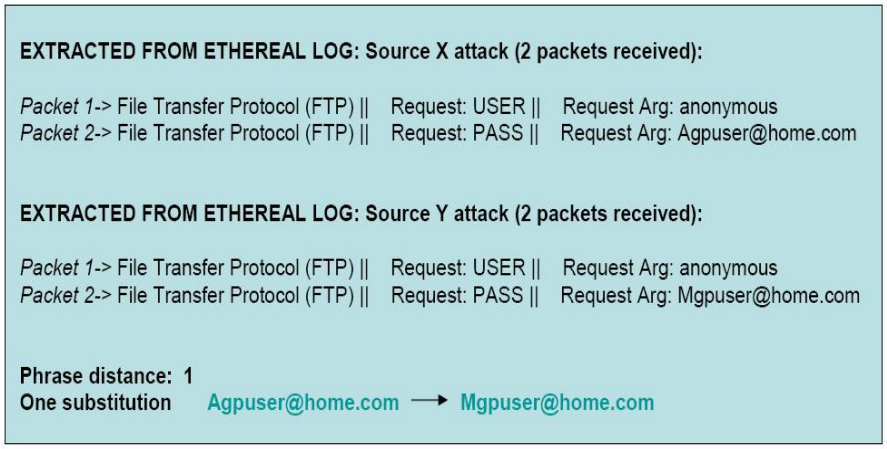


Figure 4.14: Simple Application of the Levenshtein Distance

spell checking, speech recognition, DNA analysis or plagiarism detection. It is a measure of the similarity between two strings, which we will refer to as the source string (s) and the target string (t) [10]. The distance (sometimes called edit distance) is the number of deletions, insertions, or substitutions required to transform s into t. For example,

- If s is "Agpuser@home.com" and t is "Agpuser@home.com", then $LD(s,t) = 0$, because no transformation is required to change s into t as they already are identicas are needed.

- If s is "Agpuser@home.com" and t is "Mgpuser@home.com", then $LD(s,t) = 1$, because one substitution (change "A" to "M") is sufficient to transform s into t.

In general the two components of the phrase distance (i.e. the string distance and the positional distance) can have a different cost from the default (that is 1 for both) to give another type of phrase distance. There is a third component: a cost which gives weights on the phrases that have less exact matches. It is described in details by Roger et al. in [10]. This third component is disabled by default (i.e. it has a 0 cost), but it can be enabled with custom cost. The method we apply sums the phrase distance from the words from the set (i.e. formed by the defined set of characters) and the phrase distance is calculated from the "words" belonging to the complementary set. Moreover, the algorithm used to find the distance is the "Stable marriage problem" one [114, 148]. This is a matching algorithm, used to harmonize the elements of two sets on the ground of the preference relationships. The positional distance only limits the impact of packet loss when comparing the *attack phrases* and computing their global distance. More complex methods could have also been considered, most of them being currently tested in bioinformatics to compare DNA sequences [58, 220, 229]. We leave this study for future work, as this simpler solution performs well with our dataset, as we will show later.

Hierarchy-based clustering

The hierarchy clustering technique which has been chosen to split within a same cluster all Large_Sessions that have similar *payload sentences* is the classical pyramidal clustering model built by an agglomerative bottom-up algorithm [94, 52, 53]. The goal is to obtain a hierarchical structure where each class of Large_Sessions is also partioned into sub-classes and so on, according to a given distance function between classes. The distance between two subclusters (or sub-classes) is the maximum of all pairwise distances between sentences contained in each subcluster. Applying the Levenshtein phrase distance combined with a hierarchy threshold τ_{Lev} (which is an upper bound of the maximum value of all pairwise distances) could generate new clusters from the original cluster, gathering Large_Sessions with similar payload contents (similarity given by small values of the phrase distances). As an illustration, we present in Figure 4.15 a pyramid built from the 8 payload sentences $(w_k, 0 < k < 9)$ associated to a given cluster, as well as the resulting four subclusters obtained by considering a given hierarchy threshold τ_{Lev} (the y-axis representing the considered inter-cluster distance). Let $\{C_{i,j} \mid j \in \mathbb{N}\}$ be the set of clusters obtained from the original C_i cluster (considering attributes: $A \dashrightarrow F$). We estimate the initial *cluster consistency* $\tilde{C}_i$ by computing the ratio of the largest size obtained among the new subclusters over the initial C_i cardinality:

$$\tilde{C}_i = \frac{max(card(C_{i,j}, \forall j))}{card(C_i)} \tag{4.6}$$

If the value is close to 1, it means that the cluster size has not significantly decreased during the splitting process. To illustrate this definition, it is worth noticing that the cluster consistency $\tilde{C}_i$ is equal in Figure 4.16 to $\frac{29}{39}$.

Another interesting value is the *Splitting Ratio SR*, which intuitively represents the number of obtained subclusters after applying the Levenshtein validation phase.

$$SR(C_i) = \frac{1}{card(jC_{i,j}exists) = \frac{1}{\#ObtainedSubclusters}} \tag{4.7}$$

We have computed the SR values obtained with several Levenshtein clustering thresholds $\tau_{Lev} \in [10..200]$. We first remark that the splitting does not change significantly when τ_{Lev} varies. We also note some cases where the splitting remains very low. These particular cases are discussed in the next sections, as well as the exact splitting criteria we have chosen, based on these experiments. Both indices *Splitting Ratio SR* and *cluster consistency* $\tilde{C}_i$ are employed to quantify and evaluate the impact of hierarchy-based splitting. They can also be used with other splitting methods as well.

Splitting Phase

It is important to note that some worms, also called *polymorphic worms*, can change their form of functionality as they propagate from machine to machine [130]. For instance,

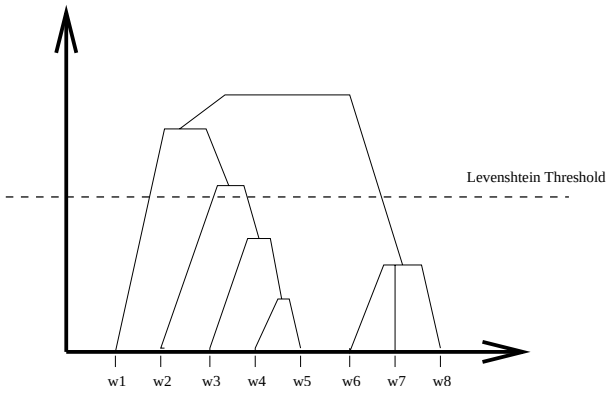


Figure 4.15: Pyramid: Levenshtein-Based Distance Splitting

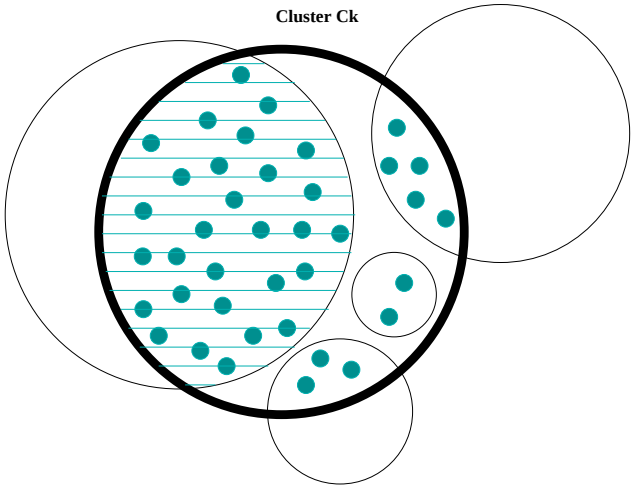


Figure 4.16: Splitting: Cluster Consistency

some families of worms contain an encryption engine, that could be very simple, e.g. just inserts *no-ops* (unnecessary system calls) into the worm code to evade signature-based detection engines, or could be as sophisticated as encrypting the entire worm using a random seed for every hop so as to evade detection during transit. Interesting studies have been presented by E. Filiol in [99, 100]. To date, only a few computer worms have used polymorphism successfully [223]. In a more general case, many packets can be encrypted or might contain random parts. Thus, the length and the content of the packets in a Session can be or not static, but are characteristic of such attacks. If such a polymorphic worm exists, it will generate a cluster, as previously described, but with a very low consistency value, in terms of the Levenshtein distance. All packets should be quite different in terms of *payload sentences*. The cluster should not be split in that case. For the same reasons, it might happen that a small field changes over the sessions of a very same attack. This can be a different identifier, or a timestamp that makes all *payload sentences* slightly different. In this situation, the cluster must also not be split.

We decide to split clusters only if cutting the hierarchy at threshold distance τ_d does

not lead an high value of the index γ_d, γ_d being computed as:

$$\gamma_d = \frac{\#\text{ Obtained Subclusters}}{\#\text{ Sources grouped in the initial Cluster}} \tag{4.8}$$

γ_d gives an indication of the number of created subclusters compared to the number of initial elements. We have presented in [242] the impact of splitting the existing clusters according to these two parameters. There is no real sense choosing high values for both τ_d and γ_d, according to the previous remarks. We however note that there is no real splitting impacts when choosing small τ_d or γ_d. The splitting phase is quite stable for small values. In the following, we consider the new clusters, obtained by choosing the following values:

- $\tau_d = 50$
- $\gamma_d = 0.2$

It is also important to keep in mind that the definition of γ_d does not reflect the size distribution of subclustering. As an illustration, consider a cluster C_k of cardinality $\bar{C}_k$. We guarantee with the previous method that the splitting phase cannot generate more than $\gamma_d * \bar{C}_k$ new clusters. However, both extreme scenarios are possible:

1. The method generates exactly $\gamma_d * \bar{C}_k$, each of size $\frac{1}{\gamma_d}$
2. The method generates two clusters, one being of size 1 and one being of size $\bar{C}_k - 1$

This does not present major drawbacks, except that we could consider in the second case the marginal cluster of cardinality 1 as a strong exception of the bigger one. We take the decision, however, to create two new subclusters, as we estimate that there might exist other reasons that could explain why this single source cannot be linked to the others.

It is important to note that each time this technique does not decide to split a cluster, according to the *Levenshtein Distance*, it estimates that there is a too large variety of different payloads within the cluster. This can be explained for two reasons: either the cluster gathers too many unique attacks, or the cluster is made of encrypted attacks. This case has been found for 5 clusters.

4.3.6 Global Consistency Index

New clusters can be obtained thanks to the Levenshtein phrase distance. We stop at this stage the discrimination phase (clustering), as we consider that all the criteria which have been determined as important for an attack tool fingerprint, have been taken into account in the clustering approach. However, it is still possible to imagine new criteria to check

the consistency of certain clusters. For instance, we can imagine home-made attacking tool which contains a bug, and which has the particularity of incrementing by 1 the TTL value of each packet it sends (modulo the maximum value 256). Such a tool exists, and has been reported in an analysis of the TTL field in [65]. This new attribute can then be considered as an additional fingerprint attribute for the cluster (or clusters) which is (are) associated to this tool (or configurations of that tool). On the other hand, this attribute might be a relevant attribute for that cluster only. We thus concentrate on the assessment of the consistency of the current clustering.

We define a ladder, the *Global Consistency Ladder*, which represents the *Global Consistency index* (GCI) of a cluster. Each time a precise attribute like the one previously mentioned (incremented TTL) is determined, GCI is incremented by 1 ($GCI + 1$) if the attribute matches the associated cluster, and decremented ($GCI - 1$) otherwise. The ladder is depicted in Figure 4.17. The matching is performed from the same equation presented by Equation 4.8 and a threshold of 90%. This means that if the attribute property covers more than 90% of the cluster, the cluster is said to be consistent with respect to the attribute ($GCI + 1$).

The GCI of each cluster might change. Some students at Eurecom have worked on particular tools that have made the associated clusters change their GCI value [65]. By default, the value is otherwise 0.

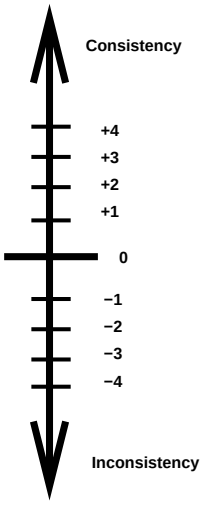


Figure 4.17: Global Consistency Ladder

4.3.7 Incremental Version of the Algorithm

Preliminaries

The classification described in the previous sections remains a method which is applied once, on collected data. Launching it on a simple computer can take several hours, and even days. This cannot work to carry out the analysis on arrivig data, to build early-warning systems. In other words, a newly observed Source should be classified according to the already existing classification. Thus, we propose, in the following, an adaptive technique which aims at providing an incremental complement of the algorithm presented in the previous section. The technique highly depends on the type of attributes, as the clustering method. Thus, three incremental steps are identified, for each attribute type.

This incremental step is important as *HoRaSis* can contribute to the building of an early-warning system. In this case, the incoming data should be analyzed and stored as fast as possible. This explains why we do not simply use fuzzy algorithms and have implemented more supervised ones. It is also worth noting that if the new monitored activity (Large_Session) cannot be related to already built clusters, then it means that the activity has not been observed so far. A report must be sent to the owner of the platform and a specific concern should be taken for this particular activity. These two features have been implemented in the *Leurré.com* project. First, anomalies are listed on the interface for all users. Second, reports are sent periodically to each partners, with different levels of details depending on the partner interests. An example of such a report can be found in Annexe E.

Discrete values

Taking the examples of Attributes A, B or C described in Section 4.3.3, it is easy to see that the incremental version will first offer to compare the triplets already observed. Otherwise, let Source S_i be the new Source and its associated traffic to a given honeypot environment: If the triplet A_i, B_i, C_i does not exist, a new cluster is created.

Modal properties

Attributes D, E and F are based on the modal properties of their distribution. We keep for each cluster an array of their values in a decreasing order according to their intensity. For each new incoming data from Source S_i, we update the array. The incoming Source increases by one the peak corresponding to the values of its Attributes D, E and F. Furthermore, each of these values can normally be attached to an existing interval. Otherwise, a new interval is created following initial algorithm presented in Section 4.3.4. The reason why peaks are monitored is justified by the fact that if a new peak modifies the

top 10 existing ones, it means that the initial frequency intervals are not valid anymore. In other words, the initial distribution has significantly changed. To date, this has not happened. In addition, it is important to note here that if there are too frequent updates, this will indicate that the distributions do not match the modal property anymore. Thus, it is something worth investigating. We have applied the algorithm presented in Section 4.3.4 during three periods of three months and for clusters defined in this period. We have then compared the obtained algorithms in terms of peaks and resulting intervals. The results are reported in Table 4.4. They clearly show the stability of each peak over several months.

Table 4.4: Classification with Supervised Intervals

	Oct-Dec 04	Jan-Mar 05	Apr-Jun 05
Similar top 5 peaks	94%	97%	98%
Avg Interval Overlap	3%	3%	5%
Avg Weight of the top 5 peaks	85%	91%	89%

Incremental Hierarchy-Based Partitioning

We saw in Section 4.3.5 how the clustering technique is refined by an hierarchy algorithm which splits the Large_Sessions sharing homogeneous payload sentences with the Levenshtein distance. This technique must also exhibit an incremental property in order to avoid the rebuilding of the hierarchy tree from scratch at each insertion. Sensitivity to input ordering is one of the major issues in incremental hierarchical clustering [101]. A basic method to update the tree would be to compare all the existing clusters with the new values. If the distance is higher than a fixed threshold, then the comparison with the cluster is considered unsuccessful. We however propose to make use of the particular structure of a hierarchy tree, as proposed in [231]. A cluster hierarchy is basically a tree structure with leaf nodes representing singleton clusters that cover single data points. Each node in the tree maintains three types of information: cluster center, cluster size and cluster density. The cluster density describes the spatial distribution of child nodes of a node. We define a clusters density as the maximum distance to the closest neighbor among the clusters members. Figure 4.18 represents the same pyramid as the one described in Figure 4.15. It includes, however, the nodes information previously listed.

Our approach for incorporating a new Large_Session payload sentence into the cluster hierarchy consists of two stages. During the first stage, the algorithm locates a node in the hierarchy that can host the new payload sentence. The second stage performs hierarchy restructuring, changing the density and size attributes of involved nodes. The approach is described in Algorithm 4:

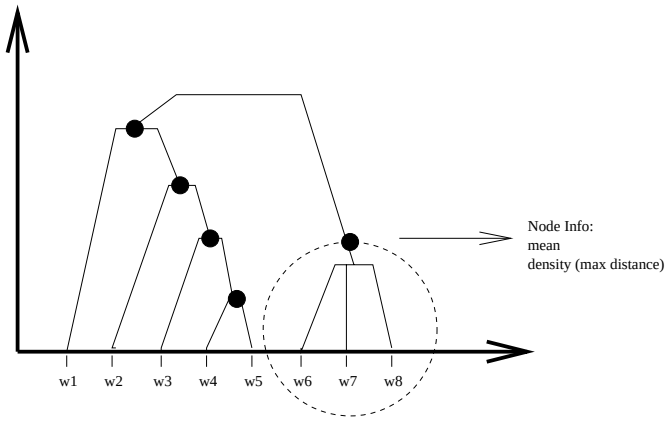


Figure 4.18: Pyramid: Incremental Hierarchy Approach

Other optimized versions might exist to parse the tree in more efficient way than this top-down approach. However, we are not in a situation where this optimization really matters. The incremental version is to date fast enough. This direction is also left for future work.

4.4 The Resulting Fingerprints

4.4.1 Global Statistics

The resulting number of clusters, including the ones which have not been split because of their heterogeneity with their content (see the Levenshtein distance splitting criterion in Section 4.3.5) is 8382. The average splitting ratio is thus $\frac{8382}{4815} = 1.74$. If we also consider the ones with less than 5 Sources which have not been considered in the second part of the algorithm, the technique has classified the whole dataset within 52159 distinct clusters. In other words, 52159 different *activity fingerprints* have been observed along the several months of data collection. This validation has an impact on the small number of clusters which are still quite large after having considered attributes $A \dashrightarrow F$. The limited splitting ratio is also due to the lack of sufficient payloads. More interaction from the honeypot sensors would improve the ratio and the global discrimination process.

This step gives the final clusters. These are the *attack fingerprints* we are looking for, as they gather all IPs sharing the same parameters. We detail and discuss the resulting clustering hereafter.

Algorithm 4 Incremental Algorithm for Hierarchy Clustering

$Res\_Clus \leftarrow \emptyset$
New attribute w_k to insert (payload sentence)
Let CH be the Clustering Hierarchy Tree for Attribute A
Parse the tree in a top-down manner
for all cluster N_i in CH with density τ_i **do**
  for all Values $w_i(j)$ **do**
    Compute $\hat{w}_i = MAX(d(w_i(j), w_k))$
    $d()$ being here the Levenshtein-based phrase distance
  end for
end for
Consider the node N_i with the largest depth that verifies:
$\hat{w}_i < \tau_i$
if Node N_i exists **then**
  return N_i
  link w_k to node N_i
  if Node N_i included in a cluster $\mathcal{C}r$ **then**
    $Res\_Clus = \mathcal{C}r$
  else
    $Res\_Clus = \{w_k\}$
    Send a notice indicating the creation of a new cluster
  end if
else
  Create new node with closest node and V_k as child nodes
  Send a notice indicating that the new entry does not match previous hierarchy
  Update tree path information if necessary for parents nodes
  $Res\_Clus = \{w_k\}$
end if

4.4.2 Attackers vs. Scanners

One of the attribute which is taken into account when building clusters is the number of targeted virtual machines on the Honeypot environment (Attribute A in the clustering process). Figure 4.19 represents two different evolutions over time. The first curve (-o-) represents all activities (Large_Sessions), which have targeted a single virtual machine on one honeypot environment, while the second curve (-x-) represents all activities which have targeted all virtual machines in an environment. Values are given in percentages, and we do not represent the activities against two virtual machines only, as they stem for less than 8% of the total number of activities and for each month.We also consider all honeypot environments. We let same analyses but performed on each environment for future work. It is interesting to observe that in the first 6 months of the experiment, the second category of activities was largely dominant, while in the last 6 months, it is the opposite situation. We note that at the beginning of 2004 (11th month), the trend abruptly changed. We have checked that it cannot be only explained by popular worms

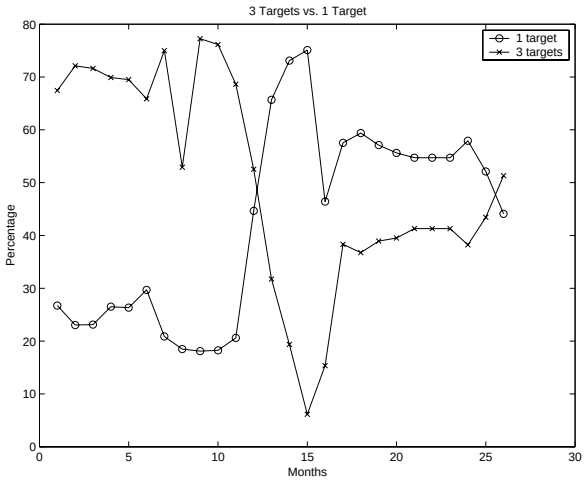


Figure 4.19: Larges_Sessions Targeting All vs. One Virtual Machines from Feb.2003

active at this date like Sasser[5].

This result also shows that scans on several consecutive IPs are still common. In addition, we have shown in [187] that almost all scans which target three consecutive IPs are programmed to hit them sequentially in increasing IP address order. In other words, it could be sufficient to define three unused IP addresses at the beginning of a network range, and *block* all external IPs that try to contact these three IPs in sequence. It would definitely block a large part of the so-called *background radiation* traffic.

Claim: This example shows the importance of monitoring malware activities over long periods of time. This allows determining quite easily the new trends and global changes of monitored activities. It also gives the opportunity to adapt security defenses to these trends.

4.4.3 Ports, Ports Sequences and Clusters

Ports Sequences and Clusters

Table 4.5 shows the differences between the notion of activity on a port, as reported in several web sites, and the concrete number of clusters associated to that port. The first column of Table 4.5 presents the top 10 ports given by the Internet Storm Center for the month of December 2005 [14]. The second column presents the number of distinct sequences of ports which have been observed including this port, and the third column represents the number of clusters[6], or *activity fingerprint* targeting at least that port on

[5]Sasser actually appeared three months after the abrupt decrease of the first curve, in the last days of April 2004 [26].

[6]All clusters, including the ones of size equal to 1.

a *leurré.com* sensor during the same month.

Table 4.5: Ports vs Clusters: different information levels

Ports	Sequences of Ports	Clusters
TCP 1026	221	446
TCP 6881	11	4
TCP 445	10447	16568
TCP 80	7504	2464
TCP 27015	2	1
TCP 135	7437	13122
TCP 40000	5	2
TCP 53	134	112
TCP 1025	9715	3413
TCP 65535	34	8

As a reminder, it is important to understand that there might exist more ports sequences than clusters. For instance, a given cluster can represent an activity which has different behaviors on each virtual machine, and can thus target different ports. We first observe from table 4.5 that there are a large number of sequences of ports. They are a first indication that many different attacks target a same port. The second column gives the number of distinct suspicious activities observed at least on that port. With no surprise, there is an important number of activities against the popular ports. It is worth investigating, at this stage, if some unexpected peaks on a port are due to a single activity or several ones. This step is made possible by the monitoring of *attack activities* instead of port statistics. This observation leads to the two following claims:

Claim: An analysis of malware activities cannot be limited only to statistics on a single port.

Claim: An analysis of malware activities cannot even be limited only to statistics on the ports sequences.

Clusters already give a better notion of *activities*, and are thus more meaningful for studying traffic on honeypots. We will show in the next chapter that activities might share common features that are also worth being investigated.

4.4.4 Interesting Activity Behaviors

One of our experiments related in [192] has led us to look at the clusters associated to the Deloder worm. The detection of the Deloder worm among the clusters is described

in more details in Annexe D. This worm, which spreads over Windows 2K/XP machines, attempts to copy and execute itself on remote systems, via accessible network shares. It tries to connect to the IPC$ share[7] and uses specific passwords. In Figure 4.20, we represent, per month and per country of origin, the amount of attack sources compromised by the Deloder worm that have tried to contact the honeypot sensors. More details of the Deloder identification are presented in Annexe D.

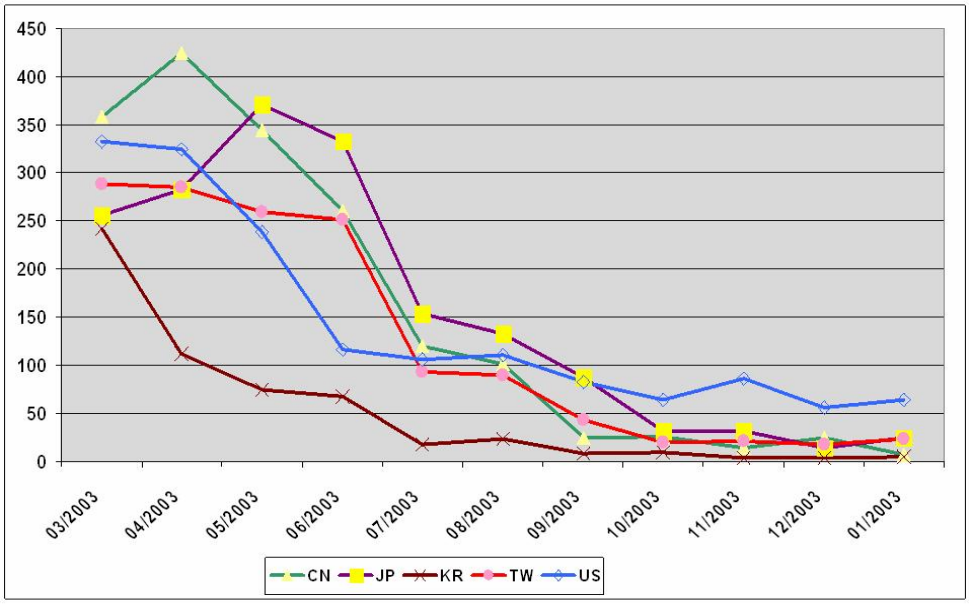


Figure 4.20: Deloder Activity (Nb associated attack sources)

A surprising observation from Figure 4.20 is the rapid decrease of its propagation around July 2003. [157] mentions that the shutdown of CodeRedII was preprogrammed for October 1, 2001. [222] mentions that Welchia worm self terminated on June 1st, 2004, or after having run 120 days. A similar mechanism could have been used for Deloder but, as far as we know, no one has ever made mention of it publicly. In the absence of such a mechanism, it is worth trying to imagine the reasons for such a sudden death. We have come with the following possible scenarios:

1. Deloder is still active but our virtual machines are not scanned anymore, for some unknown reasons. Statistically speaking, this seems unlikely and should be validated by means of other similar platforms.

2. All machines have been patched. Deloder has been eradicated. This is another unlikely scenario since Deloder has targeted a large number of platforms, many of them being personal computers which will probably never be patched. Newer successful worms targeting the same port (eg Sasser, Welchia, the Korgo family, etc.) tend to confirm this.

[7] or ADMIN$,C$,E$ shares depending on the Deloder variants.

3. Deloder bots are listening on IRC channels for commands to run attacks. One of these commands might have told them to stop the propagation process. In this case, the Deloder worm is not visible anymore but its botnet remains as dangerous as before.

