



Evaluation des systèmes de détection d'intrusion

Mohammed Gad El Rab

► To cite this version:

Mohammed Gad El Rab. Evaluation des systèmes de détection d'intrusion. Networking and Internet Architecture [cs.NI]. Université Paul Sabatier - Toulouse III, 2008. English. NNT: . tel-00366690

HAL Id: tel-00366690

<https://theses.hal.science/tel-00366690>

Submitted on 9 Mar 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE TOULOUSE III – PAUL SABATIER

T H E S E

en vue de l'obtention du

DOCTORAT DE L'UNIVERSITE DE TOULOUSE

délivré par l'Université Toulouse III – Paul Sabatier

Discipline : Systèmes Informatiques Critiques

présentée et soutenue

par

Mohammed El-Sayed GADELRAH

le 15/12/2008

Titre :

Évaluation des Systèmes de Détection d'Intrusion

Evaluation of Intrusion Detection Systems

Directeur de thèse : M. Yves Deswarte

Co-encadrant : M. Anas Abou El Kalam

JURY

M. Salem Benferhat	Président
M. Hervé Debar	Rapporteur
M. Ludovic Mé	Rapporteur
M. Abdelmalek Benzekri	Examineur
M. Yves Deswarte	Examineur
M. Anas Abou El Kalam	Examineur

To my mother and all those who supported me.

بسم الله الرحمن الرحيم

« فاما الزبد فيذهب جفاء وأما ما ينفع الناس فيمكث في الأرض »، قرآن كريم

“The scum vanish, but what benefits people remains”, Quran

“Sometimes it is useful to know how large your zero is”, Unknown Author

"Tout finit toujours bien. Si les choses ne marchent pas convenablement, c'est que vous n'êtes pas arrivé à la fin", Domingos Sabino

Remerciements

Les travaux présentés dans ce mémoire ont été effectués au *Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS)*. Je remercie Messieurs Malik Ghallab et Raja Chatila, qui ont assuré la direction du LAAS depuis mon entrée, de m'avoir accueilli dans ce laboratoire.

Je tiens à remercier particulièrement Mr. Jean Arlat, et Mme Karama Kanoun responsables successifs du groupe de recherche "Tolérance aux fautes et Sûreté de Fonctionnement informatique" (TSF) au sein duquel j'ai effectué mes travaux, de m'avoir accueilli et de m'avoir assuré des conditions idéales de travail. Je les remercie non seulement en tant que responsables du groupe mais aussi en tant que chercheurs pour leurs remarques très attentives.

J'exprime ma profonde gratitude aux personnes qui m'ont fait l'honneur de participer à mon jury de thèse :

- M. Salem Benferhat, Professeur à l'Université d'Artois,
- M. Hervé Debar, Ingénieur de recherche, Orange Labs, Caen,
- M. Ludovic Mé, Professeur à Supélec, Rennes,
- M. Abdelmalek Benzekri, Professeur à l'Université Paul Sabatier de Toulouse,
- M. Yves Deswarte, Directeur de Recherche CNRS au LAAS, Toulouse,
- M. Anas Abou El Kalam, Maître de Conférences à l'ENSEEIH, Toulouse.

Je remercie particulièrement Messieurs Hervé Debar et Ludovic Mé qui ont accepté la charge d'être rapporteurs.

Je ne sais pas si les mots seront suffisants pour remercier M. Yves Deswarte, mon cher directeur de thèse. Je le remercie d'avoir accepté de m'encadrer pour cette thèse. Je suis profondément reconnaissant de son support constant. Sans son aide, ce travail n'aurait pas été possible. Je suis impressionné par ses conseils de valeur, sa patience, ses remarques profondes. J'avoue que, quoique fastidieuse, sa méthode rigoureuse s'est avérée très efficace pour me rendre plus pointu dans ma recherche.

J'exprime également mes remerciements et ma très sincère reconnaissance à M. Anas Abou El Kalam, co-encadrant de ma thèse, qui m'a accompagné dès le début de la thèse. J'ai eu un grand plaisir à travailler avec lui et à maintenir un rythme et une efficacité élevés.

Tous mes sincères remerciements à l'ensemble des membres du groupe TSF : permanents, doctorants et stagiaires. Surtout, M. Mohammed Kaâniche, M. Vincent Nicomette, Mme Hélène Waeselynck, et M. Sébastien Gambs pour leur temps et leurs remarques très intéressantes. Je remercie particulièrement, M. Yves Crouzet (l'ange de TSF), cette personne si serviable, qui est toujours disponible pendant les moments très difficiles. Je remercie également M. Jacques Collet, dont j'ai partagé le bureau pendant mon séjour au LAAS. Finalement, je remercie Mme Gina Briand pour son aide au niveau administratif.

Je salue tous mes camarades doctorants du groupe TSF, notamment, Ossama Hamouda (Oss), Amine Baina (qui m'a accompagné de Bourges à Toulouse), Youssef Laarouchi (Yoyo), Manel Sghairi, Géraldine Vache, Etienne Baudin, Ludovic Courtes, les deux Thomas (Bob et Pareaud), Minh Duc Nguyen, Hoang Nam Chu, Eric Lacombe et Ismaïla Bradji, dont j'ai suivi en 2008 le stage de master.

Je remercie également les personnes de l'EDSYS (École Doctorale Systèmes), Mme Véronique Villemur, Mme Sophie Achte, M. Agnan Bonneval, Mme Caroline Berard, M. Robert Valette et M. Guy Juanole (les deux 1ères personnes du LAAS avec lesquelles j'ai pris contact).

Je salue tous mes amis égyptiens Ahmed Akl, Ahmed Ali, Mahmoud Mostafa, Ussama Zaghlool, Hesham Kotb, Hany Gamal et Tarek Moustafa. Leur présence autour de moi (surtout Oss Hamouda et A. Akl) m'a donnée beaucoup de motivation et d'envie de travailler.

Mes salutations vont aussi à tous mes amis ensibien, Issam Ladjal, Mostafa Hamdi, Hazem Wannous, Jawhar Ghomem, Mehdi Ammi et Jérémy Briffaut. J'ai vraiment profité de leur amitié et de nos discussions diverses et profondes.

Côté égyptien, je tiens à remercier les membres du Bureau Culturel de l'ambassade d'Égypte à Paris qui ont assuré le bon déroulement de la thèse et m'ont aidé à affronter les divers problèmes. Je remercie notamment Messieurs Taha Abdallah et Safwat Zahran, qui se sont succédés comme conseillers culturels depuis mon arrivé en France, ainsi que Mme Hanan El Sharkawy.

Je remercie tout les membres de l'Institut National de Standardisation de l'Égypte (NIS), administratifs et cadres, qui m'ont beaucoup facilité la tâche. En particulier, M. Ibrahim Motawei et M. Hesham Soultan pour leur encadrement scientifique.

Je dois également cette réussite à d'autres personnes car l'histoire n'a pas commencé seulement il y a trois ou quatre ans. Elle a commencé plutôt depuis mes premiers pas lorsque ma mère m'a accompagné à l'école où elle travaille et lorsqu'elle m'a enseigné l'alphabet. Les histoires encourageantes qu'elle nous a racontées de la part de mon père qui est décédé avant que je ne le voie. J'ai grandi jour après jour, ses histoires continuent avec moi et leur soutien a grandi aussi. Je tiens à les remercier infiniment.

Le reste de ma grande famille ont aussi participé à l'accomplissement de ce travail directement ou indirectement. Mes grands-pères Sheikh GADELRAH et Sheikh Mohammed EMAM sont porteurs des plus hauts diplômes (El Alameya) délivrés l'Université d'Al-Azhar, la plus vieille université du monde (fondée au dixième siècle). Le fait de voir mon oncle Mohammed comme un ingénieur brillant, de voir mon oncle Ahmed et ma tante Fatma comme des professeurs à la faculté de médecine, tout cela m'a donné courage. Ma sœur Amal (dont le nom signifie « l'espoir ») et mon frère Yasser, avec lesquels j'ai partagé une bonne partie de mes souvenirs, ont assuré tous les moyens pour que je fasse mes études jusqu'au bout. Donc, il n'est pas étonnant que j'ai pu être diplômé de l'université d'Al-Azhar, être ingénieur, et obtenir un doctorat pour être professeur (un jour).