At this point in time, unless a pre programmed shutdown is included in the Deloder worm code, we consider the third option as the most plausible one. A definitive answer to that question could be brought forward by someone who has access to the Deloder worm code, which we have not. If our assumption holds true, this would imply that worms writers have developed a new strategy. Instead of continuously trying to compromise more machines, they have decided to enter into a silent mode when the size of their botnets is sufficient [219]. By doing so, they dramatically reduce the likelihood of seeing an in-depth study of their worm being done as invisible worms are definitely less interesting to the security community than virulent ones. The bottom line of our findings is that such an in-depth analysis of that worm is probably worth being done if it has not been done yet. Sleeping worms might actually be more sophisticated and nefarious than active ones. We also deduce the following claim from this example:

Claim: Distributed sensor monitoring coupled with time analysis of *attack fingerprints* gives a good overview of the attack evolution over time. Fast increases or decreases of activities should be considered as abnormal behaviors, worth being investigated.

Other examples have been discussed in [192]. Due to space limitations, we report the interested reader to this work.

4.4.5 Attack Tool Identification

Cluster Signatures

The first step before doing any attack tool identification is to build a *Cluster Signature*, which represents the values of each attribute $A \dashrightarrow F$ used by the clustering technique. The discrete values are directly extracted for attributes $A \dashrightarrow C$, or the supervised intervals for attributes $D \dashrightarrow F$.

The main issue to determine a relevant *cluster signature* is the generalization of the *attack phrases*. The idea consists here in generating a regular expression from the different *attack phrases*, by taking the same approach as with the Levenshtein distance. Each detected deletion, insertion or substitution is replaced by a star $*$. This method has been carefully described in [165], and is illustrated by Figure 4.21.

More sophisticated techniques have been developed by research communities in bioinformatics (Pattern Discovery [178]) like the ELPH Gibbs sampler ([181]) and the teire-

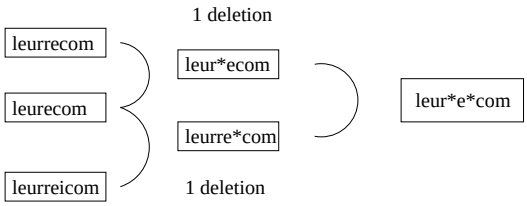


Figure 4.21: An Example of the *Attack Phrase* Generalization

sias algorithm ([200]). Future work will consist in testing and applying such techniques to replace the simple generalization mechanism currently implemented. However, the Levenshtein-based distance is currently sufficient for the clustering refinement.

As a conclusion, each cluster can be summarized by a signature. Figure 4.22 shows such an example. It symbolizes an activity observed on one single platform against the following TCP ports 80,135,139,445,1025,1433,2082,2745,3127, 5000 and 6129 for a couple of seconds ($8s \leq t \leq 10s$). The activity targets a unique virtual machine,

CLUSTER ID:	**IDENTIFICATION:**
2145	W32/Agobot−GM (sophos), also known as: Backdoor.Agobot.Id W32/Gaobot.worm.gen.k

FINGERPRINT:

* Number Targeted Virtual Machines: 1
* Ports Sequence: 2745,2082,135,1025,445,3127,6129,139,1433,5000,80
* Number Packets sent VM: 33
* Global Duration: 7s < t < 11s
* Avg Inter Arrival Time: < 1s
* Payloads: yes (DCOM, Netbios, WebDav)

Figure 4.22: Example of a Cluster Signature

Toward an automatic identification technique

The cluster presented in Figure 4.22 can be easily identified by googling as one out of the numerous Agobot variants [208]. Unfortunately, the general part which associates a name to each activity fingerprint (cluster), is currently missing in the presented framework. An on-going work intends to automatically link well-known exploit databases to attack fingerprints [226, 234].

The observed clusters can be well-known activities, or other activities hidden in the noise of bigger ones. These signatures have been applied to create new Snort alerts in the standard IDMEF format in [165]. The goal is here to add another information source to the current alert correlation engines, in order to refine their analysis. The presentation of this work is out of the scope of this document, even if it illustrates a concrete usage of the information provided by each cluster.

The method which we prone to identify tools is:

- First, look at potential tool candidates on Incident mailing lists.
- Second, run all of them in a secure environment against a honeypot sensor, to compare their traces with the *clusters signature*.
- Third, if a match is found, it indicates that such a fingerprint has already been observed. The cluster size can also present for a given period of time the frequency of such activity. On the other hand, no matching clearly indicates that the tool candidate, as such, has not been observed during the whole monitoring period.

To date, a few cases have been analyzed. Some tools have been determined by the previous method and have been detailed in [242]. They include:

- The RTSP scanner, exploiting a vulnerability on port 554 [112].
- The Grim's ping ftp scanner [2].
- The Roadkil's ftp probe [116].
- The SQLSnake worm (against port 1433) [8].
- The SFind.exe scanner (against port 1433 also) [103].
- etc.

This is however a fastidious task. One reason lies in the large amount of attack tools, or at least instances of attack tools that have been observed so far. The number of distinct clusters gives here a good hint of the value. Second, many tools are not available, or hard to find, as they are shared by a small community of users. These tools, on the other hand, would not be observed without the preliminary studies which we have made so far, and the particular environment given by the *Leurré.com* set up. It is thus quite unlikely that these tools are well-known in the security community.

4.5 Misclassified Traffic and Refinement

A Source has been defined as an IP address during a certain time window, depending on the inter-arrival time of the packets it sent. Packets sent within a 25-hour sliding window are attached to the same Source. Looking at the traffic which generated clusters of size 1 leads to the conclusion that the clustering method cannot be applied for particular scenarios, like the one illustrated in Figure 4.23. This diagram represents the arrival times

of packets from a given IP Address to the honeypot sensor. This IP address corresponds to a HPOpenview server (or more precisely a *HP Systems Insight Manager HPSIM*) that periodically scans machines in the network, using different layer 3 protocols and transport layer ports (UDP 161: SNMP, TCP 280: http-management, TCP 80: http) [29]. We have presented in Figure 4.23 its activity over a 7-day period, but it is important to note that it has been observed for five months.

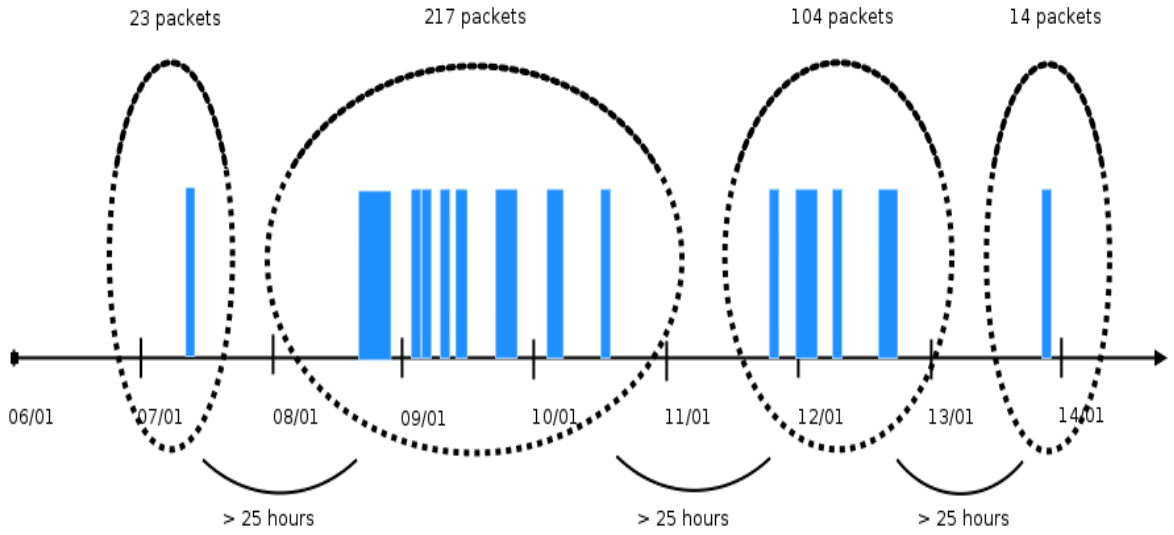


Figure 4.23: Observation of HPSIM Activities

According to our definitions, this traffic will be split into four distinct Sessions, and four different Sources. However, it is also clear at this point that the honeypot platform faces the very same *activity*. The fingerprint of this *activity* should group the four Sessions. Thus, it appears that the previous technique does not suit well for very long and recurrent yet not periodic processes like this one. Small clusters corresponding to such scenarios should be gathered into the same one. Thus, misclassified traffic might correspond to some particular misconfigured machines and/or network management activities. Instead of keeping them unclassified, it has been decided to group them by relaxing the clustering conditions. These particular misclassified activities are thus simply determined by very simple parameters. We decide to merge all very small clusters (less than 2 Sources) if and only if :

Parameter 1: They contain the very same IP addresses.

parameter 2: They are characterized by the same list of ports for each targeted virtual machine.

We thus relax the constraints of the clustering algorithm but we also add a third constraint to check that we deal with a scenario similar to the one presented in Figure 4.23 :

- The IP addresses in all clusters must be in the same subnet as the targeted virtual machines.

Such a property aims at identifying the recurrent activities from a given machine plugged in the same subnet than the honeypot sensor. As an illustration, the four clusters represented in Figure 4.23 are merged into a single cluster as they verify the three previous constraints. This technique allows to *merge* 64% of the Large_Sessions that were found in unique clusters (containing less than 2 Sources). These new clusters are interesting but remain anecdotal, as they are quite likely no malicious traffic. They are reported however to each network administrator in order for them to check the configuration of the corresponding machines and to stop if necessary these activities against the honeypot environments. Applying successively the clustering and this refinement algorithm, we find that 4% of the Large_Sessions remain within clusters of size 1. Some possible reasons are listed in the following sections.

4.6 Potential Evasions Mechanisms

4.6.1 Potential Scenarios

Our clustering algorithm has the advantage of grouping similar activities. It is however important to notice that the clustering method can be evaded. This could bias the clustering technique. This would also indicate that such a monitoring of activities in the wild disturb some communities at the origin of some traffic. Many evasion scenarii have however been considered at this time writing, and the clustering method remains efficient with most of them. A brief summary of the three most likely is given below:

1. *Scenario 1:* What if an activity had a random duration on the attacked platforms? First, TCP timeouts induce many constraints for this approach. Indeed, most TCP implementations utilize a *drop timer* which indicates the time period after which a connection not responding to keepalive probes may be considered as dead. Second, the splitting phase would generate an abnormally high splitting ratio. In other words, the randomness would make these attributes loose their modal properties. It would thus disturb the generalization process described in Section 4.3.4 by generating too many peaks. A splitting threshold limits this effect and allows detecting such a new trend. This scenario has not been observed so far. However, some tools already offer such a feature, like *advscan* which allows setting some variables such as the number of concurrent threads, the delay or the scanning duration [28].

2. *Scenario 2:* What if an activity would send several random packets in addition to the ones necessary for launching the attack? This would also disturb the splitting phase due to Attribute D. As with the previous case, a splitting threshold allows

detecting such a trend. However, this scenario, as the previous one, can be detected but is not rigorously addressed in the current version of the Algorithm. The splitting indicates that it has not been observed yet.

3. *Scenario 3:* What if the attack targets a random port before or after having targeted the one against which the exploit is launched? This scenario cannot be easily detected by the current clustering technique. It would however induce an important variation in the number of distinct ports sequences (Attribute A). Another detection mechanism would be here to build a graph with ports as nodes (vertices), and the number of ports sequences including the two ports as the edge weight between two nodes. Nodes with a high degree or a high variation in their degrees would detect such a scenario. This method is not implemented yet and is left for future work.

Our clustering technique works well, and has been proved efficient on the dataset collected for the last three years. However, attack techniques can change very fast, and the method must stay adapted to new changes. We have presented in this section possible attacks against our clustering algorithm. The first two are correctly considered at this time by the technique. To date, it is not the case of Scenario 3. However, new trends can be quite easily identified, which is by itself something important and which justifies the sensors deployment.

4.6.2 The Witty Worm Scenario

We have noted in previous Section 4.5 that 4% of the Large_Sessions are associated to clusters of Size 1. They might be due to losses that were not correctly considered during the generalization process. Another explanation could also be that we are monitoring particular activities like the Witty worm: This worm has been carefully described by the members of Caida in [22]. It is the first worm to target a particular set of security products – in this case Internet Security System's BlackICE and RealSecure. It infected and destroyed only computers that had particular versions of this software running. These tools contain a Protocol Analysis Module (PAM) to monitor application traffic. The PAM routine in version 3.6.16 of iss-pam1.dll that analyzes ICQ server traffic assumes that incoming packets on port 4000 are ICQv5 server responses but this code contain a series of buffer overflow vulnerabilities. To propagate, the worm thus generates packets with a random destination IP address, a random size between 796 and 1307 bytes, and a random destination port. The worm payload of 637 bytes is padded with data from system memory to fill this random size and a packet is sent out from source port 4000.

In this scenario, the activity fingerprinting as defined in our clustering will not work, as the worm does not target particular ports on the machine, but the firewall itself. Thus, each packet received on our sensor from Witty worm activities are likely to be found in clusters of size 1 (different destination ports, different payloads, etc). Looking to the

Source ports, it appears that only 2045 Large_Sessions share this property. It is normal as Witty was not a very active worm [22].

Thus the remaining clusters of size 1 are not artifact of this worm. However, this example shows an interesting evasion technique. The activities which intend to target tools capturing traffic[8] (like firewall, IDSs), instead of services listening on the machine ports, will not be correctly classified. It might also be the case for crafted packets which target network sniffers like ethereal or tcpdump. Such activities can however be monitored by analyzing several parameters (source ports for instance) for Large_Sessions associated to clusters of very small sizes.

4.7 Summary

This section has been rich in information. It seems important, at this stage, to summarize what has been shown so far. First, the traffic collected by the *Leurré.com* project has been gathered in a particular way: as each sensor of the project presents the very same configuration, we have grouped all Attack Sources sharing similar *activity fingerprints* on the sensors. The grouping has been made possible thanks to a particular clustering algorithm. Network effects are taken into account before and while grouping Sources. Then, a small number of parameters are considered as defining a fingerprint and help grouping all the Sources. The grouping has been also eased by the properties of some attributes:

- Number of targeted virtual machines
- Ordering of the attacks against the virtual machines
- Sequences of ports targeted against the virtual machines
- Duration of the observed activity
- Average time between sent packets
- Total number of packet sent by the source on the sensor
- Data payload sent by the source on the sensor

The clustering technique is by itself extensible and other techniques can be integrated instead of, or in addition to the ones presented and justified in this document. This direction will clearly be considered as the next steps of the project.

[8] It is not necessary for the capture to be in promiscuous mode.

The study of the *attack fingerprints* presents very valuable information. Some results have been described in the previous sections. Some other results can be found in related papers [183, 186, 192, 187, 242]. These clusters are the basis for forensics investigation. They provide a very interesting abstraction level to distinguish and compare activities between sensors and to distinguish the activities. Many studies can be performed at this data abstraction level, including:

- The temporal evolution of activities in a long term perspective;
- The determination of unique or global activities on the sensors;
- The statistical evaluation of the activities per platform;
- The early warning of newly observed activities;
- The correlation between monitored activities and alerts generated in the network hosting the honeypot sensor;
- etc.

The framework could thus be limited at this abstraction level. Indeed, all the bricks to build interesting studies now exist. We have performed a few of them, which have involved particular clusters (*activity fingerprints*). However, some important remarks emerge from these preliminary studies. When clusters are studied, there are some recurrent questions which arise, e.g. 'can the property observed on that cluster be generalized to other clusters?' 'Is this property somehow related to other properties?'

There might exist similarities between isolated activities (clusters), and the current technique does not manage to automatically extract all of them. We intend to move one step further, and propose in the next chapter a technique to identify these similarities in an automatic way, as defined by the initial requirements of the *HoRaSis* framework defined in the Introduction.

To conclude this chapter, it is important to repeat once more that all the analysis bricks have been created. They are the *attack activities* against the *Leurré.com* distributed network of sensors. The next two chapters will present an automated approach to extract all these activities that share strong similarities. They are illustrated all along by several experiences that have been carried out. This offers to the analyst an interesting framework to better understand and analyze the activities. This will complete the *HoRaSis* framework.

Chapter 5

Correlative Analysis

5.1 Preliminary Studies

5.1.1 Introduction

In Chapter 4, we have shown how to classify IP sources sharing similar *activity fingerprints* on our sensors and how to group them into so called *clusters*. Further to this classification mechanism, this section intends, by means of three short examples, to illustrate the possible relationships that might exist between some fingerprints. These relations have been found by digging into data, one thing leading to another. All of them have been reported in our previous publications ([67, 242, 186]), but are briefly summarized in the next three subsections. At this stage, it seems important to check in an automatic way if other similar relationships exist among the clusters, and determine the kind of information it can bring to the analyst.

5.1.2 Case Study 1: Country C Specialties

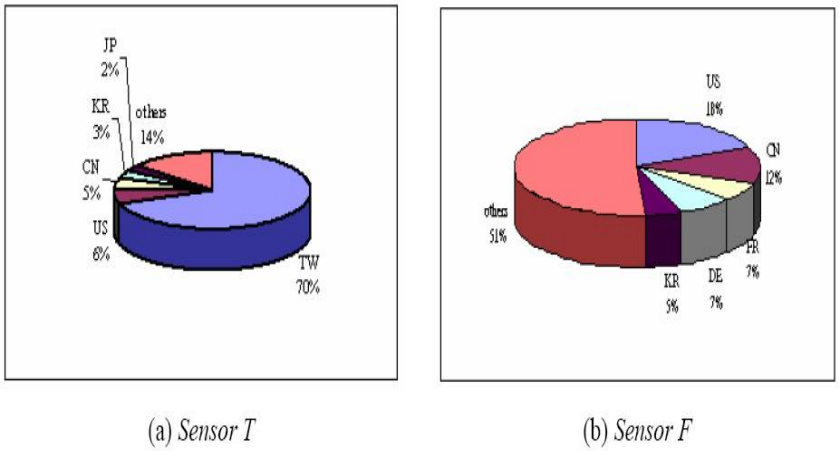
We have described several interesting results concerning Country C[1] in [67]. This honeypot environment presents some interesting features that have been detailed and related. Among them, we distinguish:

1. Very local attacks: attacks against Sensor C all originate from the same country than where C is located.

[1] The country is actually Taiwan. A complete analysis of this case study can be found in [67], as this last document has been written with the partner's consent.

2. Original attacks (attack fingerprints which have been observed on the sole Sensor C).

First, we compare the countries associated to the sources having targeted Sensor C with those having targeted a Sensor F (located in France). We observe in Figure 5.1 that the countries at the origin of the attacks against Sensor C and Sensor F are very different. The Figure provides the top five countries on each sensor, and all the other countries are grouped into the *others* category. We notice that 28003 distinct IP addresses (70% of the attacks) observed on Sensor C are coming from the very same country, Taiwan. This fact is in contrast with Sensor F where 53674 distinct IPs, that is 51% of all observed IPs, are found in the *others* category. Thus, there is no clear prevalence of attacking countries on Sensor F. Such a particularity is only encountered in the C Sensor. It has been confirmed by comparing with the other platforms as well.



(a) *Sensor T* (b) *Sensor F*

Figure 5.1: Attacking Countries Observed on Sensors C and F

It has also been shown in [67] that Sensor C has been targeted by very surprising attacks, which are unique to this sensor. They are most likely due to some specific malware scanning randomly the local network and its vicinity. Among them, we find frequent attacks targeting ports {8080,3128,1080,1813,80}. Such attacks have not been observed in any other honeypot sensor.

5.1.3 Case Study 2: Attacks From Serbia-Montenegro

Digging into the data, it has been discovered that YU (acronym of Serbia-Montenegro)[2] is quite an active country, as it belongs to the top fifteen most attacking countries over all the dataset. Strangely enough, all the attacks coming from *YU* have targeted a unique environment. We illustrate this in Figure 5.2, with a snapshot of the *Leurré.com* web

[2]As of 2006, YU is deprecated in favor of CS,as specified in the standard ISO 3166-1

interface. The Figure represents the distribution of attacks coming from YU over the honeypot sensors and per month. Sensor 6 (Env_6 in the Figure) is the only one that has been periodically targeted. From our preliminary studies, we have also found that the attack tools, or fingerprints on this Sensor were not associated to YU only: in other words, several activities observed on the considered Sensor 6 have been monitored on other platforms, but they have been all identified as coming from YU on that sole Sensor.

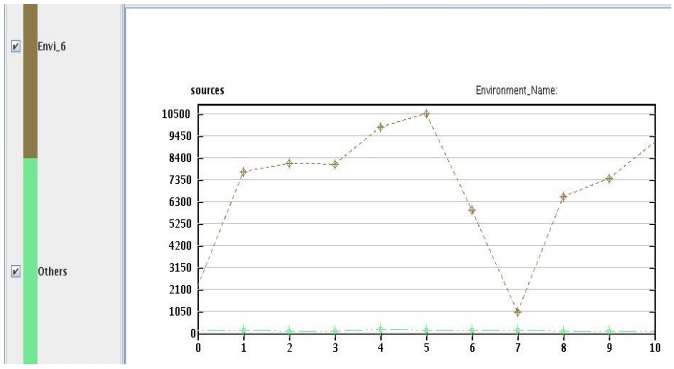
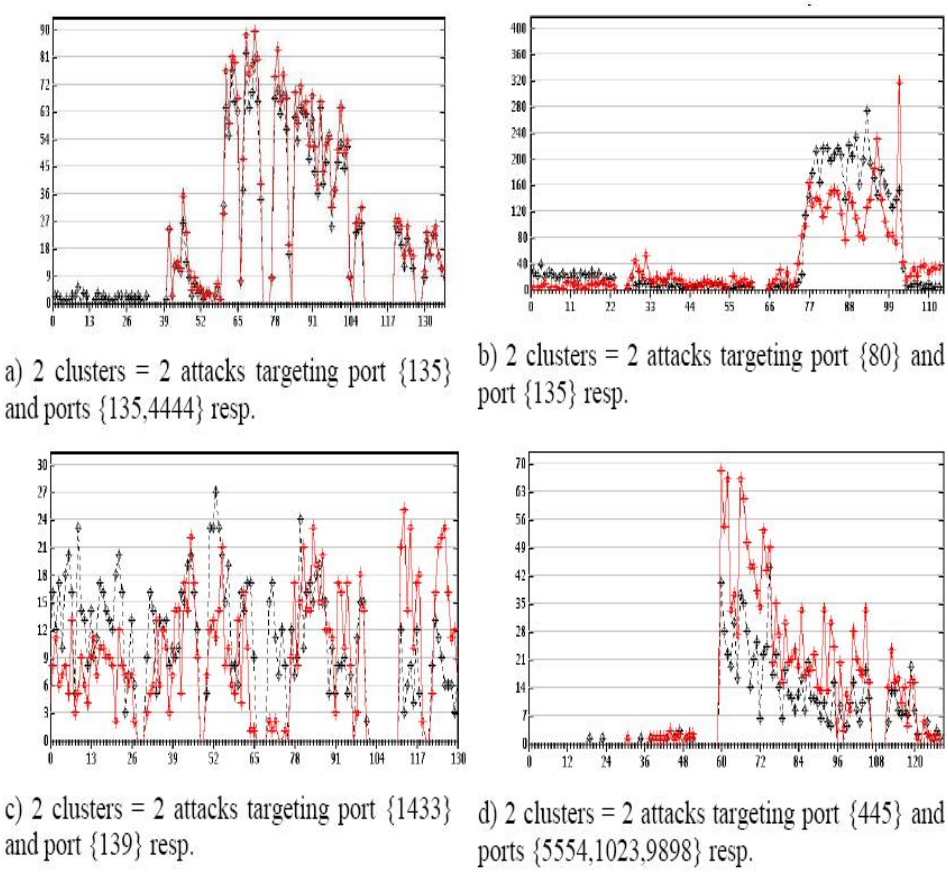


Figure 5.2: Attacks from YU Observed on Each Honeypot Sensor per Month

5.1.4 Case Study 3: Apparent Temporal Relations

We have shown in the previous chapter and publications ([242, 186, 192]) that the clusters obtained by the clustering algorithm are coherent in terms of their contents and may also reveal worth-investigating attack features (like the geographical locations of the attacks, the attack ordering, the raw profile of attacking machines, etc). We have been able to name a few of those clusters by comparing the fingerprints of some known tools on our honeypot, obtained in a controlled environment, to the fingerprints obtained in the wild. However, this task is tedious, and only a few dozens of tools have been clearly identified so far. To further complement our clustering method, we looked at the time behavior of the clusters. Indeed, as illustrated by Figures 5.3, where the y-axis represents the number of IP addresses associated to each cluster, as a function of time (with a granularity of 3 days on the x-axis), some clusters clearly exhibit a similar time evolution. It is however striking that those similar (w.r.t. time) clusters correspond to very different attack fingerprints. What is more, Figures 5.4 further highlight that the global activities against some of those ports (by summing the activities of all the clusters targeting those ports) are completely uncorrelated. Without going into the details, intervals between brackets show periods where no evident time correlation is noticeable. We report the reader to [194] for more in-depth treatment of this phenomena. An important conclusion from those examples is that some temporal correlations exist between attack fingerprints that seem otherwise

unrelated. This result clearly deserves further investigation, and it will be done in Section 5.4.7.



a) 2 clusters = 2 attacks targeting port {135} and ports {135,4444} resp.

b) 2 clusters = 2 attacks targeting port {80} and port {135} resp.

c) 2 clusters = 2 attacks targeting port {1433} and port {139} resp.

d) 2 clusters = 2 attacks targeting port {445} and ports {5554,1023,9898} resp.

Figure 5.3: Examples of Time Correlation between Clusters

5.1.5 Interesting Analyses

The previously described examples highlight the fact that some questions can frequently appear when looking into the data. Do we observe more numerous attacks in average coming from a very specific country against some sensors? Do the other sensors also monitor attacks coming from their local hosting network? In a more general way, would it be possible to find other clusters sharing particular distributions in terms of the origins of the sources? And in terms of the targeted sensors? Of course, the geographical location of the sources is definitely not the only question that might arise. For instance, which other clusters share temporal relationships? What are all the temporal relationships that can be deduced from the whole *activity fingerprint* classification? If so, do the group of time-correlated clusters also present some similarities in terms of the origin (country, domain) of the attack, or at least on the Operating Systems launching such activities? These simple but recurrent questions have led us to define a new method that would automatically deal with these recurrent questions by considering the various similarities

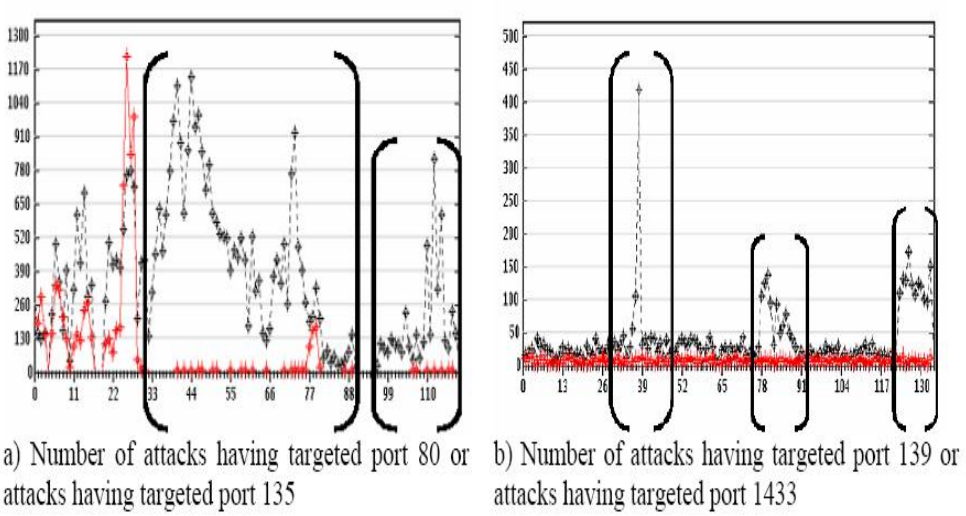


Figure 5.4: Observed Activities on some Targeted Ports

that might exist among clusters. They will enable us to determine in an automatic way all the cases which have been observed coincidentally so far.

5.2 The Theory

5.2.1 Underlying Motivations

Let try first to summarize what has been presented so far. We have gathered the traffic collected from a distributed honeypot environment, according to the fingerprint of the activities. Each cluster gathers all IP sources sharing the same fingerprint, or *attack activity*, on a honeypot platform. The honeypot data is now turned into clusters, and the rest of this thesis intends to prove the interest of working at this abstraction level. First, it is important to note that many analyses are applicable on the clusters. We distinguish two distinct analysis classes:

- *Intra-Cluster Analysis:* Within a cluster, the analysis aims at extracting features that are more specific to this cluster than to others, in order to enrich the knowledge and understanding of the phenomenon which has created those traces (*root cause* of the *activity fingerprint*).

- *Extra-Cluster Analysis:* The analysis aims at finding relationships between clusters, and potentially to group a few of them sharing common characteristics.

This chapter offers to address both analysis classes. The first type of analysis aims at finding specific features of some attacks. When they are clearly identified, they can be used to improve and check the consistency of the cluster and to improve the matching of new incoming Sessions candidates. The second type of analysis aims at checking if the previous features are shared as well as other properties by several clusters. The technique to extract such information between clusters must satisfy at least three major constraints:

1. **Modularity**: Any kind of analysis should be easily applicable and compared with other results. Attack mechanisms are more and more complex, or at least, imaginative. It is sensible to think that an analysis might not be relevant for long periods of time, and can be evaded by new attack characteristics.

2. **Cross-Analysis**: The combinations of existing analyses must be cross-correlated to find, if they exist, emerging properties. The *a priori* knowledge might help choosing the analysis methods, but new clusters correlation might emerge *a posteriori* by crossing analyses.

3. **Validation**: It is quite frequent that the analysis relies on several parameters (indexes, thresholds, etc). Several values should be tested and compared on the data set to estimate their impact and the stability of the analysis w.r.t. such parameters.

Obviously enough, there are other constraints, that have been respected from the beginning of this research. The solution must remain intuitive and meaningful, and it must scale to the dataset, that is it should deal with at least thousands of clusters and to a few millions of IP sources. With respect to all these criteria, we have considered applying a method based on the graph theory. Before explaining what has led to this decision, let first summarize the global clustering overview: we have N clusters, N being in the order of a few thousands. In the worst case, an *extra-cluster analysis* can show up relations among all the clusters, that is $\frac{N(N-1)}{2}$. From another perspective, if $N = 1000$, we get 499500 relations, and in a more normal case, if $N = 50000$, we get $1.25.10^9$ relationships. This relationship among clusters cannot be interpreted easily. The computation remains quite important. We want to extract the useful, most important sets of clusters that present strong relationships. This kind of problem might be addressed by several techniques issued from the Discovery Knowledge domain. However, we can also add some other constraints. First, the solution should work with multiple similarity functions, that is, for any analysis we will potentially perform. Second, the solution should be able to take into account many combinations of analyses, that is, we focus on a large discovery method. The method should follow simple and non counter-intuitive steps, and the outcome should remain readable and exploitable by the analyst.

Among solutions to determine relationships between large datasets, one simple but widespread solution consists in applying association rule (AR) mining, which have a wide range of applications in many areas of business practice and research - from the analysis of consumer preferences or human resource management, to the history of language [37, 242]. For instance, these techniques enable analysts and researchers to uncover hidden patterns

in large data sets for so-called market basket analysis, which aims at finding regularities in the shopping behavior of customers of supermarkets. With the induction of association rules one tries to find sets of products that are frequently bought together, so that from the presence of certain products in a shopping cart one can infer (with a high probability) that certain other products are present. AR algorithms induce rules (A and B imply C), while we want to find sets of clusters sharing similarities. The notion of rule deduction among clusters can be interesting but does not correspond exactly to the previously described criteria.

Another area that deals with information extraction from data sets is graph and matrix theories. Graphs, or matrices, are widely used with large data. Matrices can be one representation of satellite images, or graphs can be the representation of large class A networks. Many techniques and methods have been designed in this direction to deal with big datasets. Graph-Theory is a mature and important research domain, which is very interesting in this situation. Graphs can be easily visualized by end-analysts, and many algorithms have already been developed, in a large panel of areas, like telecommunications, image processing, video monitoring, etc. Many mathematical problems have found their solutions in a simple graph abstraction.

The solution we propose in the following answers the previous requirements. It is based on a particular set of graphs called *cliques* ([56]), and algorithms which aim at extracting dominant sets (maximal weighted cliques) out of each analysis graph.

5.2.2 Building Similarity Matrices

The technique we present can be applied on matrices expressing similarities. Thus, as a preliminary step, it is important to explain the different logic stages which lead an analysis on the clusters to be turned into a matrix composed of integers or real values.

The starting point is the characteristic which is under scrutiny. This characteristic is associated to either an *extra* or *intra* cluster analysis. Then, several steps are required to express the characteristic in terms of a similarity matrix among clusters. They are listed below:

1. Definition of a characteristic
2. Representation of this characteristic (as a vector, for instance)
3. Quantification of this representation (values to be included in the vector for instance)
4. Definition of a distance to measure how far away two clusters are w.r.t. this characteristic
5. To insert the values in the similarity matrix associated to the analysis

It is important to note that each of these steps can be implemented in diffrent ways. We present the ones we have applied in the next Section 5.3 by means of several examples. The different steps lead to the creation of a matrix $\mathcal{M}$, with the similarity value between Cluster i and Cluster j being reported in $\mathcal{M}(i,j)$ and $\mathcal{M}(j,i)$ (symmetrical matrix). The diagonal only contains null values. Let now assume we have these similarity matrices. Next section intends to show how they can be used to extract information out of them.

5.2.3 The Theory

A clique C_l in a simple undirected graph G is a set of nodes such that there is in G an edge between every pair of nodes in the set C_l. A clique of k nodes is called a k-clique. The size of the largest clique in the graph is called the clique number of that graph. As a simple example, every complete (fully connected) graph K_n is a clique consisting of all n nodes as illustrated in Figure 5.5 with 3-clique and 4-clique. Let G be an undirectional edge-weighted graph with no self-loops $G = (V, E, w)$ where $V = (C_1, ..., C_N)$ is the vertex set, $E \subseteq$ V x V its edge set and $w()$ be the weighted function associated to each edge :

Definition 5.1. Maximal Cliques*: A **clique** is a subset $C_l \subseteq V$ such that $(i,j) \in E$ for all $i, j \in C_l$. A clique is **maximal** if it is not contained within any other clique.*

We formalize the problem of discovering fingerprint relationships as the problem of searching for edge-weighted maximal cliques in the graph of N nodes (or clusters). Indeed, in the past, some authors have argued that maximal clique is the strictest definition of a cluster [41]. The process is the following: we find a maximal clique in the graph and remove the edges of that clique from the graph; we repeat the process sequentially with the remaining set of nodes and edges, until there remain non-trivial[3] maximal cliques in the graph. The leftover nodes after the removal of maximal cliques are dissimilar from most of the nodes. All these steps are detailed in the next Section.

5.2.4 Relation Discovery: Maximal Cliques using Dominant Sets

The Concept

Many graph-theoretic clustering algorithms consist in searching for certain combinatorial structures in the similarity graph, such as a minimum spanning tree, or a minimum cut and, among these methods, a classical approach reduces to a search for a complete subgraph, namely a *clique*. Unfortunately, while the minimum spanning tree and the

[3]The notion of *non-trivial cliques* will be discussed in the next Sections.

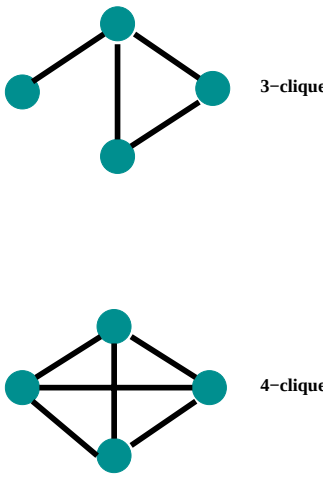


Figure 5.5: Simple Examples of Cliques

minimum cut (and variations thereof) are notions that are explicitly defined on edge-weighted graphs, the concept of a maximal tree is defined on unweighted graphs, but it has also been generalized to the edge-weighted case [56].

Finding maximal cliques in an edge-weighted undirected graph is a classical problem in graph theory. Since combinatorially searching for maximal cliques is computationally hard, numerous approximations to the solution of this problem have been proposed [199]. For our purposes, we adopt the approximate approach of iteratively finding *dominant sets* of maximally similar nodes in a graph [172]. Beside providing an efficient approximation to finding maximal cliques, the framework of dominant sets naturally provides a principled measure of the cohesiveness of a class as well as a measure of node participation in its membership class. This measure of class participation may be used for an instance based representation of a clique [128].

We have introduced in Section 5.2.2 the notion of similarity matrices which can be used to express certain similarities among clusters. Such matrices can also be seen as graph representations. We represent the similarity matrix as an undirectional edge-weighted graph with no self-loops $G = (V, E, w)$ where $V = (C_1, ..., C_N)$ is the vertex set (the list of clusters), $E \subseteq$ V x V is the edge set, and w: $E \rightarrow \Re+$ is the positive weight function (the similarity values inserted in the matrix). In summary, we represent the graph G with the corresponding weighted adjacency (or similarity) matrix, which is the n x n symmetric matrix A(i,j) defined as:

$$a_{i,j} = \begin{cases} w(i,j) & \forall \text{ (i,j)} \in \text{E} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

This matrix is computed using the notion of similarity described in Section 5.2.2. To quantize the cohesiveness of a node in a cluster, let us define the *average weighted degree* of a node, as described in [172]. Let $S \subseteq V$ be a non-empty subset of vertices and $k \in S$,

such that:

$$awdeg_S(k) = \frac{1}{|S|} \sum_{j \in S} a_{kj} \tag{5.2}$$

Observe that $awdeg_{\{k\}}(k) = 0$ for any $k \in V$. In addition, for $j \notin S$, we define:

$$\Phi_S(k, j) = a_{kj} - awdeg_S(k) \tag{5.3}$$

$\Phi_S(k, j)$ measures intuitively the similarity between nodes k and j with respect to the average similarity between node k and its neighbors in S. Note that $\Phi_S(k, j)$ can either be positive or negative, and that $\Phi_{\{k\}}(k, j) = a_{kj}$, for all $k, j \in V$ with $k \neq j$.

We are now in a position to formalize the notion of *induction* of node-weights, which is captured by the following recursive definition. Let $S \subseteq V$ be a non-empty subset of vertices and $k \in S$. The weight of k w.r.t. S is:

$$w_S(k) = \begin{cases} 1 & \text{if } |S| = 1 \\ \sum_{j \in S \setminus \{k\}} \Phi_{S \setminus \{k\}}(j, k) w_{S \setminus \{k\}}(j) & \text{otherwise} \end{cases} \tag{5.4}$$

Moreover, the total weight of S is defined to be:

$$W(S) = \sum_{k \in S} w_S(k) \tag{5.5}$$

Note that $w_{\{k,j\}}(k) = w_{\{k,j\}}(j) = a_{kj}$, for all $k, j \in V$ $(k \neq j)$. Also observe that $w_S(k)$ is calculated simply as a function of the weights on the edges of the subgraph induced by S. Intuitively, $w_S(k)$ gives us a measure of the overall similarity between fingerprint k and the vertices of $S \setminus \{k\}$ with respect to the overall similarity among the vertices in $S \setminus \{k\}$.

We are now in a position to define *dominant sets*. A non-empty subset of vertices $S \subseteq V$ such that $W(T) > 0$ for any non empty $T \subseteq S$, is said to be *dominant* if:

1. $w_S(k) > 0$, $\forall k \in S$, i.e. internal homogeneity
2. $w_{S \cup \{k\}}(k) < 0$, $\forall k \notin S$, i.e. external homogeneity

Algorithm 5 Generating Dominant Sets within weighted graphs

```
for all weighted graph G = (V, E, w) with N nodes do
  P = ∅, be the set of dominant sets
  while stopping_criterion(G) do
    S ← dominant_set(G)
    P ← P ∪ {S}
    E ← E \ E_S
  end while
end for
```

Considering this definition introduced in [172], the algorithm we have designed basically consists of iteratively finding a dominant set in the graph and then removing involved edges from the graph, until all vertices have been clustered. The algorithm is explained in pseudo-code in Algorithm 5.

In this pseudo code, the procedure *dominant_set(G)* finds a dominant set in the current graph G. The procedure is based on a technique proposed by Pavan et al. in [172], and discussed in more details in the next paragraphs.

The function *stopping_criterion(G)* simply checks whether the current graph is valid according to a few constraints we add (for instance if it contains at least two vertices or not). This is also detailed in the following paragraphs. The assignment of node weights naturally provides us with a measure of the overall similarity of a dominant set. Given a dominant set $S \subseteq V$, we measure its overall *cohesiveness* with:

$$cohesiveness(S) = \frac{1}{2}\frac{\sum_{k \in S} awdeg_S(k) w_s(k)}{W(S)} \tag{5.6}$$

As a remark, it would have been possible, in Algorithm 5 to remove the nodes instead of the edges, at each algorithm iteration. However, this scenario prevents us from determining some other interesting relationships among clusters. Figure 5.6 presents such a problematic situation: The first dominant graph that is extracted is $\{B, C, F\}$. If we then remove nodes B,C and F from the graph, we miss the other dominant set C, D, G. Simply removing the edges $B - C, C - F, B - F$ avoids such a situation, as $\{C, D, G\}$ is then extracted in a next algorithm iteration. In addition, F will not appear in a next dominant set as it is now *isolated*[4].

[4]We remind here that a *degree* of a vertex is the number of edge ends at that vertex. A vertex of degree zero (with no edge connected) is said to be isolated.

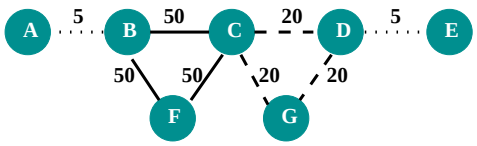


Figure 5.6: Removing Edges vs. Removing Nodes

Estimating Dominant Sets

The theory of dominant sets seems attractive. There is however a major limitation. The described algorithm would require awful amount of time to compute the weights $W(S)$ of all potential subset candidates. Assume a graph of δ nodes. This implies computing the weights of $2^{\delta} - \delta$ candidate subsets, and the weight of each node in the subset w.r.t. the subset ($w_S(k)$). At this stage, it is thus a theoretical approach to determine correlation between clusters, but it could not be applied in our concrete dataset. However, the authors in [172] have proved the tight correspondence between the problem of finding dominant sets in an edge-weighted graph and the one of finding solutions of a quadratic problem. They propose to first localize a solution of the quadratic problem with an appropriate continuous optimization technique, and then picking up the support set of the solution found. In other words, solving the problem of extracting dominant sets can be translated into the one of making a particular temporal expression converge. Solving such equations makes use of particular functions, called *replicator equations*, which are also used in theoretical biology and evolutionary game theory, since they are applied to model evolution over time of relative frequencies of interacting, self-replicating entities. The discrete time dynamical equations turn out to be a special case of a general class of dynamical systems in the context of Markov chains theory. Getting back to the pseudo-code in 5, the procedure *dominant_ set(G)* simply involves the simulation of the following dynamical system[5]:

$$x_i(t+1) = x_i(t)\frac{(Ax(t))_i}{x(t)^T Ax(t)} \tag{5.7}$$

Starting from an arbitrary initial state ($t = 0$), this replicator dynamical system is attracted by the nearest asymptotically stable point. This corresponds to a dominant set, as it has been proven in [172]. In more details, the stable vector $(x_i)_{i<V}$ corresponds to what the authors called *the weighted characteristic vector* x^S: a non-empty subset of vertices S admits a weighted characteristic vector x^S if it has non-null total weight W(S). x^S is defined as:

[5] $x(t)^T$ being the transposed vector of x(t).

$$x_i^S = x_S(i) = \begin{cases} \frac{w_S(i)}{W(S)} & \text{if i} \in \text{S} \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

The demonstration of this property can be found in [199]. It is worth mentioning that by definition, dominant sets always admit a weighted characteristic vector. Characterizing the complexity of this approach appears to be difficult since it involves the simulation of dynamical system. However, we have noted experimentally that the system converges quite quickly ($t < 1000$) when applied to all examples presented in the next Section. Furthermore, in our experiments, the initial vector choice does not apparently impact the final results. This property has been checked by running different tests with various initial vectors. All have converged in a very short amount of time to the same solution.

Stopping Criterion

We have mentioned at the beginning of this section and in Algorithm 5 a function called *stopping_ criterion()* that stops the execution of the algorithm after extracting *nontrivial maximal cliques*. More generally, it is important to specify some restrictions that have been imposed to the method. One is the stopping criterion. Indeed, the algorithm stops, theoretically, when the simplified matrix does not contain edges anymore. However, the last dominant sets are not really meaningful. In addition, performing the algorithm while some edges remain brings a major drawback: the technique will produce dominant sets, whatever their concrete weight, compared to the others. It can be easily imagined the scenario where all nodes are isolated, except two, which are linked by an extremely small weight. The algorithm will automatically extract them as last dominant sets, whereas we would not consider them as meaningful in the context of this study.

One solution would consist in specifying the maximum number of dominant sets to be extracted (or *numerus clausus*). This solution works fine with simple graph samples, but some limitations remain:

- The number of meaningful dominant sets varies depending on the analyzed graph. It is thus hard to adapt this value between graphs, and, as we will explain later, to combinations of graphs.
- Among the extracted dominant sets, we could have meaningless ones within those selected by the *numerus clausus*.

Another approach consists in setting a few global criteria that the algorithm must check.

- The matrix does not contain edges anymore. All nodes are isolated. This case is quite obvious.

- The algorithm generates a too large dominant set. Υ_1 is the maximal coverage value of the dominant set. If the ratio between the number of nodes contained in the extracted set and the total number of nodes in the graph exceeds Υ_1, the algorithm stops, as the dominant set is not pertinent (or at least the chosen characteristic). The value we set is $\Upsilon_1 = 75\%$. This case has not been observed so far, as the cliques we extract are (for this time writing) quite small. This phenomenon can depend on the similarity matrices which have been used. Thus, a uniform or non discriminatory matrix would be excluded by this criterion. The *maximal coverage value* will also be used in the next Sections to represent the relative size of each dominant set among all IPs contained in a cluster. It is also important to note that in such specific situation, the complementary set would be also worth being investigated. Indeed, if 90% of the clusters are very similar with respect to a given characteristic, it would be relevant to understand why the remaining 10% clusters are not.

- We observe that most of the last extracted dominant sets are couples. They are extracted until none remains, that is, all nodes become isolated. However, these couples might present limited interest, especially if they all share the same weight value. We thus precise that the last couples with same values are not considered as dominant sets in the following. An illustration is provided in the next paragraph.

A Short Example

An illustration is better than a long explanation. Thus let us consider the analysis matrix presented in Figure 5.7. The initial weighted graph linking the five involved clusters can be found in Figure 5.7(a). We intuitively observe that the subset of vertices {1,2,3} is dominant, and this may be intuitively explained by observing that the edge weights internal to that set (60,65 and 70) are larger than those between internal and external vertices (which are between 5 and 25). As this example suggests, the main property of a dominant set is that the overall similarity among internal nodes is higher than that between external and internal nodes, and this fact is the motivation of considering a dominant set as a particular grouping of nodes (i.e. clusters). More precisely, the algorithm converges to the following *weighted characteristic vector* x^S, $S = \{1, 2, 3\}$:

- $x_{\{1,2,3\}}(1) = 0.3360$
- $x_{\{1,2,3\}}(2) = 0.3062$
- $x_{\{1,2,3\}}(3) = 0.3579$
- $x_{\{1,2,3\}}(4) = 0$

- $x_{\{1,2,3\}}(5) = 0$

The edges involved in the first dominant set are removed from the graph as shown in Figure 5.7(b). A new dominant set, composed of {1,4,5} emerges. The new *weighted characteristic vector* x^S, $S = \{1, 4, 5\}$ is then:

- $x_{\{1,4,5\}}(1) = 0.3636$
- $x_{\{1,4,5\}}(2) = 0$
- $x_{\{1,4,5\}}(3) = 0$
- $x_{\{1,4,5\}}(4) = 0.3636$
- $x_{\{1,4,5\}}(5) = 0.2727$

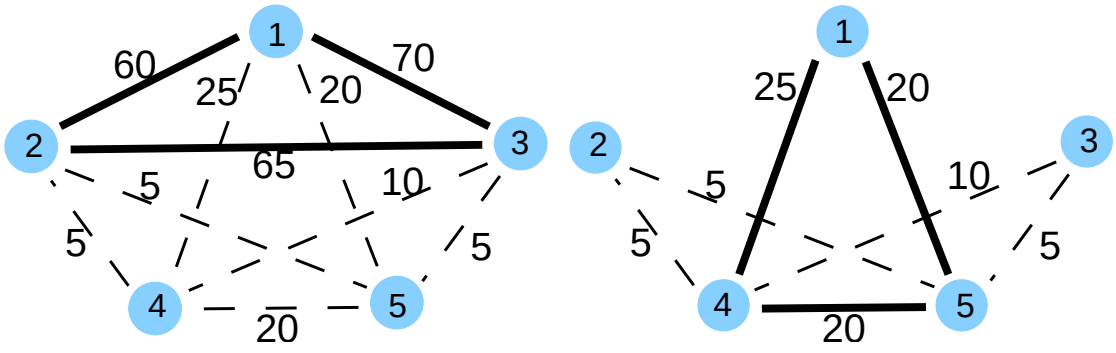
The dominant set is also removed in the next step presented in Figure 5.7(c). A dominant set made of two nodes (3 and 4) is finally extracted from the resulting graph.

Without the stopping criterion described in the previous paragraph, the next extracted dominant set would be {2,5}, as illustrated in Figure 5.7(d). However, we are not convinced by the correlation between these two nodes. First, their shared edge has quite a small weight. Second, this value is not clearly *dominant*, if it is compared with the other remaining weight values, that are exactly similar. According to the stopping criterion, it is thus not identified as a dominant set.

In conclusion, the algorithm extracts the three dominant sets in this order:

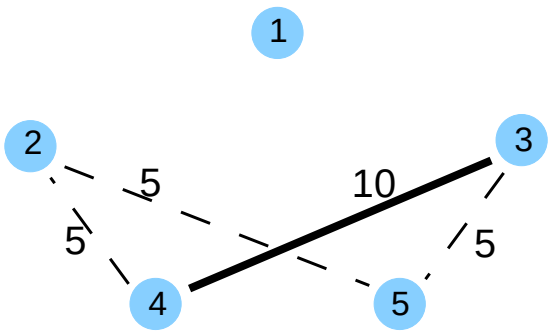
1. {1,2,3}
2. {1,4,5}
3. {3,4}

This example ends the theory section. The reader who took enough time to go through it now understands that the whole technique relies on the so-called weighted similarity matrices. We do not pretend, in this document, to show all potential matrix candidates. It is even more impossible that the number of candidate matrices is only limited by the researcher imagination and the available resources. Based on the experience we have acquired along the project, we propose a few of them, which are carefully motivated by preliminary studies. Such studies are presented in Section 5.1, and the resulting matrices we have decided to manipulate are detailed in Section 5.4. The underlying expectation is at least to find by this automatic approach an enriched version of the results of the preliminary studies.

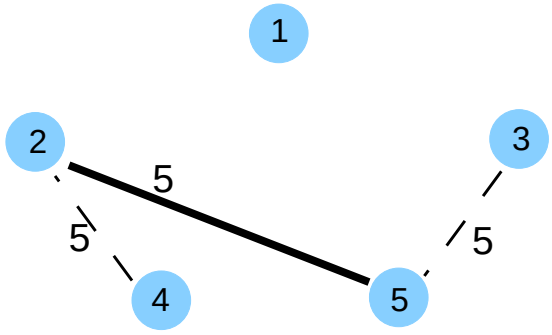


(a) Step 1

(b) Step 2



(c) Step 3



(d) Step 4

Figure 5.7: Dominant Set Extraction: A Simple Example

5.3 Building Similarity Matrices

5.3.1 Characteristics Representations

This section aims at detailing the few similarity matrices which have been used in this thesis. As it has been mentioned in Section 5.2.2, there are several steps which lead from the abstract characteristic we intend to study on the clusters to the similarity matrix filled with figures.

The first step consists in representing the characteristic of the analysis. We have considered, over this thesis, three major representation types:

- Representation 1: $\rightarrow$ *Vectors*: they are used to compare the distribution of a certain class of attributes over the clusters. Assume for instance that Characteristic $\mathcal{C}$ can be expressed by the set of Attributes A, then each Attribute A_i is a new dimension of that vector. The number of Large_Sessions associated to that cluster, and which share the property of Attribute A_i $(\forall i)$ gives the dimension value for the vector.
- Representation 2: $\rightarrow$ *Intervals Intersection*: the intersection is computed to determine which Large_Sessions among several clusters share a same property. This representation has been applied, for instance, to compare the common IP addresses which have been observed with different fingerprints at different periods (see Section 5.4.6).
- Representation 3: $\rightarrow$ *Time Series*: a particular representation of time series, using the SAX method, has been applied to compare the evolution over time of clusters. This time series representation will be detailed in Section 5.4.7. We group the description of this representation and the associated distance into a single section in order to ease the understanding of both of them. It is however important to keep in mind that they are two distinct steps (so potentially alterable) which lead to a matrix expressing temporal similarities.

Scenarios presented in Section 5.4 exemplify these three representations.

5.3.2 Potential Distances

Discussion

From the beginning of this Section, we have indifferently used terms like *similarity measures* or distance functions. The thesis cannot include all and one distance functions, but it is often more easy to find distance functions in the literature, depending on the

type of the performed analyses. The *dominant set* method, however, must be applied with similarity matrices, as indicated in Section 5.2.4 and not distance matrices. There are some guidelines that all similarity functions should follow. These guidelines are intrinsically linked to what they must express. We must pay great care to testing whether these mathematical techniques are actually appropriate when dealing with clusters. For instance, some of the properties of mathematical metrics are not always ideal for describing distances between clusters. The next Sections present a few case studies in what can go wrong if we are not careful and sensitive to the goal of our work, which is not using mathematical ideas of distance but inferring similarity of meaning in attack fingerprints (clusters). To make things more clear, we have distinguished two particular notions: one is called *distance*, and one *similarity*.