Je ne pourrai pas remercier ma femme suffisamment. C'est l'héroïne invisible de l'histoire. Nous avons vécu ensemble cette thèse à chaque instant, les moments d'espoir comme les moments de déception. Elle a supporté tous mes défauts, qui se sont encore multipliés dans les jours qui ont précédé la soutenance.

Finalement, je voudrai dire que je suis arrivé là grâce à ALLAH d'abord et grâce à vous tous.

Encore merci.

Acknowledgment

This work was carried out at the Laboratory of Analysis and Architecture of Systems of the French National Center for Scientific Research (LAAS-CNRS). I wish to express my gratitude to the successive directors of LAAS: Dr. Malik Ghallab and Dr. Raja Chatila, for hosting me within their laboratory.

I would like to express my thanks twice to Mr. Jean Arlat and Mrs. Karama Kanoun. Once, as successive directors of the Dependable Computing and Fault Tolerance research group (TSF) for having allowed me to carry out this work and for providing all the required facilities. Second, I present my thanks to both of them as researchers for their valuable feedbacks on my research work.

I would like to thank all committee members, for having attentively read my dissertation:

- Mr. Salem Benferhat, Professor at the University of Artois, France,
- Mr. Hervé Debar, Research engineer, Orange Labs, Caen, France,
- Mr. Ludovic Mé, Professor at Supélec, Rennes, France,
- Mr. Abdelmalek Benzekri, Professor at the University Paul Sabatier of Toulouse, France,
- Mr. Yves Deswarte, CNRS Research Director at LAAS, Toulouse, France,
- Mr. Anas Abou El Kalam, Associate Professor at ENSEEIHT, Toulouse France.

My special thanks go to Mr. Hervé Debar and Mr. Ludovic Mé who accepted the burden of reporting on this thesis manuscript.

I do not know if words can be sufficient to thank, Mr. Yves Deswarte, my dear teacher and my thesis advisor. Many thanks to him for accepting to be my advisor. I am deeply grateful for his constant support. Without his help, this work would not have been possible. Through his valuable advices, I learned to be more professional in my research. I am impressed by his precious advices, his patience, and his deep comments. I admit that in spite of being fastidious, his rigorous method is very effective to make his students more professional researchers.

I gratefully thank Mr. Anas Abou El Kalam, co-advisor of my thesis, who followed my first steps from the beginning of the thesis. He is a strong combatant who never gives up against any technical or administrative problem. By working with him, you find yourself obligated to be as productive as him to keep with his very active working rhythm.

Thanks to all the members of the Dependable Computing and Fault Tolerance research group: permanents, PhD students and interns. Amongst them, I felt to be at home and I have much appreciated the family-like atmosphere. I am very pleased to mention to Mr. Mohammed Kaâniche, Mr. Vincent Nicomette, Ms. Hélène Waeselync, and Mr. Sébastien Gambis, for their time and for their very interesting remarks. Mr. Yves Crouzet, the angel of our group, deserves great thanks for his great services and for his availability even during the most difficult moments. My thanks go also to Mr. Jacques Collet, with whom I shared the office during my stay at LAAS and Ms. Gina Briand for her kindness and her help on administrative issues.

I send my greetings to all current, ancient PhD students and interns of our group: Ossama Hamouda (Oss), Amine Baina (my companion from Bourges to Toulouse), Youssef Laarouchi (Yoyo), Manel Sghairi, Géraldine Vache, Etienne Baudin, Ludovic Courtes, both Thomas (Bob et Pareaud), Minh Duc Nguyen, Hoang Nam Chu, Eric Lacombe and Ismaïla Bradji, whose master internship was mentored by me.