Distance Functions

A distance function allocates a value to a pair of points in a space which indicates how far those points are from one another. Let S be a finite set. A distance on S is a function D: S x S $\mapsto \Re_+$ satisfying the following two properties:

- Symmetry: $\forall v, w \in S$, $D(v, w) = D(w, v)$
- Non-Negativity: $\forall v, w \in S$, $D(v, w) \geq 0$

The most standard distance measures in mathematics are called *metrics*, which must satisfy certain conditions or *axioms*. However, we do not impose here that the distance functions obey the triangular inequality[6], and self similarities D(i,i) are not defined[7]. We have used *distances* to build so far the analysis matrices $\mathcal{A}\_p$ summarized in Table 5.4.

We present here as illustration some distances that have been frequently used over our experiments. Once again, they are not the unique ones, but the framework gives a good opportunity to compare resulting analysis matrices. They will be named *Distance 1*, *Distance 2*, *Similarity 3* and *SAX Distance* respectively in the following Sections. They are associated to the three distinct characteristic representations described in Section 5.3. The two *Distance 1* and *Distance 2* have been chosen with a particular underlying idea: the correlation between attack fingerprints, or clusters, is often transposed to a comparison and proximity evaluation of their respective distributions according to some particular attributes. The distributions are simple vectors of dimension n, n being the number of possible attributes in the distribution. This corresponds to *Representation 1*. *Distance 3* is a distance dedicated to comparing interval intersections, that is *Representation 2*. *Distance SAX* is the distance related to the particular representation of time series (symbols) we have previously described. It thus corresponds to *Representation 3*.

[6] $D(a, b) + D(b, c) \geq D(a, c)$

[7] The similarity matrices we are using must have null values all along their diagonal.

Both the representation and the distance are presented in the very same Section, to ease the understanding of this method.

Distance 1

The first distance we decide to use in order to compare two distributions is the simple euclidean one. This distance between two points x and y in an Euclidean Space $\Re^n$ is given by:

$$d_1(x, y) = |x - y| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{5.9}$$

Distance 1 gives a good feeling of the closeness of each cluster within the attribute dimensions, but it does not provide any idea of *which* attributes are more *involved* than others in the distribution. In other words, as presented in Figure 5.8, three distributions can have a close distance but not the same coordinates. Cluster 1 has no attribute coordinate, while Cluster 2 and Cluster 3 share a same direction along with Attribute 3. This distance is thus interesting but limited, and it justifies the choice of complementing it by Distance 2.

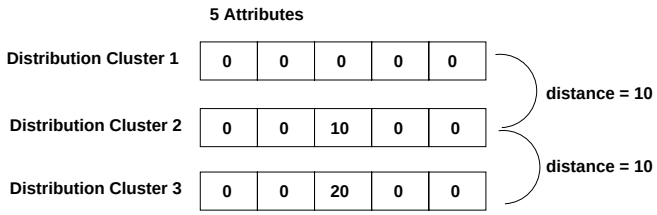


Figure 5.8: Peak Picking Distance between Distributions

Distance 2

This distance also aims at comparing distributions. Indeed, there are many possible distances to compare two groups of values. Some of them being to compute the largest cross-distance, the shortest one, the one between centroids, etc [70]. A distribution is represented as a simple vector, the dimension being the distribution attributes, and the values being the attribute frequencies. For each of this vector, we apply a peak picking technique, which aims at picking most frequent peaks. All peaks that are η times more intense than the average distribution are extracted and ordered in decreasing order[8]. This list of peaks is then compared with the list of another distribution. A distance of 1 characterizes a complete match of their ordered list of peaks, otherwise its value remains

[8] We consider $\eta = 2$ in the report, as most of the distribution are not uniformly distributed and they present clear peak activities like the one illustrated in Figures 5.8.

null. In a more formal way, the distance is defined as:

Let $\vec{d_1}$ and $\vec{d_2}$ be two vectors of size n

$$[pp_1] = peak\_picking(\vec{d_1})$$
$$[pp_2] = peak\_picking(\vec{d_2})$$

$[pp_1]$ and $[pp_2]$ being ordered sequences of peaks

$$dist(\vec{d_1}, \vec{d_2}) = \begin{cases} 0 & \text{if } [pp_1] = [pp_2] \\ 1 & \text{otherwise} \end{cases} \quad (5.10)$$

Two ordered sequences $[pp_1]$ and $[pp_2]$ are equal, if for each i, $pp_1(i) = pp_2(i)$. The peak_picking function is detailed by Algorithm 6:

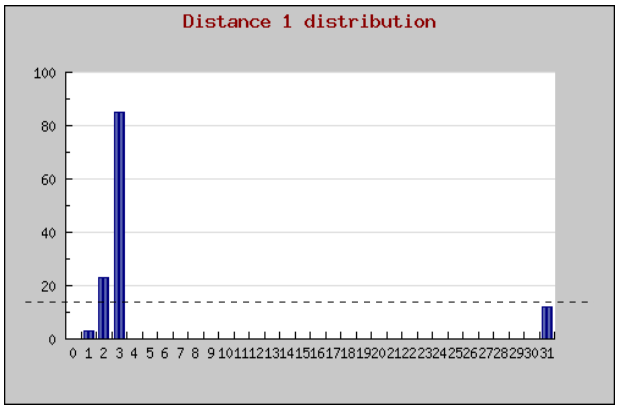


Figure 5.9: Peak Picking: Concept and Example

Algorithm 6 Details of the *peak_picking* Function

Require: A distribution vector $\vec{d_x}$ of size n
Ensure: Ordered list of prevalent peaks pp_x
1: Compute the average value of $\vec{d_x}$
2: $\bar{d_x} = \frac{1}{n} \sum_i d_x(i)$
3:
4: **for all** dimensions $d_x(k)$ **do**
5: **if** $d_x(k) > \eta \bar{d_x}$ **then**
6: $pp_x \longleftarrow pp_x, k$
7: **end if**
8: **end for**
9: Order pp_x by decreasing distribution frequency

Figure 5.9 represents a given distribution of one cluster. The *peak_picking* algorithm will then return the following list: $pp_x = \{3, 2\}$. If we now consider again the example of Figure 5.8, Cluster 1 would be found with $pp_1 = \emptyset$, Cluster 2 with $pp_2 = \{2\}$ and Cluster 3 with $pp_2 = \{2\}$. Thus, only Cluster 2 and Cluster 3 are correlated with a distance of

1. The frequency is not rounding. However, it might happen that two frequencies share an equal value. In this case, the respected order is the one given by the list of attributes, in order to ensure in this case a same peak sequence. The presented distance fulfills the symmetry and non-negativity of each distance property. This is however not a metric insofar as the triangular inequality is not satisfied. It also does not give any information on the amplitude of the peaks.

These two distances have been applied by looking at particular distributions among clusters, detailed in the following paragraphs. They include:

1. The distribution of countries as origins of attacks
2. The distribution of targeted Environments
3. The distribution of Operating Systems having launched attacks against the honeypot sensors
4. The distribution of IP distances between attacking machines and targeted honeypot sensors

Similarity 3

This similarity aims at comparing the intersections between two clusters. It can be summarized in theory like the following:

Let A and B be two distinct sets,

$$sim_{IP}(A, B) = \frac{A \cap B}{A + B - A \cap B} \tag{5.11}$$

To summarize, the more common elements sets A and B have, the more important this value becomes. The minimum is 0 and the maximum is 1. This similarity function has been applied for a very particular analysis: the common IP addresses that have been observed using several fingerprints at different dates. This similarity matrix is justified and detailed in the following Section.

SAX Representation and SAX Distance

An interesting correlation among clusters is their temporal evolution. The *Leurré.com* database contains data collected for many months (even years), thus it seems important to compare how attack fingerprints evolve over time. It seems all the more relevant that many worm models characterize different steps in the life cycle of a worm (generally 3:

Slow−Start Phase, Epidemic Phase and Slow-Finish Phase, as described in [51]). How many fingerprints follow such pattern? Are these models really accurate? To answer all these questions, we have built an analysis matrix expressing some *temporal similarities* among clusters. The major problem is to formulate such *temporal similarity*. We have presented in [183] an interesting method that aims at comparing the time evolution of clusters. The Time Series method that has been applied is called the Symbolic Aggregate ApproXimation (SAX), and as already been proven efficient in a large variety of domains. The authors propose in [142, 73] a symbolic representation for time series, that allows for dimensionality reduction and indexing with *lower-bounding* distance measure. In classic data mining tasks such as clustering, classification, indexing, etc, SAX is as good as well-known representations such as Discrete Wavelet Transform (DWT) and Discrete Fourier Transform (DFT), while requiring less storage space [142, 73].

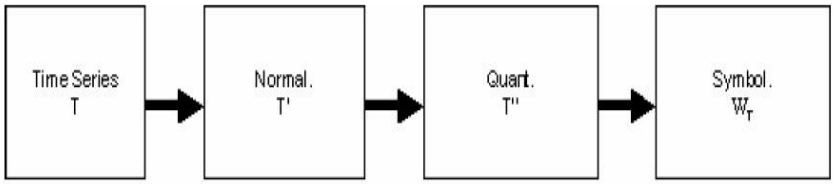


Figure 5.10: Time Series Analysis: SAX-Based Steps

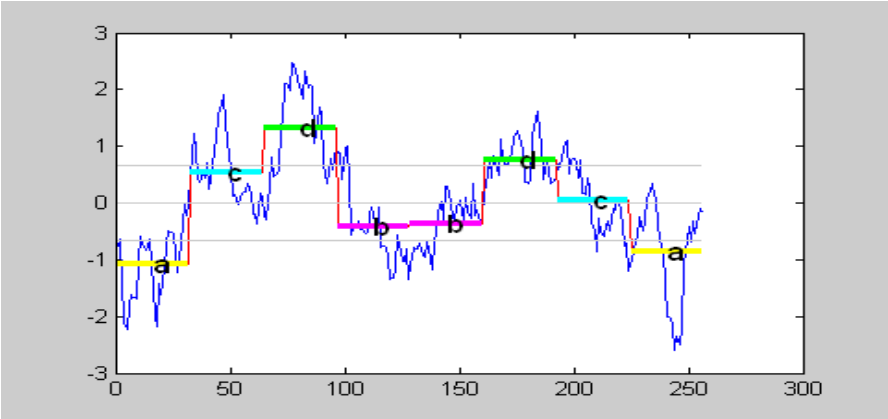


Figure 5.11: Application of the SAX Steps on a Time Series

In the configuration we use, each Cluster C_k is associated to a given time series Ts_k. The steps of the SAX method are represented in Figure 5.10 with the corresponding notations. We invite the interested reader to look at the full method description for more details on each of these steps, which are respectively the Dimensionality Reduction, the Discretization and the Symbolic Representation. The dimensionality reduction is known as the Piecewise Aggregate Approximation (PAA [125]), or Segmented Means [238]. The discretization technique is built on the creation of *breakpoints*, that determine symbols with equiprobability. Once the breakpoints have been obtained, the serie can be discretized in the following manner. All the PAA coefficients that are below the smallest breakpoint are mapped to the symbol **a**, all coefficients greater than or equal to

the smallest breakpoint and less than the second smallest breakpoint are mapped to the symbol **b**, etc. Figure 5.11 illustrates the idea.

The distance between two SAX representations (as the one in Figure 5.11) C_k and C_j of length n reduced into w symbols, is then given by the following distance function detailed in [142, 73]:

$$d(k,j) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (dist(c_{k,i}, c_{j,i}))^2} \tag{5.12}$$

The subfunction *dist()* can be implemented using a lookup table that gives the distance value between each character of the alphabet. An example of such a table is given in Figure 5.12 for an alphabet of cardinality 4. The distance can be read off by examining the corresponding row and column. For example $dist(a,b) = 0$ and $dist(a,c) = 0.67$. The details on how to build such a lookup table can be found in [142, 73].

	a	b	c	d
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

Figure 5.12: Example of a Lookup Table for an Alphabet of Cardinality 4

It is important to notice that SAX, or at least the discretization and symbolization steps, could have been implemented as an original peak picking technique. The peaks can be extracted by looking at all attributes that have the highest symbols (d in Figure 5.11). We report the interested reader to [194], in which we detail the whole SAX analysis.

Similarity Measures

A similarity measure is the converse of a distance function. Similarity functions take a pair of points and return a large similarity value for nearby points, a small similarity value for distant points. If A and B are highly similar objects, than intuitively they have

small distances. A contrario, a large distance might induce that A and B are similar. At some points, it will be more important to claim that they are similar if this large distance is rare, instead of having the majority of distance values in the same range. As an illustration, a A mark will be more prestigious for a student if he is the only one of his promotion getting it than if 90% manage to reach this mark. The attentive reader will notice that we have mixed both notions in the previous paragraphs. *Distances 1,2 and SAX* are three distance functions, whereas *Similarity 3* is, at its name indicates, a similarity measure.

One way to transform a distance function into a similarity measure is to take the reciprocal, the standard method for transforming between resistance and conductance in physics and electronics. There exist many other distributional similarity measures. A good start reference can be the state of the art presented by Lee et al. in [136].

To address the problem of transforming a distance into a similarity , we have used the following transformation: let apply an Analysis $\mathcal{A}\_p$, the vector is made of n distance values $d_1, d_2, ..., d_k, ..., d_n$. The similarity corresponding to the distance d_k is computed as follows:

$$w_k = d_{max} - d_k \tag{5.13}$$

d_{max} is the maximum distance value found in $\mathcal{A}\_p$. The resulting matrix composed of all w_k is called $\mathcal{M}\_p$, and is the Similarity Matrix of Analysis p. The theory of dominant sets can then be applied on each of the eight $\mathcal{M}\_p$ resulting from the Distance matrices described in Table 5.4. We have introduced all the elements necessary to start presenting the similarity matrices we have used in this thesis. They are all carefully listed in the next Section.

5.4 Similarity Matrices: Applications

5.4.1 Introduction and Chosen Distances

We emphasize the fact that there is no creativity limit on building *similarity matrices*, as there is a large choice of similarity measures, and a few others can be designed on purpose to serve the analysis. We do not claim to list all of them. Interesting studies on existing distances and similarity functions can be found in [95, 124, 225, 141, 108]. We present in the following the ones that have appeared as the most simple and relevant for the preliminary studies. Each distance, or similarity function, addresses a particular analytical question, related to the previously described examples.

5.4.2 Geographical Location

Presentation

It has been shown in Case Study 1 that some clusters might present very strong relationships in terms of the originating country of the attack. It is thus important at this stage to define an analysis matrix expressing the relationship in terms of the geographical origins of the clusters attacks. Clusters that have very close percentage of IP attacks issued from the same countries should be considered more similar than those which have different ones. There are 191 countries members of the United Nations and 192 countries are recognized by the United States State Department[9]. The *Leurré.com* dataset has observed over the considered period addresses coming from 185 distinct countries. However, some countries are frequently observed, while a few remain very rare. To build simple matrices, we have decided to limit the distribution of the top 30 countries presented in Table 5.1, that stem for 91.5% of all observed Sources.

The distribution attributes

Each cluster is the gathering of IP sources sharing a same fingerprint on a Honeypot platform. It is suggested here to compute the *country frequency* as the ratio (in percentage) between attacking sources identified as coming from one particular country over the total attacking sources within the same cluster. This gives the distribution of 30 countries over each cluster. An alternative would be to choose the real number of IP sources per country instead of the ratios. The distribution would then represent two distinct information types: the ratio of countries over the cluster, and the amount of sources included in the cluster. The vector we present describes the first type of information. The second one will be expressed by another vector. Both might then be combined, as we will discuss in Section 5.5.1. The matrix resulting from the first vector category and *distance 2* is called $\mathcal{A}\_Geo$.

[9]Many sources offer different answers, and depending on the source, there are 189, 191, 192, 193 or 194 independent countries in the world today. The United Nations has 191 members (including East Timor, the newest nation) but that number does not include the Vatican, an independent nation. The US State Department recognizes 192 independent countries around the world and does not include for instance Taiwan as China claims that Taiwan is simply a province of China. The 192 countries also do not include East Timor, Palestine, Greenland, Western Sahara, etc.

5.4.3 Targeted Environments

This new vector aims at finding correlation between clusters that have targeted particular environments, in comparison with those which have been observed on the majority of Honeypot sensors. The motivation comes from the first observations made with Case Study 1 and the platform in country C. This matrix will help determining if such phenomena are also observed against other environments. The resulting matrix is called $\mathcal{A}\_Env$. The process of generating the matrix is similar to the one described for $\mathcal{A}\_Geo$. We compute each *environment frequency* as the ratio (in percentage) between the fingerprints observed on that environment over the total number of fingerprints represented in the very same cluster.

5.4.4 Attacking Operating Systems

Many malware propagate thanks to specific operating systems. A large majority of them are currently spreading over Windows machines [223]. It is even said that most of the current malware are not dangerous for old versions of Windows (Windows 3.1, 98SE, etc.). An illustration comes from August 2005, a week after Microsoft issued a patch for the Plug-and-Play vulnerability described in Microsoft's August 2005 Security Bulletin MS05−039. The Zotob (w32.zotob.worm) worm family scans the Internet looking for unpatched Windows 2000 machines, then downloads malicious code to those machines via remote access; users of Windows 95, 98, and Me are not considered to be targets. Windows XP SP2 users should be safe unless they have enabled Null sessions [3]. Users of Mac OS X, Linux, and Unix are not affected. Zotob fingerprints will typically be characterized by Windows 2000 (or undetermined) operating systems, given the passive fingerprinting analysis performed on the *Leurré.com* dataset. Is it the only one working on Windows 2000 machines only? Some other malware are also spreading over less common, but not less immune, operating systems, like Linux (Adore, Ramen, Lion worms), MAC OSes (SH/Renepo.A worm for instance), CISCO IOS ([104]) . It is thus interesting to correlate attacks that are identified as coming from these specific Operating Systems.

We compute each *OS frequency* as the ratio (in percentage) between the fingerprints observed running on that OS over the total number of fingerprints represented in the very same cluster. The resulting matrix is called $\mathcal{A}\_OSs$.

Table 5.2 gives the different OSs that have been found on this experiment. They have been chosen according to their frequency on the global dataset. Other Operating Systems are more seldomly observed. It is sometimes hard to determine with passive fingerprinting techniques the exact OS, especially if the amount of packets remains limited. A fingerprint can then be counted twice, both as *Windows 2000* and *Windows NT*, if the OS passive fingerprint looks like *Windows NT,2000*. This lack of precision is due to the passive fingerprinting tools we have used: ettercap, disco and p0f [24, 4, 7]. The details of passive fingerprinting are carefully detailed in [240]. For building the matrix, we have used the

tool that appears to be the most reliable one at this time writing: p0f. The current version uses a number of detailed metrics, often invented specifically for p0f, and achieves a very high level of accuracy. It provides four different detection modes:

1. Incoming connection fingerprinting (SYN mode, default)
2. Outgoing connection (remote party) fingerprinting (SYN+ACK mode)
3. Outgoing connection refused (remote party) fingerprinting (RST+ mode)
4. Established connection fingerprinting (stray ACK mode)

Modes 1, 3 and 4 are the most solicited ones in the configuration of our platforms.

5.4.5 Name Resolution and Regular Expressions

Introduction

The *Leurré.com* dataset contains reverse DNS[10] lookups [154]. This function normally turns an IP address into a hostname. For example, it might turn 192.168.0.5 into *host.example.com*. This property does not work in many cases, including the ones where the IP is simply not registered in a DNS server, or in the case the DNS server is not correctly configured to answer reverse DNS queries (correct DNS entry is "5.0.168.192.in-addr.arpa" in our example). A domain name usually consists of two or more parts (technically called *labels*), separated by dots. For example *host.example.com*.

1. The rightmost label conveys the top-level domain (TLD, for example, the address *host.example.com* has the top-level domain *com*).
2. Each label to the left specifies a subdivision or subdomain of the domain above it.
3. Finally, the leftmost part of the domain name (usually) expresses the hostname. The rest of the domain name simply specifies a way of building a logical path to the information required; the hostname is the actual target system name for which an IP address is desired. For example, the domain name *host.example.com* has the hostname "host".

[10]The Domain Name System, or DNS, is a system that stores information about hostnames and domain names in a type of distributed database on networks, such as the Internet. Of the many types of information that can be stored, most importantly it provides a physical location (IP address) for each domain name, and lists the mail exchange servers accepting e-mail for each domain.

For each cluster, we have decided to build two distinct distributions, which are representing the TLD and the hostname.

Distribution over TLDs

In order to build the attribute vector, we need to limit the number of possible TLDs. We thus consider as attributes all TLDs that have been observed from at least 10 observed IP addresses in the whole database. We count 178 out of all TLDs, including the *undetermined* one. We introduce this class of TLD as there is an important number of unresolved names in the *Leurré.com* dataset. More precisely, they correspond to 39% of all observed attacking IPs. Furthermore, we note that a few of the selected TLDs can be associated to a large volume of observed IPs. In the decreasing order of importance, we can cite the four major ones *.net*, *.com*, *.jp* and *.de*. *Distance 2* is also applied to each pair of TLD vectors (one vector per cluster). The resulting matrix built is called $\mathcal{A}\_TLDs$.

Distribution over Hostname types

The hostnames often reveal some information about the type of machine the IP has been assigned to. For instance, we can estimate the number of personal machines by looking at specific strings in the complete hostname. If the hostname includes strings such as '%dial%','%dsl%' or '%cable%', there is a good probability that those machines are personal computers. We have classified the machines within five major categories. We list them in Table 5.3 with their associated regular expressions. *Distance 2* is again applied, this time to each pair of hostname vectors (one vector per cluster). The resulting matrix is called $\mathcal{A}\_Hostnames$.

5.4.6 Common IPs

Another meaningful matrix is the one that reveals the percentage of common IP addresses between clusters. It can be imagined that an address A.A.A.A first launches attack Y (for example a scan), and then, a few days later come back to launch attack Z. According to the classification we made so far, address A.A.A.A appears as two *Attacking Sources* (time interval between appearance dates is longer than the defined 25-hour threshold), each of them having left a different fingerprint on the honeypot sensors.

This scenario also implies that the address A.A.A.A is not dynamic. In these cases, the two involved clusters should show up a large number of common IP addresses, even if this is not a full coverage.

There are also some worms that take benefits of ports opened by other worms. A famous example is the *Dabber* worm that exploits the same vulnerability than another one called *Sasser* in order to spread. It uses pieces of code installed by the Sasser-FTP exploit application to burrow into a PC, remove Sasser, and install a server on the infected machine to further propagate. We can expect, according to this scenario, and if the volume of activities is representative enough, that clusters associated to *Sasser* and *Dabber* will share common IPs. The requirement consists in making this analysis generic in order to find, if they exist, all relationships similar to this one. To build the analysis matrix, it appears that both Distance 1 and Distance 2 are not convenient. We cannot adapt the two initial distances in this situation, as there is no particular distribution (and vectors *a fortiori*). Thus, the resulting matrix $\mathcal{A}\_CommonIPs$ is built from the *Similarity 3*.

We have also explained in Section 4 that some sources might not be properly classified within clusters, after having applied the clustering technique. The small resulting clusters share many common IPs identified by several sources. Such relationship should appear as well with such a similarity matrix.

5.4.7 Time Series Analysis

We do not explain here the building of the matrix which express the temporal similarity between clusters. The SAX method has already been carefully described in the previous Section 5.3.2. A specific distance function which makes use of the symbolic representation of the time series has been presented. The resulting matrix is called $\mathcal{A}\_SAX$.

5.4.8 IP Proximities

IPs_Dist Definition

For this analysis, we have made use of an original vector. We use a particular comparison that returns the first bit position from which two IP addresses IP_1 and IP_2 differ, with a Big-Endian approach. This distance thus gives the i^{th} bit position between IP_1 and IP_2. An illustration is presented in Figure 5.13. The first bit which differ between $IP_1 = X.X.X.X$ and $IP_2 = Y.Y.Y.Y$ is at position 1, so the distance is 1. The obtained value is thus within the interval $[0, 32]$. This operation is performed for each pair of large_Session within a cluster and the considered vector is simply the distribution of these values over all the cluster. As a first consequence, attacks which favor specific CIDR masks should have a particular distribution with a high peak around CIDR values. Indeed, some malware have been found favoring the propagation over local networks, changing the last IP bits. Code Red II implements a similar strategy [157]. This worm will 1/8th of the time generates a random IP not within any ranges of the local IP Address. 1/2th of the time, it will stay within the same class A range of the local IP Address 3/8th

of the time, it will stay within the same class B range of the local IP Address. If the IP the worm generates is 127.x.x.x, 224.x.x.x, or the same as the local systems IP address then the worm will skip that IP address and generate a new IP address. Therefore, this worm has a particular signature in terms of IP distances. Over the whole fingerprints characterized by Code Red, the distribution should tend to the above ratios.

Code Red II has been carefully analyzed and modeled [157, 243]. Zotob worm also propagates by keeping the first 2 bytes and tries to connect to random IP addresses within the same B-class (255.255.0.0) than the compromised machine [3]. However, reverse engineering of worm codes is a time-consuming task, and it does not help determining if other malwares propagate and follow the same characteristics. To provide such answers, the matrix $\mathcal{A}\_IPprox$ is built from this IPs_Dist vectors. As an example, the distribution would be incremented by 1 for attribute number 1, after having computed the value presented in Figure 5.13 and described at the beginning of this Section.

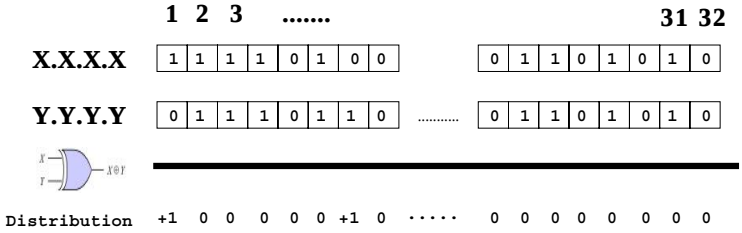


Figure 5.13: IP_Dist Computation and Distance Distribution

It is important to note that this particular technique to compute IP distances between attackers and attackees does not allow to identify the malware which propagate by switching dedicated bits in the IP address of their victim. Another approach would have consisted for analyzing such tools in computing the cumulative bit-to-bit XOR between IP_1 and IP_2. This would have been the final distribution presented at the bottom of Figure 5.13, after having performed all the XORs. This new similarity would indicate the different malware codes that change specific bits from the infected IP to propagate. Such a scenario will not show up from the previous distance. However, this second analysis is left for future work, as this scenario seems at this time writing less used by popular worms [223].

5.4.9 Summary

We have presented in the last paragraphs the *Analysis matrices* which are used with the method of *dominant set extraction*. We summarize in Table 5.4 their names and their principal features.

5.5 Derived Properties

5.5.1 Mixing Similarity Matrices

The matrices have been created, based on some analyses we intended to automatize. It could be interesting, at this stage, to check if there are groups of clusters sharing several characteristics. In other words, it would be relevant to determine all the clusters which are linked each other within distinct *dominant sets*. One solution consists in looking for the intersection of extracted *dominant sets*.