I wish also to thank the staff members of the postgraduate school EDSYS (École Doctorale Systèmes): Ms. Véronique Villemur, Ms. Sophie Achte, Mr. Agnan Bonneval, Ms. Caroline Berard as well as Mr. Robert Valette and Mr. Guy Juanole, (the first two persons of LAAS with whom I took contact).

My warm thanks go to my Egyptian friends: Ahmed Akl, Ahmed Ali, Heba Shaarawy, Mahmoud Mostafa, Ussama Zaghlol, Hesham Kotb, Hany Gamal and Tarek Mostafa. Their presence around me (especially Oss Hamouda and A. Akl) gave me a lot of enthusiasm and the desire to work more.

Also, I thank warmly my “ensibian” friends: Issam Ladjal, Mostafa Hamdi, Hazem Wannous, Jawhar Ghomem, Mehdi Ammi and Jérémy Briffaut. I really benefited from your friendship and our rich discussions.

From the Egyptian side, I wish to thank the team of the Egyptian Cultural and Educational Bureau in Paris for their tireless efforts and being there for me all the time. In particular, I have the pleasure to

mention hereafter: Mr. Taha Abdallah, Mr. Safwat Zhran, two successive Cultural counselors at the head of the Cultural Bureau as well as Mrs. Hanan El Sharkawy.

From the Egyptian National Institute of Standards, where I am working, I thank all the administrative and scientific teams. Their help had made my task easier. Special thanks to Mr. Ibrahim Motawei, Mr. Hesham Soultan from the IT laboratory for their scientific advice.

In fact, I owe the success also to other persons because the story had not begun 3 or 4 years ago. It had begun earlier since my first steps when my mother took me to the school where she had been working and when she taught me the alphabet. I remember the stories which she told us about our father who was dead before I could see him. These stories grew up with me and mother's support grew up too. My infinite thanks to my parents.

The rest of the family has also participated, directly or indirectly in the achievement of this thesis. My two grand fathers have "El Alameya" (the highest diploma) from Al-Azhar University, which is the most ancient university in the world (established in the tenth century). The fact that my uncle Mohamed was a brilliant engineer; my uncle Ahmed and my aunt Fatma are professors at the faculty of medicine; all gave me a lot of inspiration. I share many of souvenirs with my sister Amal (means hope) and my brother Yasser who did their best to ensure all the good conditions for me to complete my studies. Therefore, it is not surprising to be graduated from Al-Azhar University, to be an engineer and to obtain a PhD to become a professor (someday).

I cannot thank my wife enough as she is the invisible heroine of the history. We have lived together this thesis moment by moment: those of hope as well as the times of disappointments. She has tolerated all my defects that were multiplied in the last days before defending the thesis.

Finally, I would like to say that I have arrived here thanks to Allah first and thanks to you all.

Again thank you.

Résumé

Cette thèse vise à contribuer à l'amélioration des méthodes d'évaluation des systèmes de détection d'intrusions (en anglais, *Intrusion Detection Systems* ou IDS). Ce travail est motivé par deux problèmes actuels : tout d'abord, l'augmentation du nombre et de la complexité des attaques que l'on observe aujourd'hui nécessite de faire évoluer les IDS pour leur permettre de les détecter. Deuxièmement, les IDS actuels génèrent de trop fréquentes fausses alertes, ce qui les rend inefficaces, voir inutiles. Des moyens de test et d'évaluation sont donc nécessaires pour déterminer la qualité de détection des IDS et de leurs algorithmes de détection. Malheureusement, aucune méthode d'évaluation satisfaisante n'existe de nos jours. En effet, les méthodes employées jusqu'ici présentent trois défauts majeurs : 1) une absence de méthodologie rigoureuse d'évaluation, 2) l'utilisation de données de test non représentatives, et 3) l'utilisation de métriques incorrectes.