Algorithm 7 Combination of Analyses

for all $\mathcal{M}_p$ Similarity Matrix of Analysis p, $0 < p < N + 1$ **do**
  Compute $DS(\mathcal{M}_p)$ $\tilde{\mathcal{M}}_p(k)$
  the extracted dominant sets
  $DS(\mathcal{M}_p) = \{\tilde{\mathcal{M}}_p(k)\}$
  as described in Section 4.2.
end for
for all Combinations $\mathcal{C}_i$ of $\mathcal{M}_p$ **do**
  Compute the new dominant sets
  associated to $\mathcal{C}_i$:
  $DS(\mathcal{C}_i) = \bigcap_{i \in \mathcal{C}_i} \tilde{\mathcal{M}}_i(k)$
end for

The algorithm can be found in Algorithm 7. It simply computes the intersection of dominant sets extracted for each matrix individually. It can be easily proved that computing the intersection of cliques (we remind here that *dominant sets* are maximal weighted cliques) generates cliques. The algorithm works on the corresponding $2^P - 1$ combinations of analyses, P being the number of matrices. In this thesis, we present 8 matrices (see Table 5.4). It thus implies 55 different combinations of matrices. Figure 5.14 describes the situation with three analysis matrices, labeled A_1, A_2 and A_3.

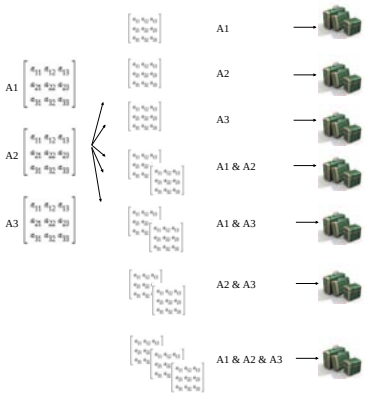


Figure 5.14: Mixing 3 Similarity Matrices: an Example

Another approach would have been to combine matrices into another resulting matrix, and extract dominant sets from it. There is however a difficulty in mixing up matrices. The mixing could be performed by computing a weighted average value of each edge coming from the initial matrices. In other words, for two analyses matrices A and B, a new resulting matrix C will be $C_{nxn} = [c_{ij}]=\frac{1}{\alpha_A+\alpha B}.(\alpha_A.A_{nxn} + \alpha B.B_{nxn}) = \frac{1}{\alpha_A+\alpha B}.[\alpha_A.a_{ij} + \alpha_B.b_{ij}]$. For the sake of clarity, we do not investigate this possibility here, and only present results from the simple intersection method described in Algorithm 7. An issue might also arise in the case where analysis matrices are not of the same size, that is, are not performed on the same set of clusters. In this situation, the method must be applied on the set of clusters common to both analyses, that is the intersection of cliques that have been used for the two analyses. Indeed, the dominant node sets are extracted with respect to other nodes in the graph. Thus, the intersection of dominant sets must be considered for the same set of nodes. For some clarity concerns, we will not investigate this possibility here, and only present results from the simple intersection method described by Algorithm 7. In other words, we consider for all following analyses the same set of clusters (the ones with size larger than 10 Sources as described in Section).

5.5.2 Algorithm Limitations

Applying this algorithm on each Similarity Matrix $\mathcal{M}\_p$ enables us to deal with any size of graphs, from small to large ones (with thousands of nodes). However, it presents some negative points that are worth being mentioned and considered.

1. There is no guarantee on the order of which the sets are found. Let imagine that two sets share an equal weight on the Similarity Matrix, there is no way to determine which one will be extracted first. This is not a real problem, however, as both will be extracted and collected.

2. The resulting dominant sets highly depend on the weights distribution in the matrix. (constant high values distances?) We have decided not to extract the overlarge dominant sets, but there is no way to handle this limitation better, except by refining the initial analysis and maybe by changing the distance function.

3. The resulting dominant sets might also be biased in the case the matrix contains too many disconnected nodes, as the dominant sets are computed as the maximal weighted cliques *within* the complete graph.

5.5.3 Validation: Relation Projection

Concept

The matrix analysis considers potential similarities among activity fingerprints, also called clusters. It would be interesting to determine if the similarities are still valid on each environment, or if they are emerging properties when looking at the fingerprints over all sensors. This implies to build the same approach than previously described but on a small cluster portion; the large_Sessions associated to this cluster and to a given Environment. We propose in this section a method to test the relationship observed between clusters per Environment (honeypot sensor). This is also an interesting approach to evaluate the relevance of the dominant sets resulting from the previous algorithms.

First Solution

We decide in the following to project each cluster onto the different environment dimensions. Such a projection is described as follows (see Figure 5.15 as an illustration): A cluster is by definition the set of IP sources sharing the same attack fingerprints against honeypot environments. We call $P_i(C_j)$ the set of sources having targeted the environment number i with the fingerprint associated to Cluster C_j. The *weight* of this subset of IP sources is defined as:

$$Wght(j,i) = \frac{card(P_i(C_j))}{\sum_k card(P_k(C_j))} \tag{5.14}$$

It represents the weight of sources in a cluster with respect to the environment. We can compute this value for each cluster of the dominant set. The average value gives the average weight of the environment representation within the cluster. Thus, high values indicate the environment is strongly represented within the cluster, while a small value indicates that cluster characteristic is not strongly represented on the environment.

Another Solution

Another solution would be to build other matrices to filter relationships in terms of environments. Each similarity function will be applied to the *Large_ Sessions* of a cluster from a given honeypot environment. This method is however very costful. It can be considered, but in some very specific cases. Let assume the database has currently E environments. The user gives directly the matrices corresponding to each cluster partition,

that is, one matrix per environment. However, the average partitioning of N clusters into βN partitions ($0 < \beta \leq E$) induces a potential increase of $\frac{N^2(\beta^2-1)-N(\beta-1)}{2}$ edges, that is, for $N = 30000$ and $\beta = 15$, 10^{11} more edges than between initial clusters. It seems more sensible to restrict this analysis to the previously extracted clusters.

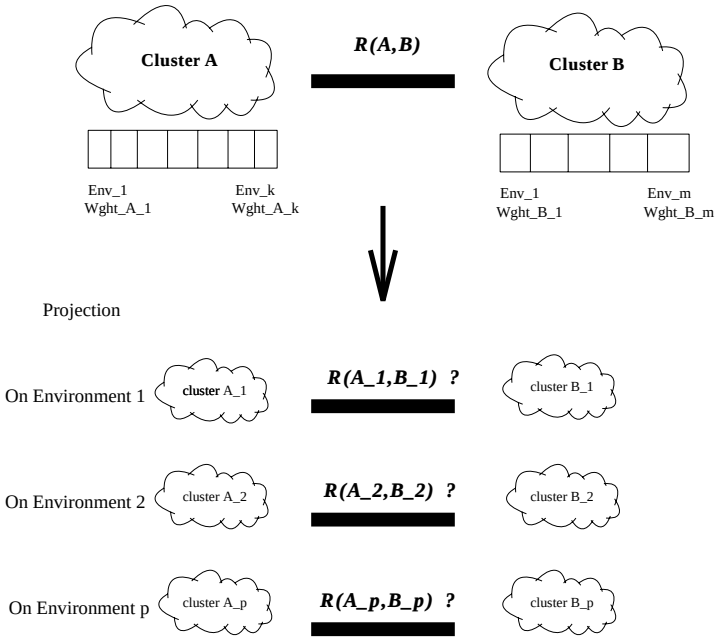


Figure 5.15: Projection on Honeypot Environments

5.6 Conclusion

Suspicious activities have been classified according to the fingerprints they leave on each honeypot sensor. The first results have shown very interesting properties and trigger the need to automate the analysis. This chapter has presented a graph-based technique to help identifying groups of activity fingerprints that share strong similarities. The preliminary analyses have helped orienting the creation of simple similarity matrices. The next chapter intends to first check that the case studies found and described in Section 5.1 are correctly found, and that all similar properties are discovered for all clusters. It is also expected at this stage to discover other interesting relationships by computing the intersection between *dominant sets* extracted from distinct matrices.

Table 5.1: Countries considered in the distribution (% Total Sources)

USA	18.2%
China	10.6%
Germany	6.5%
Taiwan	6.0%
Serbia-Montenegro	5.8%
France	4.5%
Republic of Korea	4.1%
Japan	3.7%
Canada	3.6%
UK	3.6%
Spain	3.4%
Italy	2.4%
Poland	2.2%
Russia	1.8%
Sweden	1.7%
The Netherlands	1.6%
Brazil	1.1%
Turkey	1.1%
Switzerland	1.0%
Greece	1.0%
Portugal	0.9%
Austria	0.9%
Australia	0.9%
Danemark	0.9%
Hong-Kong	0.8%
Belgium	0.8%
Mexique	0.8%
Israel	0.6%
Norway	0.5%
Finland	0.5%

Table 5.2: Considered Operating Systems used to build $\mathcal{A}\_OSs$

Chosen Operating Systems
Windows 98 SP2
Windows NT 4.0
Windows 2000 (All service packs 1-4)
Windows XP Service Pack 1
Windows XP Service Pack 2
Linux (RedHat, Debian, Mandrake: 2.4-2.6)
Solaris (versions 8 and 9)
OpenBSD (versions 3.0-3.4))
FreeBSD (versions 4.6-4.8)
Cisco IOS (all versions)

Table 5.3: Hostnames Classification based on Regular-Expressions

Hostname Category	**Regular Expressions**
Personal Machines	%dial%,%dsl%,%cable%
Mail Servers	%pop%,%smtp%,%imap%,%mail%
Web Servers	%web%,%http%
Routers	%.cisco%,%route%,%gw%
Unresolved names	%undetermined%

Table 5.4: Analysis Matrices used in this thesis

Matrix Name	**Similarity Meaning btw clusters**
$\mathcal{A}\_Geo$	Distribution of attacking countries
$\mathcal{A}\_Env$	Distribution of targeted environments
$\mathcal{A}\_OSs$	Distribution of attacking OSs
$\mathcal{A}\_IPprox$	Attacking sources IP proximities
$\mathcal{A}\_TLDs$	Distribution of attacking Top-Level Domains
$\mathcal{A}\_Hostnames$	Attacking machine types
$\mathcal{A}\_CommonIPs$	Shared attacking IPs
$\mathcal{A}\_SAX$	Temporal evolution over weeks

Chapter 6

Automated Knowledge Discovery

6.1 Preliminary Results

6.1.1 Summary

This section aims at presenting the application of the method presented in Chapter 5 on the clusters built and detailed in Chapter 4. We have considered the whole *Leurré.com* dataset representing activities during one year and a half, i.e. from April 2004 to November 2005. In brief, the traffic collected on the different *Leurré.com* honeypot sensors has been clustered according to traces left by the observed IPs on a given platform. Each cluster thus gathers all IP Sources having the very same fingerprint on at least one sensor. A contrario, it can be found a same Source in different clusters, but this scenario is rare: it would mean that a Source has been observed during the same period of time attacking several sensors in different ways. The results shown so far have led to the remark that there exists different and potentially correlated similarities between these clusters. The technique presented in Chapter 5 aims at extracting dominant sets (or *cliques*) from similarity matrices and it has been designed to automatically extract the similarities we have noticed so far among clusters. This chapter details the concrete results and the information which has been inferred from the *dominant sets*.

6.1.2 Example 1: $\mathcal{A}\_Geo$

Two important points have been explained in Section 5: first, the dominant set extraction is a technique that can work on any kind of *similarity matrices*. Second, a small number of matrices (see Table 5.4) can be easily derived from all experiments we have conducted

by hand. As a follow-up, this section intends to show that such matrices can easily help finding similarities between clusters in an automatic way, and more importantly, that the extracted similarities have been found for all clusters in the current dataset.

The first matrix which has been introduced in Section 5.4.2 is $\mathcal{A}\_Geo$. It presents an analysis of the distribution of attacking countries per cluster. Clusters coming from a few and clear identical countries will be considered as similar. We intend here to extract all clusters that present such strong relationships.

The *dominant set extraction* algorithm generates 9 cliques from matrix $\mathcal{A}\_Geo$. Their major characteristics are presented in Table 6.1. Each clique is identified by a dedicated identifier (*Clique ID*). We introduce a simple indicator (in percentage) which gives a hint of the degree of difference between clusters within a *dominant set*, named *Clique Relevance*[1]. It indicates how clusters within a dominant set differ between each other with respect to the two following attributes:

- The average percentage of distinct targeted ports among sequences of ports between each couple of clusters.
- The percentage of clusters which have different numbers of targeted virtual machines.

The more different the clusters within a dominant set are (in terms of targeted machines and ports), the more relevant we label the dominant set. In other words, it means that we put here more emphasis on clusters which have very distinct fingerprint characteristics (in terms of targeted ports or number of targeted machines) but which share a very strong common property expressed by the dominant set. We compute the *Clique Relevance* as the product of the two previous percentages (expressing a new percentage). This is not a perfect solution, but it is a first indicator to compare relances of dominant sets. It is computed as follows:

The *Clique Coverage* value provides the ratio of involved Large_Sessions within the clique out of the total number of Large_Sessions considered in the dataset. It gives a good incentive of the practical volume of Large_Sessions of the clique over the dataset. Thus, a small *Clique Coverage* would imply that the similarity (or similarities) associated to that clique is quite rare in the dataset, while a large value indicates that it is a common relationship. The *Peaks* attribute is related to our definition of similarity, which aims at picking peaks. Each clique is the manifest, as such, of a different sequence of distribution peaks. The peaks are here the most frequent attacking countries.

[1] We remind here that a *dominant set* is a particular *clique*, called the maximal weighted clique.

Algorithm 8 Computing Clique Relevance

$var1 = 0$
$var2 = 0$
Require: Clique $\mathcal{CLIQUE}_i$ be a set of k clusters $C_j, 1 \leq j \leq k$
  for all C_m, C_l, $1 \leq m < l \leq k$ clusters of $\mathcal{CLIQUE}_i$ **do**
    Compute the percentage p_1 of common ports
    Over the respective sequence of ports
    See Cluster Signatures in Section 4.4.5
    Ex: $C_m : \{445, 135, 80\}$ and $C_l : \{80\} \rightarrow \frac{1}{3}$
    $var1 = var1 + (1 - p_1)$
    Compute $var2$ as:
    if $Num\_Targets_m \,!= Num\_Targets_l$ **then**
      $var2 = var2 + 1$
    end if
  end for
  $var1 = \frac{2}{k(k-1)} * var1$ (normalized value)
  $var2 = \frac{2}{k(k-1)} * var2$ (normalized value)
  Clique Relevance$= var1 * var2 * 100$ (combined percentage)

It can be noted that there is a prevalence of clusters very specific to Asian countries, as we have observed manually. Peak extraction is quite stable, insofar as most of the peaks are limited to 2 or 3. Furthermore, we note that the magnitude order of the peak is not really important in this situation: we do not observe cliques corresponding to peaks {P1,P2} and then {P2,P1} for instance, that is different sequences of similar peaks[2].

6.1.3 Example 2: $\mathcal{A}\_Env$

The second matrix we have defined in the previous section is $\mathcal{A}\_Env$. Peaks are, in this case, sequences of Environment IDs. The results of the *dominant set extraction* algorithm generates this time 12 cliques. They are all presented in Table 6.2, with the same column definitions than those already used in Table 6.1.

We note from Table 6.2 that peaks are various and not numerous for each dominant set. Six cliques involve a single environment. As an illustration, we note that 30 distinct fingerprints (or clusters) are specific to platform 20, and 28 are only observed on platform 6, etc. This confirms once more the distinctive nature of some attacks. All those which have been observed on a unique set of honeypot sensors appear in the list of Table 6.2.

[2]In addition, we note that only 8 countries (CN, US, YU, GR, JP, KR, CA, TW) out of 192 appear in the cliques! CN appears in 6 out of 9 cliques, YU in 2, JP in 2, KR in 2 and all others in only one clique.

Table 6.1: Cliques obtained from Matrix $\mathcal{A}\_Geo$

Clique ID	# Clusters	Clique Relevance	Clique Coverage (%)	Peaks
ID 1	20	61.7	2.17	{CN}
ID 2	14	50.4	2.08	{CN,US}
ID 3	12	6.5	0.95	{YU}
ID 4	11	8.8	0.82	{YU,GR}
ID 5	10	43.4	1.78	{CN,US,JP}
ID 6	6	58.7	0.49	{CN,KR}
ID 7	10	8.1	1.98	{CN,CA}
ID 8	4	33.4	0.39	{CN,KR,JP}
ID 9	9	37.6	0.98	{CN,US,TW}

Table 6.2: Cliques obtained from Matrix $\mathcal{A}\_Env$

Clique ID	# Clusters	Clique Relevance	Clique Coverage (%)	Peaks
ID 1	30	3.5	4.62	{20}
ID 2	28	12.3	2.39	{6}
ID 3	20	13.5	3.00	{20,8}
ID 4	18	31.8	2.39	{32}
ID 5	14	5.6	2.01	{20,25}
ID 6	26	31.9	3.88	{25}
ID 7	43	4.1	6.42	{6,31}
ID 8	10	54.3	0.97	{8,6}
ID 9	8	8.3	0.93	{6,8}
ID 10	14	5.1	1.60	{23}
ID 11	12	17.3	2.28	{10}
ID 12	5	61.2	0.42	{25,20,36}

6.1.4 Example 3: $\mathcal{A}\_Geo$ vs. $\mathcal{A}\_Env$

We present in Table 6.3 the result of the intersection between cliques obtained from the two previous analyses. Rows are the cliques presented in Table 6.1 from the $\mathcal{A}\_Geo$ analysis, while columns are the cliques described in Table 6.2. The values in the cells indicate the number of clusters two cliques have in common. The other value (between brackets) is computed as the number of clusters in the intersection divided by the minimum cardinality of the root cliques. The value is given as a percentage (%). This percentage gives an indication of the number of clusters within an initial matrix that are also found after the intersection. A value of 100% means that all clusters sharing characteristic A also share characteristic B when intersecting matrices $\mathcal{A}\_A$ with $\mathcal{A}\_B$[3].

[3]Assuming that the number of clusters in A is smaller or equal to the number of clusters in B.

Table 6.3: Clique Intersection from $\mathcal{A}\_Env$ and $\mathcal{A}\_Geo$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	0	0	0	0	0	4 (20%)	0	0	0	0	0	1 (20%)
2	0	0	0	0	0	0	0	0	0	0	1 (8.3%)	1 (20%)
3	0	7 (58.3%)	0	0	0	0	0	0	0	0	0	0
4	0	7 (63.6%)	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	2 (20%)	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	1 (11.1%)	0	0	0	0	0	0	0	0

The intersection of the clique ID 3 from $\mathcal{A}\_Geo$ and ID 2 of $\mathcal{A}\_Env$ is a new clique containing 7 clusters. These 7 clusters are 7 distinct fingerprints which have been issued from Serbia-Montenegro on the Environment 6. Table 6.4 provides a few details associated to each Cluster. It highlights the diversity of the attacks identified thanks to this method, but also the surprising fact that all fingerprints have been observed against a Microsoft port (445). The cluster details, however, indicate these activities have nothing else in common[4]. Yet, they are only issued from one specific country to a unique platform. The 7 fingerprints can be presented to the administrator in charge of the Network hosting the honeypot Environment 6. It is again important to notice that we have found here all fingerprints that share both a strong correlation in terms of origin of the attacks and of targeted environments. The method has enabled us to find all of them, whereas in our preliminary studies, we had found only one, by a tedious process of trials and errors.

Table 6.4: Clusters from $\mathcal{A}\_Env$ ID 2 and $\mathcal{A}\_Geo$ ID 3

Cluster	**Ports Sequences**	**Avg Duration**	**Number VMs**	**Rcv payload**
1	{445}	7h	1	0 byte
2	{445}	15s	1	40 bytes
3	{445}	5s	1	0 byte
4	{445}	15min	2	0 byte
5	{445}	15min	2	240 bytes
6	{445}	2min	2	105 bytes
7	{445}	10s	2	0 byte

[4]The other cluster parameters are also very different.

6.1.5 Time Correlation between Fingerprints

About the $\mathcal{A}\_SAX$ Matrix

Table 6.5: Cliques obtained from Matrix $\mathcal{A}\_SAX$

Clique ID	# Clusters	Clique Relevance	Clique Coverage (%)
ID 1	9	2.1	3.06
ID 2	5	11.7	0.65
ID 3	7	2.5	3.04
ID 4	4	28.2	0.40
ID 5	5	12.4	0.40
ID 6	3	67.8	0.31
ID 7	4	0	0.39
ID 8	3	35.1	0.61
ID 9	3	0	0.98

We present in Table 6.5 the result of the *dominant set extraction* algorithm applied to the matrix $\mathcal{A}\_SAX$. This matrix gives the similarity between fingerprints in terms of time series as computed with the SAX technique (see Section 5.4.7). We limit the details to the nine first dominant sets, whereas 38 dominant sets have been extracted in total. There is a maximal size of 12 clusters for one particular dominant set. 32 of them also group no more than 5 clusters. The method has been applied with the following parameters, justified in [194]:

- Alphabet Size: 5
- Compression Ratio: 8

We have chosen a compression ratio equal to 8. The fingerprint activities observed within the same fixed time window of 8 days are thus counted together. There is one SAX symbol for each value. This simply means that the evolution of the *activity fingerprints* are compared per period of 8 days, instead of a *per day* granularity. One motivation is that most of the activities , except about a hundred ones which are large enough, have a time series which is too irregular: the time series are often made of long periods of inactivity, and, conversely, very intensive periods. Furthermore, there is no specific requirement for this time scale. A discussion about compressions ratios is presented in Section 6.1.6.

There are a few surprising cliques, showing explicitely that clusters share similar time evolutions. It is in agreement with the preliminary remarks presented in Section 5.1.4. It

is important to note that the probability of getting one similarity out of K symbolized time series of size w, with the previously chosen parameters is:

$$P = \frac{K.(K-1)}{2}.(\frac{13}{25})^w \tag{6.1}$$

The probability of getting one similarity out of 600 time series is thus smaller than 10^{-6}. A first remark is that the size of each clique is relatively small. The largest clique does not include more than 9 clusters. The others are limited to three or two clusters. On the other hand, it is also very surprising to find so many similarities for such a long period. Table 6.6 presents for each clique the different ports associated to each of its cluster. *Dominant set* with ID 9 corresponds to the scenario described in Section 4. The tool has launched successive attacks against two closed ports on all virtual machines, respectively ports 5554 and 9998[5]. Two small clusters have been created, one associated to the ports sequence {9898}, and one to sequence {5554}. They are both residues of losses, and have been identified by means of the clusters temporal similarity. As we have mentioned in Chapter 4, the task of interpolating losses is quite hard. Such a result, however, shows that this task can be addressed in original ways.

Table 6.6: Some cliques obtained from Matrix $\mathcal{A}\_SAX$

Clique ID	**Ports Lists**
ID 1	{80},{139}
ID 2	{139},{1433}
ID 3	{1434_*udp*},{445, 135}
ID 4	{1433},{1434_*udp*},{445, 135}
ID 5	{80},{1434}
ID 6	{445}
ID 7	{445},{135},{5000},{6129}
ID 8	{80},{22}
ID 9	{9898},{5554},{5554, 9898}

Crossing Matrices: $\mathcal{A}\_SAX$ with $\mathcal{A}\_commonIPs$, $\mathcal{A}\_Hostnames$ and $\mathcal{A}\_OSs$

This section aims at presenting the results of intersecting the dominant sets obtained from the different matrices introduced in Chapter 5. Table 6.7 represents the details of the intersection between $\mathcal{A}\_SAX$ and the other matrices. The colums provide the

[5]It is worth mentioning here that many web sites like [145] associate activities on ports 5554 and 9898 to the dabber worm. They precise that the worm first send its exploit via port 5554, and then scans port 9898 to check that its backdoor is correctly opened. It is precisely said that: *Sequential scans on port 5554 and 9898 are an indicator of a dabber infection.*In our situation, both ports are closed, thus this scenario cannot be possible and the clique indicates another activity over the very same ports.

number of common clusters, as well as the percentage of clusters existing in the initial $\mathcal{A}\_SAX$ dominant sets, and which are still correlated after the intersection with the other matrices[6].

Table 6.7: Intersection btw $\mathcal{A}\_SAX$ and other matrices

Intersection $\mathcal{A}\_SAX$	**# Common Clusters**	**% initial clusters**
with $\mathcal{A}\_commonIPs$	7	6.1%
with $\mathcal{A}\_Hostnames$	35	30.7%
with $\mathcal{A}\_OSs$	102	86.5%

Without going into the details of each intersection, it seems that clusters which share temporal similarities also share either common IPs or similar patterns of hostnames. The crossing with $\mathcal{A}\_OSs$ also confirms that most of these clusters are issued from *Windows* or *undetermined* machines, as the percentage is a little bit higher than the average values (82.4% Windows machines and 8.4% unresolved names are found in the whole dataset[7]). This intersection, however, is quite limited at this stage and examplifies the limitations of current passive OS fingerprinting techniques, which have an important uncertainty about the OS versions (resp. kernels).

We distinguish three major scenarii from the analyses we have described in [194]:

1. **Dominant sets involving clusters that share clear relationships w.r.t. ports sequences:** while the sequences of ports differ from one cluster signature to another (see the definition of a *cluster signature* in Section 4.4.5), one port sequence is always a prefix of the other. Let PS_a and PS_b be the ports sequences associated to a pair of clusters C_a, C_b. We find in this scenario that $PS_a = (PS_b, *)$ or $PS_b = (PS_a, *)$. Such a behavior is a characteristic of sophisticated tools that always scan the same sequence of ports on a machine, but stop scanning if ever one of the ports is closed. The use of time analysis is thus a good way to find out this type of tools. It represents a costless alternative to a complex reverse engineering of the code (that first needs to be captured!) that would reveal that the tool stops scanning a machine whenever a port in the pre-defined sequence of scanned ports is closed. This property is one reason that motivates the need for enriching the diversity of the sensors configuration in a near future. It is also important to notice that this scenario can be easily cross-correlated with the attributes of the clusters from the dominant sets. The number of targeted virtual machines for all these clusters is equal to 1 or 2 (out of 3), and the port status of each virtual machine must be different.

[6] We remind here that, by definition of a clique, the intersection of two cliques can only be *one* clique or an empty set.

[7] according to p0f Passive Fingerprint tool [24]

2. **Loss interpolation:** As described in the previous Section, a few clusters of small sizes could be correlated in terms of time series with larger ones, but they are not because of insidious losses. Unlike the previous scenario, the clusters share several common ports, except that one is missing compared to the other port sequences. This scenario is also easily validated by checking that targeted ports are currently closed on the machines, and that all combinations of *missing* ports are equally distributed. Formalization of losses and interpolation approaches are two tasks which are left for future work.

3. **Dominant sets involving clusters that do not match the previous scenarii:** they stem for 12 dominant sets, out of the 38 extracted ones. They are expressions of what has been called *multi-headed* worms in [194]: These worms combine several known exploits within a single piece of software. This is not a new technique, as the very first worm, the Internet worm, did already contain several infection techniques [211]. However, the specificity of this class of attacks is that only one of the available exploits will be used to launch an attack against a given target. In other words, machines targeted by those multi-headed tools see different attacks originating from different sources that can easily be interpreted as different tools. As a consequence, the fact that several exploits have been combined within a single piece of code remains invisible to the victims as long as the malware is not captured and analyzed. If the spread of the malware is not too aggressive, its existence may remain unknown for a while. A few of these sophisticated tools have already been identified, e.g. Welchia. However, this is the result of their malicious activities on users' machines and there is a high probability that some other similar, but stealthier, tools of this type are currently active in the Internet. The identification of these tools remains a great challenge, and the comparison of attack fingerprints over time enables us to identify a few of them.