Partant de ce constat, nous proposons une démarche rigoureuse couvrant l'ensemble des étapes de l'évaluation des IDS. Premièrement, nous proposons une méthodologie d'évaluation qui permet d'organiser l'ensemble du processus d'évaluation. Deuxièmement, afin d'obtenir des données de test représentatives, nous avons défini une classification des types d'attaques en fonction des moyens de détection utilisés par les IDS. Cela permet non seulement de choisir les attaques à inclure dans les données de test, mais aussi d'analyser les résultats de l'évaluation selon les types d'attaques plutôt que pour chaque attaque individuellement. Troisièmement, nous avons analysé un grand nombre d'attaques réelles et de programmes malveillants (communément appelés maliciels) connus, tels que les virus et les vers. Grâce à cette analyse, nous avons pu construire un modèle générique de processus d'attaques qui met en évidence la dynamique des activités d'attaque. Ce modèle permet de générer un nombre important de scénarios d'attaques, qui soient le plus possible représentatifs et variés.

Pour montrer la faisabilité de notre approche, nous avons appliqué expérimentalement les étapes de notre démarche à deux systèmes différents de détection d'intrusions. Les résultats montrent que l'approche proposée permet de surmonter les deux défauts principaux des évaluations existantes, à savoir l'absence de méthodologie et l'utilisation de données non représentatives. En particulier, elle permet de mieux gérer le processus d'évaluation et de choisir les cas de test les plus adaptés à l'IDS sous test et les plus pertinents vis-à-vis des objectifs de l'évaluation en cours, tout en couvrant une large partie de l'espace d'attaques.

Ce manuscrit de thèse est divisé en deux parties rédigées respectivement en français et en anglais. Les deux parties suivent la même structure ; la première étant un résumé étendu de la deuxième.

Mots clés : Sécurité, Système de détection d'intrusions (IDS), Évaluation, Test, Attaque, Maliciel.

Abstract

This thesis contributes to the improvement of intrusion detection system (IDS) evaluation. The work is motivated by two problems. First, the observed increase in the number and the complexity of attacks requires that IDSes evolve to stay capable of detecting new attack variations efficiently. Second, the large number of false alarms that are generated by current IDSes renders them ineffective or even useless. Test and evaluation mechanisms are necessary to determine the quality of detection of IDSes or of their detection algorithms. Unfortunately, there is currently no IDS evaluation method that would be unbiased and scientifically rigorous. During our study, we have noticed that current IDS evaluations suffer from three major weaknesses: 1) the lack of a rigorous methodology; 2) the use of non-representative test datasets; and 3) the use of incorrect metrics.

From this perspective, we have introduced a rigorous approach covering most aspects of IDS evaluation. In the first place, we propose an evaluation methodology that allows carrying out the evaluation process in a systematic way. Secondly, in order to create representative test datasets, we have characterized attacks by classifying attack activities with respect to IDS-relevant manifestations or features. This allows not only to select attacks that will be included in the evaluation dataset but also to analyze the evaluation result with respect to attack classes rather than individual attack instances. Third, we have analyzed a large number of attack incidents and malware samples, such as viruses and worms. Thanks to this analysis, we built a model for the attack process that exhibits the dynamics of attack activities. This model allows us to generate a large number of realistic and diverse attack scenarios.

The proposed methods have been experimented on two very different IDSes to show how general is our approach. The results show that the proposed approach allows overcoming the two main weaknesses of existing evaluations, i.e., the lack of a rigorous methodology and the use of non-representative datasets. Moreover, it allows to better manage the evaluation process and to select representative attack test cases in a flexible manner while providing a better coverage of the attack space.

This dissertation is divided into two parts: in French and in English respectively. Both parts follow the same structure where the first is an extended summary of the second.

Keywords: Security, Intrusion detection system (IDS), Evaluation, Test, Attack, Malware.