From the cliques perspective, we have identified a variant of the worm Nachi, also called Welchia [222] that exploits one of the following vulnerabilities:

- DCOM RPC vulnerability described in MS03-026 bulletin
- WebDav vulnerability described in MS03-007 bulletin
- Workstation Service vulnerability described in MS03-049 bulletin Welchia is an example of multi-headed tools. To infect other machines, it randomly chooses an IP address and then attacks it either against port 135 or port 445, but not both (it is thus a real multi-headed tool). From our platform viewpoint, traces left by machines infected by Welchia look very different. They are thus stored in two different clusters, one for the attacks against a unique virtual machine on port 135 while the other contains attacks against port 445.

Another example of such multi-headed tools is Spybot.FCD [64, 221]. This tool tries to exploit Windows vulnerabilities either on port 135, 445 or 443. In the case of Spybot.FCD,

we have thus observed three similar clusters in the clique. Welchia, Spybot.FCD or W32.Kobot.A are examples of multi-headed stealthy tools that have been studied and analyzed. Many more remain to be identified. Our time signature analysis provides a simple and efficient way to reveal their existence. It should provide valuable input to other research teams interested in studying specific attack tools and/or in reverse engineering them.

6.1.6 Checking Time Series Technique

Alphabet size Impact on the Cliques

This small section aims at showing that the graph approach presented in Chapter 5 can be used to check the clique consistency with different analysis techniques or algorithm flavors. It has been explained in [194] that the alphabet size would normally not impact severely the similarity analysis. A short comparison is presented in the following for three different alphabet sizes (or discrete degrees): 4, 5 and 6. The results are presented with different compression ratios in the next paragraph.

Compression ratio impact on the Cliques

We intend to present here the impact of different compression ratios on the method of *dominant set* extraction. Table 6.8 gives for different values of alphabet sizes (As) and compression ratios (CRs[8]) the corresponding number of dominant sets. The number of *dominant sets* remains quite stable, when changing either the alphabet size or the compression ratios, around the ones we have used ($A = 5$ and $CR = 8$). It is however interesting to notice that changing the alphabet size for a given compression ratio does not impact much on the *dominant sets*, which keep gathering the same clusters (except a few exceptions). The opposite is not true: for a given alphabet size, the different compression ratios might induce completely different sets of clusters, even if the overall number of cliques is quite similar. The reason is explained through a small example presented in Figure 6.1. The x-axis represents a given time window, while the y-axis represents the time series amplitude. With regards to compression ratio 2 (that represents the whole curve into a single SAX symbol), the two curves in this figure are correlated, as they globally have the very symbol *{a}*. Unfortunately, the compression ratio 1 (two SAX symbols for each curve) gives two different sequences of symbols, resp. *{a,a}*, and *{a,b}*. SAX is an efficient technique to determine similarities between time series, however, the granularity of the time series is also an important factor as some peak effects can be smoothed and hidden. The details of SAX analysis are described in [194].

[8]The different CR values have been chosen so that the length of the initial time series remains a multiple of the compressed one.

Table 6.8: $\mathcal{A}\_SAX$:Alphabet Sizes vs. Compression Ratios

	CR 2	**CR 4**	**CR 5**	**CR 8**	**CR 10**
A 4	2	30	31	40	101
A 5	3	28	28	38	65
A 6	3	23	27	35	58

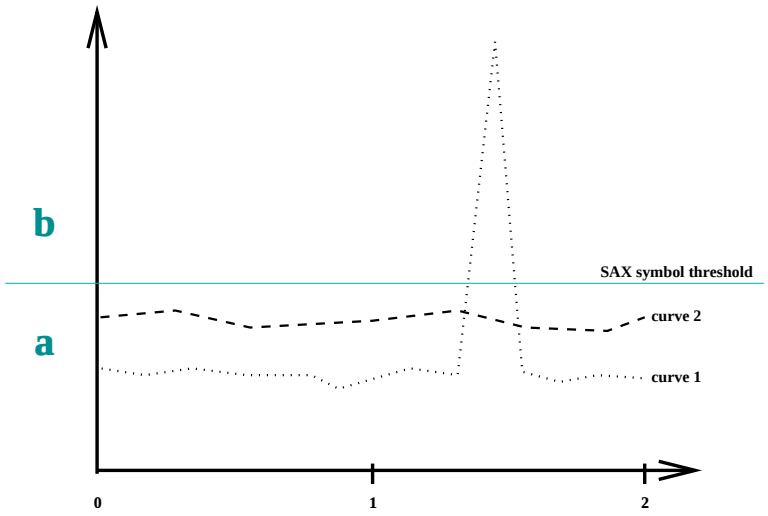


Figure 6.1: SAX and Compression Ratios

Conclusion on SAX

This section leads to two major remarks: First, comparing the resulting cliques enable us to determine the stability of the cliques according to these various analysis techniques. Second, the chosen example, SAX, with different parameter values ($CR \in [4,8]$, $AS \in \{4,5,6\}$), keeps providing very stable cliques. This confirms the results presented in [194] that the SAX approach is quite stable for sensible parameter choices. They do not impact the analysis. Future work will consist in applying SAX to other types of series, and not necessarily temporal ones, but still related to the *Leurré.com* dataset.

SAX is a clear example of the value of *dominant sets* to evaluate the impact of different techniques on the dataset. For a given technique, it is thus possible to test the extracted *cliques* and check their stability over the parameters.

6.2 Knowledge Discovery

6.2.1 Surprising Results

Each extracted *dominant set* requires at this stage an in-depth analysis. The first results have permitted to determine the attack scenarios which have been observed (somehow) *by chance* so far. In the following Section, we present two different *dominant sets* which have been picked up and analyzed. It is important to note here that all *dominant sets* are worth being investigated, as they do not express the same information. Unfortunately, due to space and time limitations, we limit their analysis to two particularly interesting ones.

6.2.2 Case Study 1

We focus in this section on two dominant sets which have been introduced in Table 6.2 with IDs 8 and 9. These dominant sets group together all *activity fingerprints* (or clusters) which have targeted two honeypot sensors in particular: namely sensors 6 and 8. The first one is placed in a European industry network, while the second one runs on an Asian academic network. The IPs in use are apparently not correlated. Furthermore, the intersections of this clique with matrices $\mathcal{A}\_TLDs$ and $\mathcal{A}\_IPprox$ are invariant. In other words, this clique is included in one dominant set extracted from $\mathcal{A}\_TLDs$, in another dominant set from $\mathcal{A}\_IPprox$. The clique however does not appear when computing the intersection with $\mathcal{A}\_SAX$ dominant sets.

The *vector* associated to the related *dominant set* of $\mathcal{A}\_IPprox$ is the peak $\{24\}$. This result can be interpreted as follows: the fingerprints associated to the considered clusters have been observed on these two unique sensors, and they have been left by machines from the /24 network they belong to. This result is confirmed when computing the intersection of the cliques with the other matrix $\mathcal{A}\_TLDs$. It provides, without any surprise, the Top level Domain of the two /24 networks. A deeper analysis of the clusters signatures reveals that all the fingerprints have been each time observed against the two virtual machines emulating Windows. They have targeted their port 135 (Microsoft Remote Procedure Call RPC[9]), but in a large variety of manners (in terms of duration, payload, etc).

This analysis thus provides an interesting example of a weird, or at least unusual, activity, and indicates a few things:

- The involved clusters only concern the two Windows machines. It is very unlikely that they have been targeted randomly. The attack was *a priori* aware of the

[9] An RPC service is a protocol that allows a computer program running on one host to cause code to be executed on another host without the programmer needing to explicitly code for this.

operating systems running on these machines. It imposes that a scanning phase from different IP sources have been launched and have been successful in determining the OS of our virtual machines.

- The attack always target port 135, but in a large variety of manners, and the fingerprints are unique to these two platforms.
- They cannot be interpreted as Windows radiations noises, as there is no service running on port 135 of the virtual machines and they do not respond to multicast requests.
- There should be different attackers, insofar as the attack has been launched several times on the virtual machines, without being successful.
- There is no evident temporal pattern between the different attacks. It is thus not a process trying to test the Windows machines in a periodic manner. It can however be a monitoring process distributed over different machines with random time intervals.

This example leads to the following conclusion:

Observation: Both class C networks where Sensors 6 and 8 are running host a common (or very similar) *botnet*. Let first recall the *botnet* definition, as given in[149, 111]: "a **botnet** is a structure consisting of many compromised machines which can be remotely managed (in general from an *Internet Relay Chat* IRC channel)". The authors report such 'many-to-many' tools, and their numerous remarks correctly match the previously described scenario. The activities we are observing characterize a very same botnet launching multiple attacks against port 135 on the same class C windows machines than the compromised ones. In addition, the *activity fingerprints* characterize this botnet, as its fingerprints have not been observed in any other honeypot sensors.

6.2.3 Case Study 2

The extraction of *dominant sets* from the $\mathcal{A}\_SAX$ matrix has also shown that some clusters are temporarily correlated, while they group activity fingerprints on different ports. One example is the clique with ID 8 on Table 6.6: it involves three clusters, linked to two distinct ports sequences {80} (one cluster) and {22} (two clusters). The clique has been observed as well with a SAX alphabet size of 6.

The analysis is not obvious, as the intersections with other matrices remain empty. However, this information, if it does not allow to conclude on the activity, helps determining what property it *does not* present. The empty intersections with matrices like $\mathcal{A}\_TLDs$, $\mathcal{A}\_IPprox$, $\mathcal{A}\_Hostnames$ and $\mathcal{A}\_Geo$ therefore indicate there is no obvious relationships among the Sources of each cluster. The empty correlation with matrix $\mathcal{A}\_Env$ also indicates that the activities have not targeted the same sets of honeypot

sensors. There is no payload, whereas ports were opened on some virtual machines. Etc. In short, there are very few properties between these three clusters, except that they all belong to the large dominant set representing *Windows* machines with $\mathcal{A}\_OSs$. This intersection, unfortunately, does not help so much the analysis.

There is a unique property shared by these activities, apart from their strong temporal relationship: they are all grouping fingerprints which target the three virtual machines.

Observation: The three clusters represent scanning activities. They have been observed on all honeypot sensors indifferently. Their strong temporal similarity tends, however, to indicate that they have a common *root cause* and are launched in parallel by a large number of machines in the wild.

Nothing else can be deduced at this stage. Further cross-analyses would maybe enriched this initial observation. In addition, it would be interesting, in this scenario, to compare the activities of the scanning IPs from a more global Internet point of view. As it was stated from the beginning of this thesis, the approach we offer is complementary to other *larger* visions, like *telescopes*, *darknets* and *blackholes*. These solutions have all been introduced in Chapter 3. They would typically enrich this case study.

6.3 Discussion

6.3.1 Abnormal Correlation and Potential Improvements

The example of SAX is rich, in the sense that it clearly shows the values and limitations of the *dominant set* method. Among its advantages, it enables to compare different parameters from one technique (see the different alphabet sizes and compression ratios). It also gives a good framework to compare different techniques. However, the weakest or most sensitive point remains the intrinsic matrix. Most of the attention must be paid to the similarity function it represents and the characteristics it intends to highlight. Moreover, the final intersections are obviously limited to the existing matrices. Further inquiries might be required after picking up one clique. If such inquiries starts being frequent, it will be relevant to also express their characteristics in terms of a similarity matrix.Another issue is the understanding of *overconsistent* intersections: in the scenario where the intersection conserves the large majority of clusters in the initial clique, it seems important to understand why a small number does not follow this rule. Either they are exceptions worth investigating, or error reflects of the chosen similarity functions. Both scenarii are hard to discriminate to date. Finally, the previous case studies have been interpreted from the two randomly chosen *dominant sets*. This interpretation, if not completely automatic, must be simplified and clear to the analyst. Next section offers a simple method to ease the interpretation step.

6.3.2 On the Labeling of Dominant Sets

Some examples have been presented so far. The method however generates all possible *dominant sets*, given the similarity matrices. The idea consists in avoiding to extract several times the *dominant sets* for a given similarity matrix. Thus, the clusters involved in a clique are labeled by the doublet made of:

1. The similarity matrix unique identifier
2. The different *dominant sets* identifiers for this matrix when the cluster is associated to one of them

It is important to note here that a cluster can be labeled twice or even more for the same similarity matrix, if it is related to several extracted cliques from that matrix. The labels are then an easy way to work on Clusters and to consider all the analyses made so far. A simple schema illustrates the labeling process in Figure 6.2.

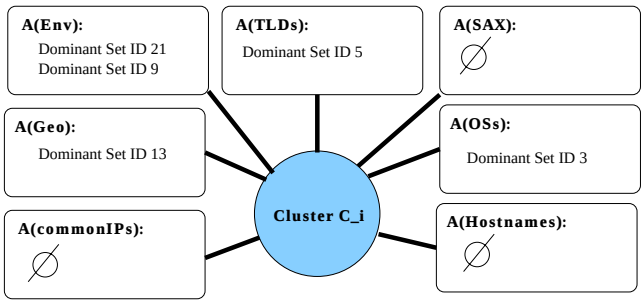


Figure 6.2: Labeled Clusters

6.3.3 On the Derivation of Observations

As the case studies have shown in Section 6.2.1, there is good value in extracting a summary of all observations obtained so far. It has also been noticed that the clusters might not be all in the same sets of *dominant sets*. It is thus worth mentioning these cases as well. To date, we derive the observations based on the Algorithm 9. It simply consists in labeling each cluster with all *dominant sets* it belongs to, and also the matrix analyses in which it was not involved (not included in any resulting *dominant set*). This labeling process enables us to complete the *cluster signature* as illustrated by Figure 6.3.

This Signature is the one of a Blaster variant. The identification task was quite easy due to a few publications on that worm, like [42, 93]. The worm exploits a remote procedure call (RPC) vulnerability of Microsoft Windows 2000 and Windows XP operating systems. The infection steps have already been presented in Figure 2.2 of Section 2.3.2.

Seven distinct clusters have been identified as Blaster variants in our dataset. Only three have been found correlated with the temporal matrix $\mathcal{A}\_SAX$ (Clique ID 21). The others have not been found for two reasons:

- They are less frequent and have not been considered in the 683 clusters considered as big enough for the SAX analysis (time series).
- Even though, they have very different temporal patterns and have been first observed four months after the first Blaster infections in August 11th, 2003.

The cluster has been merged with other clusters by the $\mathcal{A}\_OSs$. Unfortunately, the precision of our fingerprinting method does not let us precisely determine, to date, what Windows machines were mostly infected. It would have been interesting, however, to validate the Symantec claim in [21]. According to this antivirus company, the worm decides whether it will use the exploit code for Windows XP with a 80% probability, or the one for Windows 2000 with 20% probability. These two scenarii belong to two distinct clusters in our case, as the fingerprints are distinct. This is thus a variant of what has been called *multi-headed* tool in previous Section 6.1.5.

Algorithm 9 Deriving observations from the *dominant sets* labeling

```
Let C_i be one cluster
expressing an activity fingerprint on the sensors
(See Chapter 4)
Let L(C_i) be the sets of labels attached to C_i
for all Dominant Sets DS_k from Characteristic analysis A_p do
  if (p, k) ∈ L(C_i) then
    Cluster C_i has characteristic A_p
    List all other clusters linked to DS_k
  else
    Cluster C_i does not show up with characteristic A_p
  end if
end for
```

6.3.4 Summary

We can now work on the fingerprint level, and quite easily perform analyses by simply looking at labels, instead of digging once more into the data. This ends here the *HoRaSis* framework we were looking at, as it seems the current method fulfills the requirements listed in the introduction. Other interesting reporting approaches can complement the labeling previously described. The method, however, enables, at this stage, any analyst to understand the activities on the network where the honeypot sensor is placed, as well as their distinctiveness and their relationships with others. The framework is quite open to develop other analyses.

CLUSTER ID:	**IDENTIFICATION:**
1931	W32.Blaster.A (symantec), also known as: W32/Lovesan.worm.a (McAffee) Win32.Poza.A (CA) Lovesan (F−Secure) WORM_MSBLAST.A (Trend) W32/Blaster (Panda) Worm.Win32.Lovesan (KAV)

FINGERPRINT:

* Number Targeted Virtual Machines: 3
* Ports Sequence VM1: {135,4444,135,4444}
* Ports Sequence VM2: {135}
* Ports Sequence VM3: {135}
* Number Packets sent VM1: 10
* Number Packets sent VM2: 3
* Number Packets sent VM3: 3
* Global Duration: < 5s
* Avg Inter Arrival Time: < 1s
* Payloads: 72 bytes + 1460 bytes + 244 bytes

CORRELATIVE ANALYSIS:

A(SAX): clique 21
A(Env):
A(Geo):
A(Hostnames):
A(TLDs):
A(commonIPs):
A(IPprox):
A(OSs): clique 3

Figure 6.3: New Cluster Signature

Chapter 7

Conclusions and Perspectives

Conclusions

Summary

We have proposed in the previous Chapters to create a kind of *identity card* for each activity observed against the honeypot sensors. These cards express two different information categories:

1. First, the characteristics of the *activity fingerprints* which enable us to discriminate this activity from others.

2. Second, the correlation that might exist between all *activity fingerprints*. This correlation might exist for several reasons, some of them having been discussed along the thesis.

A few thousands of distinct *identity cards* have been extracted from the data collected with the *Leurré.com* project by several sensors deployed for many months in a large variety of places. These cards can then be reused for different purposes, including the following ones:

- To determine the root causes of some activities.

- To insert the information they carry for event and alert correlation, as an additional contextual information source.

- To detect new or original abnormal activities and provide meaningful information from this discovery.
- To understand the life cycles of activities over a long period of time.
- To validate assumptions and to explode myths.
- To model certain activities or improve current models.

The method we have proposed is composed of several steps, from data storage of raw packets to classification of packets into *activity fingerprints* and given prominence of relationships among these fingerprints. Several techniques have been applied and tuned to reach this abstraction level, including clustering and graph-oriented algorithms. Several steps of the method have been published along with this thesis, and have also been described in the respective chapters of the document. Some listed applications have also started being investigated, like the modeling aspects and the insertion of contextual information within alert correlation engines.

Critics

The framework we propose has also some limitations, or, said differently, a few points that must be considered or/and improved in the future. The proposed techniques have been chosen because of their relevance and their simplicity, and they have helped building an interesting framework. These applied techniques have proved that they were applicable in our context as they have provided interesting results. Other techniques however might be applied as well. This work is at the intersection of various research domains. We have tried to build something coherent. This work does not mean that no other solution exists or cannot be applied. *A contrario*, this framework would really benefit from the work of specialists improving each step of this technique. Some other potential applications should be worth being tested.

The experiments have also clearly shown that the current interaction of the honeypot sensors is limited. The technique would really benefit from additional traffic to improve the discrimination phase and collect more exploit information. The new Scryptgen technique described in [138] will solve this problem and should be deployed in the *Leurré.com* sensors soon. It is also important to keep comparing regularly the activities on different interaction honeypots, in order to check that the honeypot sensor itself does not introduce particular bias in the collect of data; or it is important, at least, to qualify and quantify this bias. We have presented a mechanism to deal with this problem in [187].

The *framework* does not include, by definition, too many detection mechanisms, as it is not its initial purpose. However, it seems relevant, at this stage, to incorporate several mechanisms which would detect changes, either directly related to *attack fingerprints*, or other more global trends. They will be very useful, among other things, for reporting

the activities and for warning against recent abnormal activities. These mechanisms can be inserted at different steps of the framework. We have mentioned some of them along the thesis, but other techniques, derived from the Intrusion Detection System (IDS) field, could also be easily integrated. In addition, the framework has not been optimized with the prospect of being an efficient early-warning system. The tuning of some steps to shorten the warning delays might also be possible. An additional advantage of detecting changes can also be to potentially adapt the discrimination step and to create a new correlative analysis between the observed activities.

Conclusion

Echoing the Introduction, there were two major questions we wanted to address in this thesis. First, we wondered if the dataset at our disposal, which represents malicious activities collected by various sensors in the world, contains useful and original information. All examples cited in this thesis bring a clear affirmative answer to that question: data collected locally for a long time period enables to better understand the activities that occur in the Internet and are definitely worth being considered. Secondly, we were looking for a possible framework called *HoRaSis* that would automatize the adequate analysis of the data. If it exists, it should at least follow a few properties that have been detailed in the introduction.

Along this thesis, we have presented an analysis technique which has both confirmed the preliminary findings we made and has permitted to acquire a new and original knowledge out of the huge amount of collected data from the *Leurré.com* project. This analysis technique is open to other approaches, due to its interesting approach of classifying data. Its modular aspects ease the evolution of the mechanism and the plug-in of additional analysis layers. Finally, the methods we have proposed to classify data and extract information remain intuitive enough for the analyst to understand the outcome of the technique. In other words, the proposed technique is not an obscure or magical black box, and the analyst should now be able to understand all the steps that have led us to particular observations.

As a conclusion, we have correctly presented over this thesis a *HoRaSis* framework, with respect to the properties imposed in the Introduction. The proposed framework is the bases for *Honeypot tRaffic analySis*. It has been implemented and applied on the data collected from the *Leurré.com* distributed network of honeypot sensors.

The *HoRaSis* framework we have defined is a key element in our argumentation in favor of a better knowledge acquisition of malware activities. However, we think that the proposed framework presents more value by the questions it arises than by its implementation itself. Instead of being an end in itself, this framework is the illustration of positions defended in this document. We thus hope it will provide food for thoughts for future work. We conclude by giving a hint of potential research directions which seem

promising.

Perspectives

As previously mentioned, we believe that the framework we have proposed opens new interesting perspectives, and also many questions.

- *Global vs. Local Monitoring:* it is clear from this presented work that both approaches are complementary. A complete analysis of attack processes clearly requires the two positioning. We have not addressed, in this thesis, the problem of making them interact and exchange information. They both provide different abstraction levels and it would be interesting to merge the two approaches within a general monitoring system, able to interpret both types of abstraction, and thus, understand their respective limitations.
- *Dynamic Configuration of Sensors:* To determine the fingerprints of activities, it has been assumed that all sensors share the very same configuration. However, it would be worth diversifying the configurations, with different types of services and operating systems. in other words, it would be interesting to copy the diversity of real world systems into the network of sensors. Unfortunately, a few issues must then be correctly addressed. First, this approach might require quite numerous sensors deployed over the Internet. Second, the cross-correlation between activity fingerprints and configurations must be carefully understood and formalized. The database architecture used within the *Leurré.com* project has however been designed with this perspective in mind.
- *Context Provisioning:* To date, each activity is reported as a *card*, including its fingerprint parameters and the labels characterizing particular correlation with other activities. From another point of view, vulnerabilities and exploits are frequently published and many incidents are also reported. They all form an additional information context that might help understanding the monitored activities. It would thus be interesting to associate both and express their potential relationships.
- *Sensor Positioning:* The sensors have all been plugged in a large variety of places, in front of partners' networks. It would be interesting to determine if a very same framework can be applied with sensors inside a private network. This, however, presents a few privacy issues, and most of the partners would be reluctant to share such information. On the other hand, the discrimination phase of the framework might help the administrator in her analysis. A direct application would then be the insertion of the resulting information into the correlation engines she uses.

It is not an ordinary fact that we end a thesis with so many research directions. We have wanted to highlight the large exploratory fields which have appeared when consider-

ing the requirement of a better understanding of malware activities. Such an understanding is necessary and possible to acquire. We hope that it is now demonstrated with the proposed *HoRaSis* framework.

Bibliography

[1] "The CAIDA Project: FlowScan, Network Traffic Flow Visualization and Reporting Tool", Internet: http://www.caida.org/tools/utilities/flowscan/.

[2] "CJB pages: The Grim's Ping homepage", Internet: http://grimsping.cjb.net/.

[3] "CNET Security Center: reviews: Zotob prevention and cure", Internet: http://reviews.cnet.com/4520-6600_7-6299565.html.

[4] "Disco Passive Fingerprinting Tool", Internet: http://www.altmode.com/disco.

[5] "DShield Distributed Intrusion Detection System", Internet: http://www.dshield.org.

[6] "Ethereal Software: Tethereal, a Network Protocol Analyzer", Internet: http://www.ethereal.com/docs/man-pages/tethereal.1.html.

[7] "Ettercap: Passive scanning of the LAN", Internet: http://ettercap.sourceforge.net.

[8] "GIAC Practical Reports: SQLSnake presentation", Internet: http://www.giac.org/practical/Christopher_Short_GCIH.doc.

[9] "Honeyd Virtual Honeypot from N. Provos", Internet: http://www.honeyd.org.

[10] "Levenshtein association: Efficient Implementation of the Levenshtein Algorithm, Fault-Tolerant Search Technology", Internet: http://www.levenshtein.net.

[11] "MaxMind GeoIP Country Database - Commercial Product", Internet: http://www.maxmind.com/app/products.

[12] "MIT LCS's Parallel and Distributed Operating Systems Group: The Click Modular Router Project", Internet: http://pdos.csail.mit.edu/click/.

[13] "QoSient Inc.: Argus Open Project, AUditing Network Activity", Internet: http://www.qosient.com/argus/index.htm.

[14] "The SANS Institute - Internet Storm Center: The Trusted Source for Computer Security Trainind, Certification and Research", Internet: http://isc.sans.org.

[15] "The SCAMPI Eureopean Project: A Scalable Monitoring Platform for the Internet Contract No IST-2001-32404: D0.1 Description and analysis of the state-of-the-art", Internet: www.ist-scampi.org/publications/deliverables/.

[16] "Symante Service Support: Ports used for communication in Norton Antivirus Corporate Edition", Internet: http://service1.symantec.com/SUPPORT/ent-security.nsf/docid/20000101210181048.

[17] "TCPDump utility", Internet: http://www.tcpdump.org.

[18] "Towards a taxonomy of intrusion-detection systems", *Comput. Networks*, 31(9):805–822, 1999.

[19] "LURHQ and Corporation: Critical Microsoft Messenger Patch Released", October 2003.

[20] "LURHQ and Corporation: Windows Messenger Popup Spam on UDP Port 1026", June 2003.

[21] "Symantec Security Response - W32.Blaster.Worm", Internet: http://securityresponse1.symantec.com/sarc/sarc-intl.nsf/html/fr-w32.blaster.worm.html, August 2003.

[22] "The CAIDA Project: The Spread of the Witty Worm", Internet: www.caida.org/analysis/security/witty/, 2004.

[23] "honeyd Homepage", Internet: http://wwww.honeyd.org/, 2004.

[24] "p0f: Passive OS Fingerprinting Tool", Internet: http://lcamtuf.coredump.cx/p0f.shtml, 2004.

[25] "Specter Homepage", Internet: http://www.specter.com/, 2004.

[26] "Symantec Security Response - W32.Sasser.Worm", Internet: http://securityresponse.symantec.com/ avcenter/venc/data/w32.sasser.worm.html, 2004.

[27] "University of Michigan Internet Motion Sensor (IMS)", Internet: http://ims.eecs.umich.edu/, 2004.

[28] "The AdvanceSCAN advscan utility", Internet: http://advancemame.sourceforge.net/doc-advscan.html, 2005.

[29] "Hewlett Packard: Managing HP servers through firewalls with HP Systems Insight Manager", Internet: http://h200001.www2.hp.com/bc/docs/support/SupportManual, 2005.

[30] "Insecure post: Idle Scanning and Related IPID Games", Internet: http://www.insecure.org/nmap/idlescan.html, 2005.

[31] "MWCollect Trac, The Honeynet Project", Internet: http://www.mwcollect.org/, 2005.

[32] "Nepenthes official Website", Internet: http://nepenthes.sourceforge.net/, 2005.

[33] "SANS - Internet Storm Center - Cooperative Cyber Threat Monitor And Alert System", Internet: http://isc.sans.org/, 2005.

[34] "The Computer Network Defence Operational Picture", Internet: http://securitywizardry.com/radar.htm, 2005.

[35] "The Honeynet Scan of the Month Challenges", Internet: http://www.honeynet.org/misc/chall.html, 2005.

[36] "The WormRadar Project", Internet: http://www.wormradar.com, 2005.

[37] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pp. 487–499, Morgan Kaufmann, 12–15 1994.

[38] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control", *RFC 2581*, April 1999.

[39] M. Allman, W. M. Eddy, and S. Ostermann, "Estimating loss rates with TCP", *SIGMETRICS Perform. Eval. Rev.*, 31(3):12–24, 2003.

[40] C. Associates, "Computer Associates, Virus Information Center: Win32.Rbot.H Method of Distribution", Internet: http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=39662, 2004.

[41] J. G. Augustson and J. Minker, "An Analysis of Some Graph Theoretical Cluster Techniques", *J. ACM*, 17(4):571–588, 1970.

[42] M. Bailey, E. Cooke, F. Jahanian, D. Watson, and J. Nazario, "The Blaster Worm: Then and Now", In *IEEE Journal on Security and Privacy*, pp. 26–31, Washington, DC, USA, 2005, IEEE Computer Society.

[43] R. Bartels, J. Beatty, and B. Barsky, "Hermite and Cubic Spline Interpolation", Ch. 3 in An Introduction to Splines for Use in Computer Graphics and Geometric Modelling, Morgan Kaufmann, pp. 9-17, 1998.

[44] R. Baumann and C. Plattner, "White Paper: Honeypots", Internet: "citeseer.ist.psu.edu/baumann02white.html.

[45] R. A. Becker, S. G. Eick, and A. R. Wilks, "Visualizing Network Data", *IEEE Transactions on Visualization and Computer Graphics*, 1(1):16–28, 1995.

[46] R. Bejtlich, "Network Security Monitoring with Sguil", In *The Technical BSD Conference BSDCan2004*, Ottawa, Canada, May 2004.

[47] R. Bejtlich, *The Tao of Network Security Monitoring, Beyond Intrusion Detection*, Addison-Wesley, 2004.

[48] J. Bellardo and S. Savage, "Measuring Packet Reordering", In *Proc. of the Internet Measurement Workshop IMW 2002*, 2002.

[49] S. Bellovin, "A Technique for Counting NATed Hosts", Marseille, France, November 2002, In Proc.of the 2nd Internet Measurement Workshop 2002 (IMW'02).

[50] S. M. Bellovin, "There Be Dragons", In *Proc. of the Third Usenix UNIX Security Symposium*, 1992.

[51] V. H. Berk, R. S. Gray, and G. Bakos, "Using Sensor Networks and Data Fusion for Early Detection of Active Worms", In *Proc. of AeroSense 2003: SPIE's 17th Annual International Symposium on Aerospace/Defense Sensing, Simulation, and Controls*, Orlando, Florida, April 2003.

[52] P. Bertrand and M. F. Janowitz, "Pyramids and weak hierarchies in the ordinal model for clustering", *Discrete Appl. Math.*, 122(1-3):55–81, 2002.

[53] P. Bertrand, "Set systems and dissimilarities", *Eur. J. Comb.*, 21(6):727–743, 2000.

[54] J. Bethencourt, J. Franklin, and M. Vernon, "Mapping Internet Sensors With Probe Response Attacks", In *Proc. of the 14th Usenix Security Symposium (USENIX'05)*, Anaheim, CA, USA, April 2005.

[55] H. Bos and K. Huang, "Towards software-based signature detection for intrusion prevention on the network card", In *Proc. of the Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, WA, September 2005.

[56] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph", *Commun. ACM*, 16(9):575–577, 1973.

[57] E. Brosh, G. Sharon, and Y. Shavitt, "Spatial-Temporal Analysis of Passive TCP Measurements", In *Proc. of the 24th Annual IEEE Infocom Conference 2005*, Miami, USA, March 2005.

[58] M. Brown, W. Grundy, D. Lin, N. Christianini, C. Sugnet, M. Jr, and D. Haussler, "Support vector machine classification of microarray gene expression data", Internet: citeseer.ist.psu.edu/brown99support.html, 1999.

[59] N. Brownlee and M. Murray, "Streams, flows and torrents", In *Proc. of the Passive and Active Measurement PAM Workshop*, 2001.

[60] J. Buhmann, "Large Scale Monitoring of Broadband Internet Infrastructure: lobster Home Page", Internet: http://www.ist-lobster.org.

[61] J. Buhmann, "Learning and data clustering", Internet: citeseer.ist.psu.edu/buhmann95learning.html, 1995.

[62] H. Burch, "Measuring an IP Network in situ", 2005, PhD report, School of Computer Science, Carnegie Mellon University, Pittsburgh.

[63] CAIDA, "The CAIDA Project: Netgeo Utility - The Internet Geographical Database", Internet: http://www.caida.org/tools/utilities/netgeo/.

[64] M. Cauzomb, "symsvc.exe: a forum post", Internet: http://www.iamnotageek.com/history/topic.php/77580-1.html.

[65] S. Chainay, "Leurrécom, Répartition de pots de miel: Outil d'analyse pour la caractérisation des attaques informatiques", M.S. Thesis, ENST Paris, 2005.

[66] P. Chambet and T. F. H. Project, "FakeNetBIOS tools for Honeyd", Internet: http://honeynet.rstack.org/tools.php, 2005.

[67] P. T. Chen, C. S. Laih, F. Pouget, and M. Dacier, "Comparative survey of local honeypot sensors to assist network forensics", In *Proc. of SADFE'05, 1rst International Workshop on Sytematic Approaches to Digital Forensic Engineering, November 7-9, 2005, Taipei, Taiwan*, Nov 2005.

[68] S. Chen and Y. Tang, "Slowing Down Internet Worms", In *Proc. of the 24th International Conference on Distributed Computing Systems (ICDCS)*, pp. 312–319, IEEE Computer Society, March 2004.

[69] Z. Chen, L. Gao, and K. Kwiat, "Modeling the Spread of Active Worms", In *IEEE INFOCOM*, 2003.

[70] C.-H. Cheng, "A new approach for ranking fuzzy numbers by distance method", *Fuzzy Sets Syst.*, 95(3):307–317, 1998.

[71] Y. Cheng, U. Hoelzle, N. Cardwell, S. Savage, and G. Voelker, "Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying", In *Proc. of USENIX Technical Conference*, June 2004.

[72] W. Cheswick, "An evening with Berferd", pp. 103–116, 1998.

[73] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs", In *Proc. of KDD '03, the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 493–498, New York, NY, USA, 2003, ACM Press.

[74] A. Chuvakin, "Days of the Honeynet: Attacks, Tools, Incidents", Internet: http://www.linuxsecurity.com/feature_stories/feature_story141.html, 2003.

[75] Cisco, "Cisco Systems Inc.: NetFlow Services and Applications, White Paper", Internet: cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/nappswp.htm.

[76] K. C. Claffy, "CAIDA: Visualizing the Internet", *IEEE Internet Computing*, 5(1):88, 2001.

[77] K. Claffy, "Correlating Heterogeneous Measurement Data to Achieve System-Level Analysis of Internet Traffic Trends", Internet: http://www.caida.org/projects/trends/, February 2004.

[78] K. C. Claffy, H.-W. Braun, and G. C. Polyzos, "A Parameterizable Methodology for Internet Traffic Flow Profiling", *IEEE Journal of Selected Areas in Communications*, 13(8):1481–1494, 1995.

[79] J. B. Colombe and G. Stephens, "Statistical profiling and visualization for detection of malicious insider attacks on computer networks", In *Proc. of VizSEC/DMSEC '04, the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 138–142, New York, NY, USA, 2004, ACM Press.

[80] H. M. Company, "The American Heritage Dictionary of the English Language", Internet: http://www.thefreedictionary.com, 2003.

[81] E. Cooke, M. Bailey, Z. Mao, D. Watson, F. Jahanian, and D. McPherson, "Toward Understanding Distributed Blackhole Placement", In *Proc. of the Recent Advances of Intrusion Detection RAID'04*, September 2004.

[82] E. Cooke, M. Bailey, Z. M. Mao, D. Watson, F. Jahanian, and D. McPherson, "Toward understanding distributed blackhole placement", In *Proc. of WORM '04, the 2004 ACM workshop on Rapid malcode*, pp. 54–64, New York, NY, USA, 2004, ACM Press.

[83] J. Coppens, S. V. den Berghe, H. Bos, E. P. Markatos, F. D. Turck, A. Oslebo, and S. Ubik, "SCAMPI: A Scalable and Programmable Architecture for Monitoring Gigabit Networks.", In *MMNS*, pp. 475–487, 2003.

[84] J. R. Crandall and F. T. Chong, "Minos: Control Data Attack Prevention Orthogonal to Memory Model", In *Proc. of MICRO 37, the 37th annual International Symposium on Microarchitecture*, pp. 221–232, Washington, DC, USA, 2004, IEEE Computer Society.

[85] J. R. Crandall, S. F. Wu, and F. T. Chong, "Experiences Using Minos as a Tool for Capturing and Analyzing Novel Worms for Unknown Vulnerabilities.", In *DIMVA*, pp. 32–50, 2005.

[86] M. Cristea, W. de Bruijn, and H. Bos, "FPL-3: towards language support for distributed packet processing", In *Proc. of IFIP Networking'05*, Waterloo, Ontario, Canada, May 2005.

[87] T. Cymru, "Team Cymru: The Darknet Project", Internet: http://www.cymru.com/Darknet, 2004.

[88] M. Dacier, F. Pouget, and H. Debar, "Attack Processes found on the Internet", In *Proc. of the NATO Symposium IST-041/RSY-013*, April 2004.

[89] M. Dacier, F. Pouget, and H. Debar, "Honeypots, a Practical Mean to Validate Malicious Fault Assumptions", In *Proc. of the 10th Pacific Ream Dependable Computing Conference (PRDC04)*, February 2004.

[90] Darpa, "RFC 791: Internet Procol, Darpa Internet Program, Protocol Specification", Internet: http://www.faqs.org/rfcs/rfc791.html, September 1981.

[91] F. Dressler, C. Sommer, and G. Muenz, "IPFIX Aggregation", Internet-Draft, IETF, jul 2005.

[92] S. Dudoit and R. Gentleman, "Cluster Analysis in DNA Microarray Experiments", Internet: bioconductor.org/workshops/2002/Seattle02/Cluster/cluster.pdf, 2002.

[93] T. Duebendorfer and B. Plattner, "Observations of the Blaster and Sobig Worm Outbreaks in an Internet Backbone", In *the IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment DIMVA 2005*, July 2005.

[94] M. Eisen and M. de Hoon, "Cluster 3.0 Manual, Stanford University", Internet: http://bonsai.ims.u-tokyo.ac.jp, 2002.

[95] T. Eiter and H. Mannila, "Distance Measures for Point Sets and their Computation", *Acta Informatica*, 34(2):109–133, 1997.

[96] D. Ellis, "Worm anatomy and model", In *Proc. of WORM '03, the 2003 ACM workshop on Rapid Malcode*, pp. 42–50, New York, NY, USA, 2003, ACM Press.

[97] N. Feamster, J. Jung, and H. Balakrishnan, "An Empirical Study of "Bogon" Route Advertisements", In *Compute Communication Review, Volumn 35, Number 1*, „ January 2005.

[98] A. Feldmann, J. Rexford, and R. Cáceres, "Efficient policies for carrying Web traffic over flow-switched networks", *IEEE/ACM Trans. Netw.*, 6(6):673–685, 1998.

[99] E. Filiol, "Applied Cryptanalysis of Cryptosystems and Computer Attacks Through Hidden Ciphertexts Computer Viruses", ISSN 0249-6399, INRIA Rennes, IRISA, January 2002.

[100] E. Filiol, *Computer Viruses: from theory to applications*, Springer Paris, 1. Ed. edition, 2005.

[101] D. Fisher, L. Xu, and N. Zard, "Ordering effects in clustering", In *Proc. of ML92, the ninth international workshop on Machine learning*, pp. 163–168, San Francisco, CA, USA, 1992, Morgan Kaufmann Publishers Inc.

[102] S. Floyd and V. Paxson, "Difficulties in simulating the internet", *IEEE/ACM Trans. Netw.*, 9(4):392–403, 2001.

[103] Force5web, "Sfind Scanner homepage", Internet: http://www.force5web.com/articles/sql_scan.htm.

[104] V. R. Garza, "Security Researcher causes furor by releasing flaw in Cisco Systems IOS", July 2005, from SearchSecurity.com News.

[105] C. Gates, M. Collins, M. Duggan, A. Kompanek, and M. Thomas, "More Netflow Tools for Performance and Security.", In *LISA*, pp. 121–132, 2004.

[106] A. Graycar and R. Smith, "Identifying and Responding to Electronic Fraud Risks", In *Proc. of the 30th Australian Registrars's Conference, Australian Institute of Technology*, November 2002.

[107] S. Grundschober and M. Dacier, "Design and Implementation of a Sniffer Detector", In *Proc. of Recent Advances in Intrusion Detection Workshop RAID98*, 1998.

[108] L. Hamers, Y. Hemeryck, G. Herweyers, M. Janssen, H. Keters, R. Rousseau, and A. Vanhoutte, "Similarity measures in scientometric research. The Jaccard index versus Saltons cosine formula", *Information processing and management*, 25(3):315–333, 1989.

[109] L. T. Herberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor.", In *IEEE Symposium on Security and Privacy*, pp. 296–305, 1990.

[110] K. K. Hirji, "Discovering Data Mining: From Concept to Implementation (Book Review).", *SIGKDD Explorations*, 1(1):44–45, 1999.

[111] T. Holz, "The Honeynet Project and Research Alliance: Know Your Enemy: Tracking Botnets", Internet: http://www.honeynet.org/papers/bots/, March 2005.

[112] Homelinux, "RTSP Scanner", Internet: http://iperl.homelinux.org/haxor/scanner.pl.

[113] H. Hubbard, "Scob Infection statistics, etc...", Internet: http://www.securityfocus.incidents, June 2004.

[114] W. Hunt, "Lecture note: The Stable Marriage Problem", Internet: http://www.csee.wvu.edu/%7Eksmani/courses/fa01/random/lecnotes/lecture5.pdf.

[115] F. A. Hussain, "Identification of Repeated Attacks Using Network Traffic", Internet: citeseer.ist.psu.edu/640480.html.

[116] IHUG, "Roadkil's FTP Probe home page", Internet: http://homepages.ihug.com.au/ roadkil/ftpprobe.htm.

[117] IP2Location.com, "Map IP Address to Geographical Location for the Internet: IP2Location", Internet: http://www.ip2location.com.

[118] Y. D. J. Riordan, D. Zamboni, "Lessons Learned from Billy Goat, an Accurate Worm-Detection System", Zurich IBM Research laboratory, Research Report.

[119] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Measurement and classification of out-of-sequence packets in a tier-1 IP backbone", In *Proc. of IMW '02, the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 113–114, New York, NY, USA, 2002, ACM Press.

[120] S. Jaiswal, G. Iannaccone, C. Diot, and D. F. Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements.", In *INFOCOM*, 2004.

[121] X. Jiang and D. Xu, "Collapsar: A VM-Based Architecture for Network Attack Detention Center.", In *USENIX Security Symposium*, pp. 15–28, 2004.

[122] K. Julisch, "Mining Alarm Clusters to Improve Alarm Handling Efficiency", In *Proc. of the 17th Annual Computer Security Applications Conference (ACSAC)*, pp. 12–21, December 2001.

[123] K. Julisch, "Clustering intrusion detection alarms to support root cause analysis", *ACM Trans. Inf. Syst. Secur.*, 6(4):443–471, 2003.

[124] J.-K. Kamarainen, V. Kyrki, J. Ilonen, and H. Kälviäinen, "Improving similarity measures of histograms using smoothing projections", *Pattern Recogn. Lett.*, 24(12):2009–2019, 2003.

[125] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra, "Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases", *Knowledge and Information Systems*, 3(3):263–286, 2001.

[126] S. A. Khayam and H. Radha, "Analyzing the Spread of Active Worms over VANET", In *Proc. of the ACM VANET 2004*, ACM, 2004.

[127] Y. Kim, W. cheong Lau, M. C. Chuah, and H. J. Chao, "PacketScore: Statistical-based overload control against Distributed Denial-of-Service Attacks.", In *INFOCOM*, 2004.

[128] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment", *Journal of the ACM*, 46(5):604–632, 1999.

[129] T. Kohno, A. Broido, and K. C. Claffy, "Remote Physical Device Fingerprinting", In *Proc. of SP '05, the 2005 IEEE Symposium on Security and Privacy*, pp. 211–225, Washington, DC, USA, 2005, IEEE Computer Society.

[130] O. Kolesnikov and W. Lee, "Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic", Internet: citeseer.ist.psu.edu/678163.html.

[131] R. Koradi, M. Billeter, M. Engeli, P. Guntert, and K. Wuthrich, "Automated peak Picking and Peak Integration in Macromolecular Nmr Spectra AUTOPSY", In *Journal of Magnetic Resonance 135, 288-297*, 1998, Article N. MN981570.

[132] K. Lakkaraju, W. Yurcik, and A. J. Lee, "NVisionIP: netflow visualizations of system state for security situational awareness", In *Proc. of VizSEC/DMSEC '04, the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 65–72, New York, NY, USA, 2004, ACM Press.

[133] L. Lamport, "Time, clocks, and the ordering of events in a distributed system", *Commun. ACM*, 21(7):558–565, 1978.

[134] E. Lange, C. Gr"opl, K. Reinert, O. Kohlbacher, and A. Hildebrandt, "High-Accuracy Peak Picking of Proteomics Data using Wavelet Techniques", In *Proc. of the Pacific Symposium on Biocomputing (PSB 2006)*, 2006, To appear.

[135] A. J. Lee, G. A. Koenig, X. Meng, and W. Yurcik, "Searching for Open Windows and Unlocked Doors: Port Scanning in Large-Scale Commodity Clusters", In *5th IEEE International Symposium on Cluster Computing and the Grid*, May 2005.

[136] L. Lee, "Measures of Distributional Similarity", In *37th Annual Meeting of the Association for Computational Linguistics*, pp. 25–32, 1999.

[137] S. Lee and C. Shields, "Tracing the Source of Network Attack: A Technical, Legal and Societal Problem", In *IEEE Workshop on Information Assurance and Security*, US Military Academy, West Point, NY, June 2001.

[138] C. Leita, K. Mermoud, and M. Dacier, "ScriptGen: an automated script generation tool for honeyd", In *Proc. of the 21st Annual Computer Security Applications Conference (ACSAC2005)*, December 2005.

[139] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray, "Simulating realistic network worm traffic for worm warning system design and testing", In *Proc. of the 2003 ACM Workshop on Rapid Malcode (WORM)*, pp. 24–33, ACM Press, 2003.

[140] M. Liljenstam, Y. Yuan, B. J. Premore, and D. M. Nicol, "A Mixed Abstraction Level Simulation Model of Large-Scale Internet Worm Infestations.", In *Proc. of the 10th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 109–116, IEEE Computer Society, 2002.

[141] D. Lin, "An Information-Theoretic Definition of Similarity", In *Proc. of ICML '98, the Fifteenth International Conference on Machine Learning*, pp. 296–304, San Francisco, CA, USA, 1998, Morgan Kaufmann Publishers Inc.

[142] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms", In *Proc. of DMKD '03, the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11, New York, NY, USA, 2003, ACM Press.

[143] S. Lin and N. McKeown, "A simulation study of IP switching", In *Proc. of SIGCOMM '97, the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 15–24, New York, NY, USA, 1997, ACM Press.

[144] P. Liu, W. Zang, and M. Yu, "Incentive-based modeling and inference of attacker intent, objectives, and strategies", *ACM Trans. Inf. Syst. Secur.*, 8(1):78–118, 2005.

[145] LURHQ, "Dabber Worm Analysis", Internet: http://www.lurhq.com/dabber.html, 2004.

[146] M. Mahoney and P. Chan, "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection", In *Proceeding of Recent Advances in Intrusion Detection (RAID)-2003*, volume 2820 of *Lecture Notes in Computer Science*, pp. 220–237, Springer Verlag, September 8-10 2003.

[147] G. R. Malan and F. Jahanian, "An extensible probe architecture for network protocol performance measurement", In *Proc. of SIGCOMM '98, the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 215–227, New York, NY, USA, 1998, ACM Press.

[148] MathWorld, "Stable Marriage Problem", Internet: http://mathworld.wolfram.com/StableMarriageProblem.html.

[149] B. McCarty, "Botnets: Big and Bigger", *IEEE Security & Privacy*, 1(4):87–90, 2003.

[150] J. McHugh, "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory", *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000.

[151] J. McHugh, "Sets, Bags, and Rock and Roll: Analyzing Large Data Sets of Network Data.", In *ESORICS*, pp. 407–422, 2004.

[152] D. McPherson, "Attack Fingerprint Sharing: The Need for Automation of Inter-Domain Information Sharing", Internet: http://www.ripe.net/ripe/meetings/ripe-50/presentations/ripe50-plenary-tue-attack-fingerprint.pdf, May 2005.

[153] J. McPherson, K.-L. Ma, P. Krystosk, T. Bartoletti, and M. Christensen, "PortVis: a tool for port-based detection of security events", In *Proc. of VizSEC/DMSEC '04, the 2004 ACM workshop on Visualization and data mining for computer security*, pp. 73–81, New York, NY, USA, 2004, ACM Press.

[154] P. Mockapetris and K. J. Dunlap, "Development of the domain name system", In *Proc. of SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pp. 123–133, New York, NY, USA, 1988, ACM Press.

[155] S. B. Moon, *Measurement and analysis of end-to-end delay and loss in the internet*, Ph.D. Thesis, 2000, Director-James F. Kurose and Director-Donald F. Towsley.

[156] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The Spread of the Sapphire/Slammer Worm", , CAIDA, ICSI, Silicon Defense, UC Berkeley EECS and UC San Diego CSE, 2003.

[157] D. Moore, C. Shannon, and J. Brown, "Code-Red: a case study on the spread and victims of an Internet worm", In *Proc. of Internet Measurement Workshop 2002*, Nov 2002.

[158] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Network Telescopes: Technical Report", Internet: http://www.caida.org.

[159] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm", *IEEE Security and Privacy*, 1(4):33–39, 2003.

[160] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage, "Inferring Internet Denial-of-Service activity", 2006, To appear in IEEE/ACM Transactions on Computer Science.

[161] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Network Telescopes", , CAIDA, 2003.

[162] D. Moore, G. Voelker, and S. Savage, "Inferring Internet denial of service activity", August 2001, Proc. of the Usenix Security Symposium.

[163] myNetWatchman, "Network Intrusion Detection and Reporting", Internet: http://www.mynetwatchman.com.

[164] P. Newman, G. Minshall, and T. L. Lyon, "IP switchingATM under IP", *IEEE/ACM Trans. Netw.*, 6(2):117–129, 1998.

[165] H. Q. Nguyen, "Rapport de stage de fin d'etudes: Programme d'alertes base sur des pots de miel", Master Thesis 2005, IFI and Institut Eurecom.

[166] T. Nguyen, W. de Bruijn, M. Cristea, and H. Bos, "Scalable network monitors for high-speed links:a bottom-up approach", In *Proc. of IEEE IPOM'04*, Beijing, China, October 2004.

[167] T. Oetiker and D. Rand, "MRTG, The Multi Router Trafic Grapher", Internet: http://people.ee.ethz.ch/ oetiker/webtools/mrtg/.

[168] G. Ollmann, "Stopping Automated Attack Tools: NGSSoftware Insight Security Research report", Internet: http://www.ngssoftware.com/papers/StoppingAutomatedAttackTools.pdf.

[169] V. Padmanabhan, L. Qiu, and H. Wang, "Server-based inference of Internet link lossiness", In *Proc. of IEEE INFOCOM'03, San Francisco, CA, USA*, April 2003.

[170] Y. Paker and T. Kindberg, "The worm program model: an application centred point of view for distributed architecture design", In *Proc. of EW 3, the 3rd workshop on ACM SIGOPS European workshop*, pp. 1–4, New York, NY, USA, 1988, ACM Press.

[171] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of internet background radiation", In *Proc. of IMC '04: the 4th ACM SIGCOMM conference on Internet measurement*, pp. 27–40, New York, NY, USA, 2004, ACM Press.

[172] M. Pavan and M. Pelillo, "A new graph-theoretic approach to clustering and segmentation", In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[173] V. Paxson and et al, "Known TCP Implementation Problems", *RFC 2525*, March 1999.

[174] V. Paxson, "End-to-End Internet Packet Dynamics", In *Proc. of the ACM SIGCOMM '97 conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, volume 27,4 of *Computer Communication Review*, pp. 139–154, Cannes, France, September 1997, ACM Press.

[175] V. Paxson, "Measuring adversaries", In *Proc. of SIGMETRICS 2004/PERFORMANCE 2004, the joint international conference on Measurement and modeling of computer systems*, pp. 142–142, New York, NY, USA, 2004, ACM Press.

[176] V. Paxson, "Strategies for sound internet measurement", In *Proc. of IMC '04, the 4th ACM SIGCOMM conference on Internet measurement*, pp. 263–271, New York, NY, USA, 2004, ACM Press.

[177] V. E. Paxson, *Measurements and analysis of end-to-end Internet dynamics*, Ph.D. Thesis, Berkeley, CA, USA, 1998.

[178] P. A. Pevzner and S.-H. Sze, "Combinatorial Approaches to Finding Subtle Signals in DNA Sequences", In *Proc. of the International Conference on Intelligent Systems for Molecular Biology*, pp. 269–278, AAAI Press, 2000.

[179] P. Phaal, S. Panchen, and N. McKee, "RFC 3176: InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", Internet: http://www.faqs.org/rfcs/rfc3176.html, September 2001.

[180] R. Poortinga, R. van de Meent, and A. Pras, "Analyzing campus traffic using the meter-MIB", Internet: citeseer.ist.psu.edu/poortinga02analyzing.html.

[181] M. Pop, S. L. Salzberg, and M. Shumway, "Genome Sequence Assembly: Algorithms and Issues", *Computer*, 35(7):47–54, 2002.

[182] J. Postel, "Transmission Control Protocol", *RFC 793*, September 1981.