Table of Contents

REMERCIEMENTS.....	7
ACKNOWLEDGMENT.....	9
RÉSUMÉ	11
ABSTRACT	13
TABLE OF CONTENTS	15
LIST OF FIGURES	19
LIST OF TABLES	21
PART I	23
INTRODUCTION.....	25
I. ÉTAT DE L'ART : TECHNIQUES D'ÉVALUATION DES IDS	26
I.1. ÉVALUATION PAR TEST	26
I.2. ÉVALUATION ANALYTIQUE	27
I.3. DISCUSSION	27
II. NOUVELLE MÉTHODOLOGIE POUR L'ÉVALUATION DES IDS	28
II.1. VUE GLOBALE.....	28
II.2. BESOINS DES UTILISATEURS ET OBJECTIFS DE L'ÉVALUATION	29
II.3. ENVIRONNEMENT.....	29
II.4. CARACTÉRISTIQUES DE L'IDS.....	29
II.5. PARAMÈTRES DU SYSTÈME ET DES DONNÉES DE TEST	30
II.6. TECHNIQUES D'ÉVALUATION	30
II.7. SÉLECTION DES DONNÉES DE TEST	30
II.8. SÉLECTION DES MÉTRIQUES	30
II.9. ÉVALUATION DIAGNOSTIQUE	31
II.9.1. <i>Modèle de défaillance de l'IDS.....</i>	<i>32</i>
III. CARACTÉRISATION DES ATTAQUES ÉLÉMENTAIRES : CLASSIFICATION DES ATTAQUES ..	35
III.1. ANALYSE DES CLASSIFICATIONS D'ATTAQUES EXISTANTES	35
III.2. DISCUSSION	37
III.3. NOUVELLE CLASSIFICATION.....	38
III.4. SCHÉMA POUR LA SÉLECTION DES CAS DE TEST	40
III.4.2. <i>Exemple de sélection des cas de test.....</i>	<i>41</i>
IV. CARACTÉRISATION ET MODÉLISATION DES SCÉNARIOS D'ATTAQUE	42
IV.1. APPROCHE	42
IV.2. ANALYSE DE MALICIEUX	43
IV.3. MODÈLE DE PROCESSUS D'ATTAQUES.....	44
V. GÉNÉRATION DES SCÉNARIOS D'ATTAQUE	45
VI. MISE EN ŒUVRE ET EXPÉRIMENTATION.....	46
CONTRIBUTIONS ET PROSPECTIVES	47
PART II	49
I. CHAPTER 1: INTRODUCTION.....	51
1.1. MOTIVATION.....	51
1.2. RESEARCH GOAL	52
1.3. APPROACH	52
1.4. CONTRIBUTIONS.....	52