[183] F. Pouget and M. Dacier, "Honeypot-based Forensics", In *Proc. of the AusCERT Asia Pacific Information Technology Security Conference 2004 (AusCERT2004)*, May 2004.

[184] F. Pouget, M. Dacier, and H. Debar, "HOneynets: Foundations For the Development of Early Warning Systems", In *Proc. of the Cyberspace Security and Defense: Research Issues*, 2005, Publisher Springler-Verlag, LNCS, NATO ARW Series.

[185] F. Pouget, M. Dacier, and V. Pham, "The Leurre.com Project home page", Internet: http://www.leurrecom.org, 2004.

[186] F. Pouget, M. Dacier, and V. Pham, "Leurre.com: On the advantages of Deploying a Large Scale Distributed Honeypot Platform", In *Proc. of the E-Crime and Computer Evidence Conference(ECCE 2005)*, March 2005.

[187] F. Pouget and T. Holz, "A Pointillist Approach for Comparing Honeypots", In *Proc. of the IEEE Conference on Detection of Intrusions and Malware & Vulnerability Assessment DIMVA 2005, Vienna, Austria*, July 2005.

[188] F. Pouget and M. Dacier, "Alert correlation", EURECOM+1291, Institut Eurecom, France, Dec 2003.

[189] F. Pouget and M. Dacier, "Alert correlation: Review of the state of the art", EURECOM+1271, Institut Eurecom, France, Dec 2003.

[190] F. Pouget and M. Dacier, "White paper: honeypot, honeynet: a comparative survey", EURECOM+1273, Institut Eurecom, France, Sep 2003.

[191] F. Pouget, M. Dacier, and H. Debar, "White paper: honeypot, honeynet, honeytoken: terminological issues", EURECOM+1275, Institut Eurecom, France, Sep 2003.

[192] F. Pouget, M. Dacier, and V. H. Pham, "Understanding threats: a prerequisite to enhance survivability of Computing Systems", In *Proc. of IISW'04, International Infrastructure Survivability Workshop 2004, in conjunction with the 25th IEEE International Real-Time Systems Symposium (RTSS 04) December 5-8, 2004 Lisbonne, Portugal*, Dec 2004.

[193] F. Pouget, G. Urvoy-Keller, and M. Dacier, "Impact of losses and reordering on malware analysis", EURECOM+1813, Institut Eurecom, France, Nov 2005.

[194] F. Pouget, G. Urvoy-Keller, and M. Dacier, "Time signatures to detect multi-headed stealthy attack tools", EURECOM+1814, Institut Eurecom, France, Nov 2005.

[195] N. Provos, "A Virtual Honeypot Framework", In *Proc. of the 13th USENIX Security Symposium*, pp. 1–14, 2004.

[196] T. H. Ptacek and T. N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", , 1998.

[197] X. Qin, D. Dagon, G. Gu, and a Lee, "Worm detection using local networks", Internet: citeseer.ist.psu.edu/qin04worm.html, 2004.

[198] J. R. Quinlan, *C4.5: programs for machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[199] V. V. Raghavan and M. Y. L. Ip, "Techniques for measuring the stability of clustering: a comparative study", In *Proc. of SIGIR '82, the 5th annual ACM conference on Research and development in information retrieval*, pp. 209–237, New York, NY, USA, 1982, Springer-Verlag New York, Inc.

[200] I. Rigoutsos and A. Floratos, "Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm [published erratum appears in Bioinformatics 1998;14(2): 229].", *Bioinformatics*, 14(1):55–67, 1998.

[201] E. Rosch, "Principles of Categorization", In E. Rosch and B. B. Lloyd, editors, *Cognition and Categorization*, pp. 27–48, Lawrence Erlbaum, Hillsdale, 1978.

[202] K. Sadasivam, B. Samudrala, and T. A. Yang, "Design of network security projects using honeypots", *J. Comput. Small Coll.*, 20(4):282–293, 2005.

[203] C. Schleippmann, "Design and Implementation of a TCP Rate Analysis Tool", M.S. Thesis, TU Muenchen/Eurecom, 2003.

[204] G. Serazzi and S. Zanero, "Computer virus propagation models", In M. C. Calzarossa and E. Gelenbe, editors, *Tutorials of the 11th IEEE/ACM Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecom. Systems - MASCOTS 2003*, 2003.

[205] S. Sidiroglou and A. Keromytis, "A Network Worm Vaccine Architecture", In *Proc. of the IEEE Workshop on Enterprise Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security*, 2003.

[206] J. M. Smith and D. C. P. Smith, "Database abstractions: aggregation", *Commun. ACM*, 20(6):405–413, 1977.

[207] R. Sommer and A. Feldmann, "NetFlow: information loss or win?", In *Proc. of IMW '02, the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 173–174, New York, NY, USA, 2002, ACM Press.

[208] SOPHOS, "Sophos Virus Analysis: W32/Agobot-PQ", Internet: http://www.sophos.com.au/virusinfo/analyses/w32agobotpq.html, 2004.

[209] SourceFire, "Snort, the de facto standard for intrusion detection and prevention", Internet: http://www.snort.org.

[210] J. F. Sowa, *Conceptual structures: information processing in mind and machine*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[211] E. Spafford, "An Analyis of the Internet Worm", In *Proc. European Software Engineering Conference*, pp. 446–468, Lecture Notes in Computer Science, Vol. LNCS 387, Springer-Verlag, September 1989.

[212] L. Spitzner, *Honeypots: Tracking Hackers*, Addison-Wesley, 2002.

[213] L. Spitzner, "Know Your Enemy: Honeynets", http://project.honeynet.org/papers/honeynet/, 2001.

[214] L. Spitzner, "The Honeynet Project: Trapping the Hackers", 1(2):15–23, March/ April 2003.

[215] S. Staniford, D. Moore, V. Paxson, and N. Weaver, "The Top Speed of Flash Worms", In *Proc. of the Recent Advances of Intrusion Detection RAID'04*, September 2004.

[216] S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time", In *Proc. of the 11th USENIX Security Symposium*, pp. 149–167, USENIX Association, 2002.

[217] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans", *J. Comput. Secur.*, 10(1-2):105–136, 2002.

[218] C. Stoll, *The Cuckoo's Egg: Tracking a Spy through the Maze of Computer Espionage*, Pocket Books, 2000.

[219] V. Stone, "W32 Deloder Worm: The Building of an Army", In *GCIH Practical Assignment, as part of GIAC Practical Respository*, 2003.

[220] Z. Sun, G. Bebis, and R. Miller, "Object Detection Using Feature Subset Selection", Internet: citeseer.ist.psu.edu/638662.html.

[221] Symantec, "Symantec Security Response W32.Spybot.FCD", Internet: http://securityresponse.symantec.com/avcenter/venc/data/w32.spybot.fcd.html, 2004.

[222] Symantec, "Symantec Security Response W32.Welchia.Worm", Internet: http://response.symantec.com/avcentr/venc/data/w32.welchia.b.worm.html, 2004.

[223] P. Szor, *The art of computer virus research and defense*, Addison-Wesley, 2005.

[224] K. Thompson, G. Miller, and R. Wilder, "Wide-Area internet traffic patterns and characteristics".

[225] A. Tversky, "Features of Similarity", *Psychological Review*, 84(4):327–352, 1977.

[226] urb23, "Metasploit Framework Tutorial", pp. 1–5, 2005.

[227] A. Wagner, T. Dübendorfer, B. Plattner, and R. Hiestand, "Experiences with worm propagation simulations", In *Proc. of WORM '03, the 2003 ACM workshop on Rapid Malcode*, pp. 34–41, New York, NY, USA, 2003, ACM Press.

[228] N. Weaver, "Potential Strategies for High Speed Active Worms: A Worst Case Analysis", http://www.cs.berkeley.edu/~nweaver/worms.pdf, Mar 2002.

[229] X. wen Chen, "Clustering Gene Expression Data With Min-Max-Median Initialized Fuzzy C-Means Algorithms", Internet: citeseer.ist.psu.edu/573111.html.

[230] M. C. Wendl and S.-P. Yang, "Gap statistics for whole genome shotgun DNA sequencing projects", *Bioinformatics*, 20(10):1527–1534, 2004.

[231] D. Widyantoro, T. Ioerger, and J. Yen, "An Incremental Approach to Building a Cluster Hierarchy", In *Proc. of the 2 nd IEEE International Conference on Data Mining, pp. 705-708.*, 2002.

[232] M. M. Williamson, "Throttling Viruses: Restricting propagation to defeat malicious mobile code", In *Proc. of ACSAC '02, the 18th Annual Computer Security Applications Conference*, p. 61, Washington, DC, USA, 2002, IEEE Computer Society.

[233] J. Wu, S. Vangala, L. Gao, and K. A. Kwiat, "An Effective Architecture and Algorithm for Detecting Worms with Various Scan.", In *NDSS*, 2004.

[234] Xoops, "WHAX3.1 started", Internet: http://www.iwhax.net/modules/news.

[235] V. Yegneswaran, P. Barford, and D. Plonka, "the design and use of internet sinks for network abuse monitoring", In *Proc. of the Recent Advances in Intrusion Detection 2004*, 2004.

[236] V. Yegneswaran, P. Barford, and S. Jha, "Global Intrusion Detection in the DOMINO Overlay System.", In *NDSS*, 2004.

[237] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha, "An Architecture for Generating Semantics-Aware Signatures", Internet: http://www.cs.wisc.edu/wisa/papers/security05/, 2005.

[238] B.-K. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary Lp Norms", In *Proc. of VLDB '00, the 26th International Conference on Very Large Data Bases*, pp. 385–394, San Francisco, CA, USA, 2000, Morgan Kaufmann Publishers Inc.

[239] Y. Yuan, "On the Design of an Immersive Environment for Security-Related Studies", TR2005-552, Dartmouth College, Computer Science, Hanover, NH, August 2005.

[240] M. Zalewski, *Silence on the Wire: A Field Guide to Passive Reconnaissance and Indirect Attacks (Paperback)*, No Starch Press, 2005.

[241] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker, "On the characteristics and origins of internet flow rates", In *Proc. of SIGCOMM '02, the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 309–322, New York, NY, USA, 2002, ACM Press.

[242] J. Zimmermann, A. Clark, G. Mohay, F. Pouget, and M. Dacier, "The use of packet inter-arrival times for investigating unsolicited Internet traffic", In *Proc. of SADFE'05, 1rst International Workshop on Sytematic Approaches to Digital Forensic Engineering, November 7-9, 2005, Taipei, Taiwan*, Nov 2005.

[243] C. Zou, W. Gong, and D. Towsley, "Code Red Worm Propagation Modeling and Analysis", In *ACM CCS 02*, November 2002.

[244] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for internet worms", In *Proc. of CCS '03, the 10th ACM conference on Computer and communications security*, pp. 190–199, New York, NY, USA, 2003, ACM Press.

[245] C. C. Zou, W. Gong, and D. Towsley, "Worm propagation modeling and analysis under dynamic quarantine defense", In *Proc. of WORM '03, the 2003 ACM workshop on Rapid Malcode*, pp. 51–60, New York, NY, USA, 2003, ACM Press.

[246] C. C. Zou, D. F. Towsley, and W. Gong, "Email Worms Modeling and Defense.", In *ICCCN*, pp. 409–414, 2004.

Appendix A

Entity Relationship Diagram

The following diagram has been described in [184]. It represents the database structure used to store the data from each *Leurré.com* honeypot sensor.

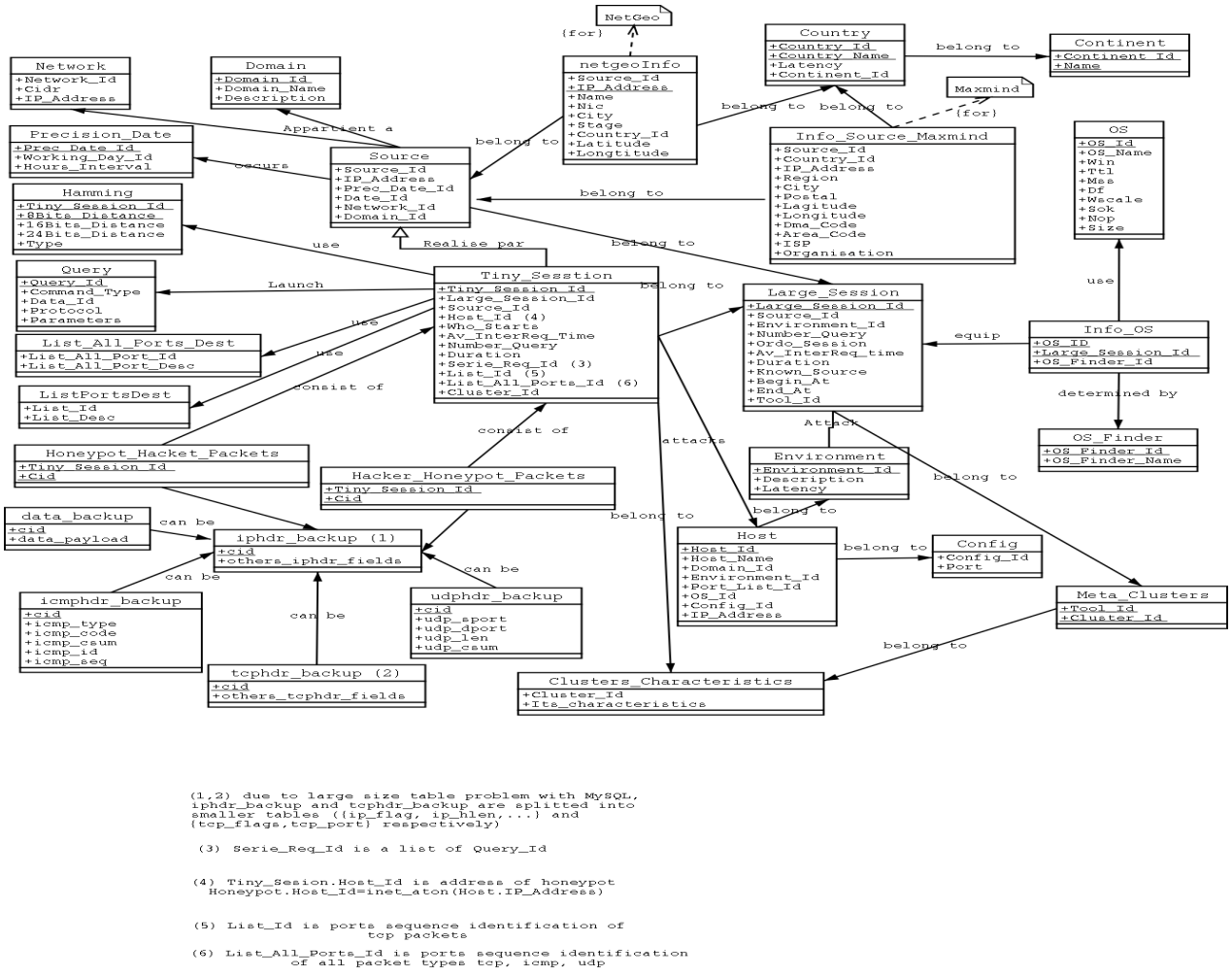


Figure A.1: Data Storage: Database Architecture

Appendix B

Leurré.com Interfaces

This Appendix epresents two different Screenshots. Figure B.1 is the global public project homepage, with global statistics on the dataset. Figure B.2 is a simple GUI that is accessible for partners to write queries on the database.

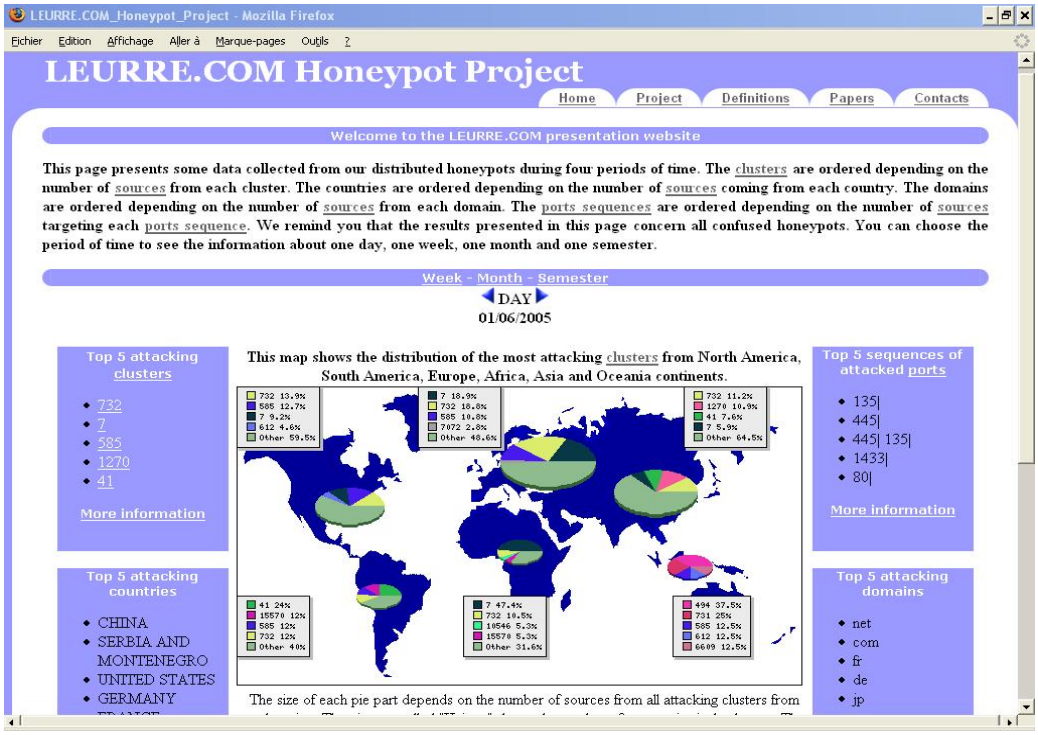


Figure B.1: Public Interface *www.leurrecom.org*

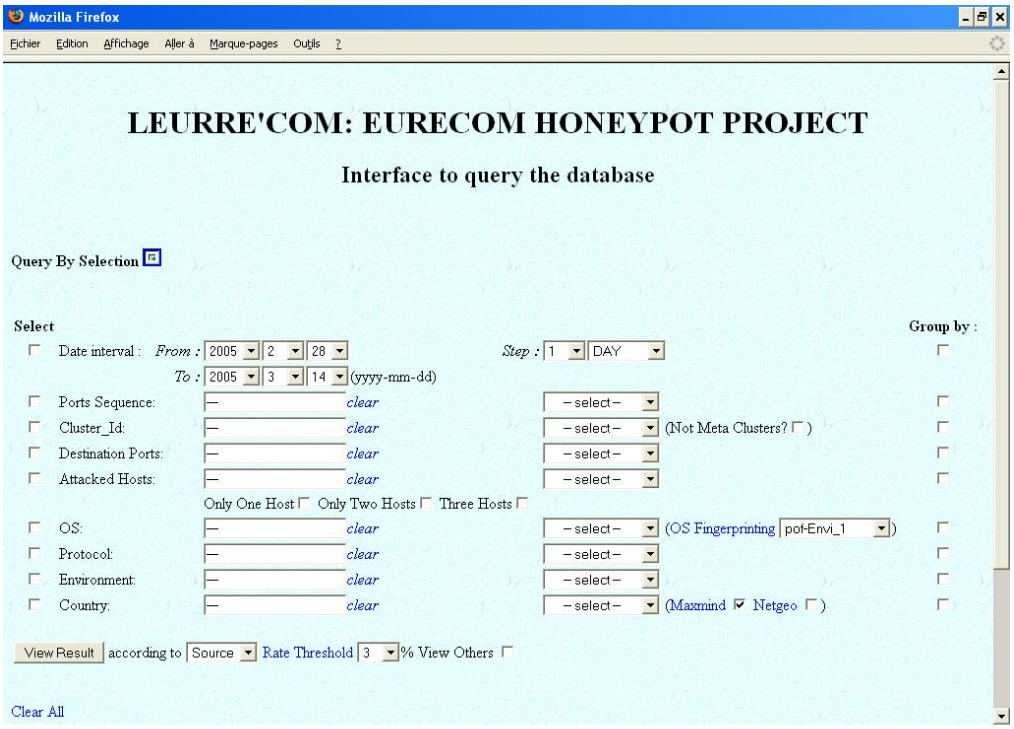


Figure B.2: Partner DB Interface: GUI

Appendix C

Reporting Activities on the *Leurré.com Project*

Figure C.1 is a screenshot of the main reporting page each partner can access and where she can get information on her specific *Leurré.com* honeypot sensor.

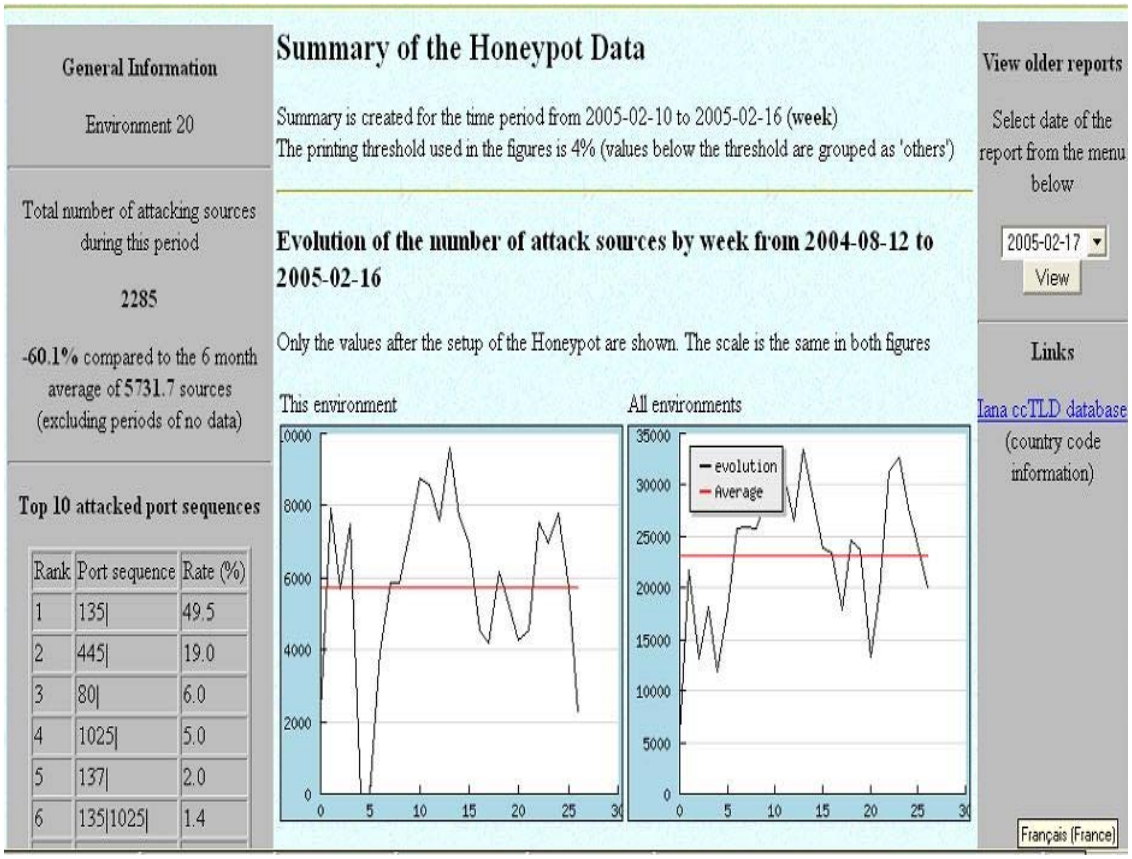


Figure C.1: Partner Reports from the *Leurré.com* Interface

Appendix D

Identification of Deloder among the *Activity Fingerprints*

This appendix is related to the description of the Deloder worm presented in Section 4.4.4.

The Deloder worm spreads by scanning random IPs, and attempts to connect to Windows 2000 or XP shares, which is TCP port 445 (SMB over TCP).

Barford et al. presents in [237] an interesting signature of the worm that can be matched with the *activity fingerprints* we have. Figure D.1 has been extracted from one of the signatures presented in [237].

The clusters derived from Deloder are then easily identified. They have the following parameter values:

- Number Virtual Machines = 1
- Targeted Ports Sequence= $\{445, 139\}$
- Number Received Packets = $[20, 23]$
- Duration $< 10s$
- Date First Observation = $March2003$
- Payloads Netbios as detailed in [237][1]

[1] It is also interesting to note that Snort has no particular rule for Deloder. The worm generates Snort alerts for IPC share access only.

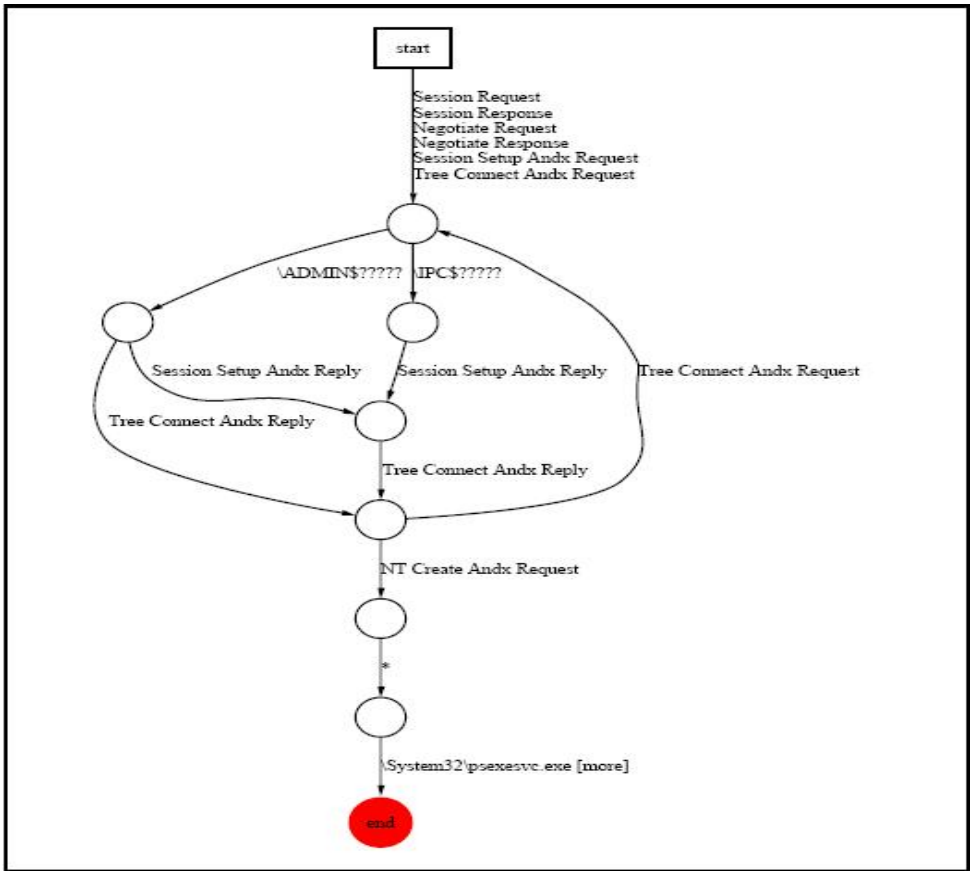


Figure D.1: Deloder Signature [237]