1.5. THESIS OUTLINE	53
II. CHAPTER 2: BACKGROUND	55
2.1. INTRODUCTION	55
2.2. EXAMPLES OF COMMON ATTACKS.....	57
2.2.1. <i>Gathering Security-relevant Information</i>	57
2.2.2. <i>Access Gain Attacks</i>	58
2.2.3. <i>Denial of Service</i>	58
2.2.4. <i>Malware Attacks</i>	59
2.3. SECURITY COUNTERMEASURES	60
2.3.1. <i>Firewalls</i>	60
2.3.2. <i>Antivirus</i>	60
2.4. INTRUSION DETECTION SYSTEMS.....	60
2.4.1. <i>IDS types</i>	62
2.4.2. <i>The source of event data</i>	63
2.4.3. <i>Detection method</i>	63
2.4.4. <i>Locations of data collection and data processing</i>	64
2.5. LIMITATIONS OF INTRUSION DETECTION SYSTEMS	64
2.6. CONCLUSION.....	65
III. CHAPTER 3: AN EVALUATION FRAMEWORK FOR INTRUSION DETECTION SYSTEMS	67
3.1. INTRODUCTION	67
3.2. AN OVERVIEW OF EXISTING IDS EVALUATIONS.....	68
3.2.1. <i>Evaluation by Test</i>	68
3.2.2. <i>Evaluation by analysis</i>	70
3.3. COMMON CRITIQUES OF PREVIOUS EVALUATIONS	73
3.4. EVALUATION FRAMEWORK.....	74
3.5. AN ENGINEERED EVALUATION METHODOLOGY	75
3.5.1. <i>User Needs and Evaluation Goals</i>	76
3.5.2. <i>The Environment</i>	76
3.5.3. <i>The Characteristics of the IDS under Evaluation</i>	76
3.5.4. <i>System and Workload Parameters</i>	76
3.5.5. <i>Evaluation Technique</i>	77
3.5.6. <i>Selecting the Evaluation Workload</i>	77
3.5.7. <i>Selecting Metrics</i>	78
3.6. A DETECTION MODEL.....	78
3.7. IDS FAILURE MODEL.....	79
3.7.1. <i>Failure Modes and Effects Analysis</i>	80
3.7.2. <i>Fault Tree Analysis</i>	81
3.8. BENEFITS OF THE EVALUATION FRAMEWORK.....	84
3.9. CHALLENGING ISSUES IN IDS EVALUATION	84
IV. CHAPTER 4: CHARACTERIZATION OF IDS WORKLOAD	87
4.1. INTRODUCTION	87
4.2. RELATED WORK	88
4.2.1. <i>Attack Models</i>	88
4.2.2. <i>Attack Generation Techniques</i>	90
4.2.3. <i>Attack description Languages</i>	92
4.3. CHARACTERIZING ATTACK ACTIVITIES	92
4.4. LOW-LEVEL ATTACK CHARACTERIZATION (ELEMENTARY ATTACK ACTION CLASSIFICATION).....	93
4.4.1. <i>Analysis of Existing Attack Classifications</i>	93
4.4.2. <i>Classification Requirements</i>	95
4.4.3. <i>New Classification Proposal</i>	96
4.4.4. <i>Classification-based Selection of Attack Test-cases</i>	97
4.4.5. <i>Discussion and Limitations</i>	100
4.5. SCENARIO-LEVEL ATTACK CHARACTERIZATION (ATTACK PROCESS MODEL)	100
4.5.1. <i>Malware Analysis</i>	100
4.5.2. <i>Other Attacks</i>	103
4.5.3. <i>A Generic Attack Process Model</i>	104
4.6. EXAMPLES OF ATTACK ACTIVITY CHARACTERIZATION	106
4.7. METASPLOIT CHARACTERIZATION.....	110

4.7.1. <i>An Overview of Metasploit Framework</i>	110
4.7.2. <i>Exploit Characterization</i>	111
4.8. CHARACTERIZING BACKGROUND ACTIVITIES	112
4.8.1. <i>Network Activity Characterization</i>	112
4.8.2. <i>Host Activity Characterization</i>	114
4.9. CONCLUSION ON ATTACK WORKLOAD CHARACTERIZATION	116
V. CHAPTER 5: MODEL-DRIVEN DATASET GENERATION	119
5.1. INTRODUCTION	119
5.2. MODEL-DRIVEN ATTACK-SCENARIO GENERATION	120
5.2.1. <i>Underlying Models</i>	120
5.3. NOVEL APPROACH FOR ATTACK-SCENARIO GENERATION	124
5.3.1. <i>An overview of Attack Scenarios</i>	124
5.3.2. <i>Scenario Computing</i>	125
5.3.3. <i>Transformation into Concrete Scenarios</i>	126
5.4. DEVELOPMENT OF THE EVALUATION KIT	129
5.4.1. <i>Evaluation Manager</i>	129
5.4.2. <i>Attack Injection Tool (AIT)</i>	132
5.4.3. <i>Traffic Trace Replayer (TrTR)</i>	134
5.5. CONCLUSION ON DATASET GENERATION	138
VI. CHAPTER 6: A PROOF-OF-CONCEPT EVALUATION	139
6.1. EVALUATION GOAL / USER NEEDS	139
6.2. SYSTEM AND WORKLOAD PARAMETERS	139
6.3. EVALUATION TECHNIQUE	140
6.3.1. <i>Hardware Equipment</i>	141
6.3.2. <i>Software Equipment</i>	141
6.4. EVALUATION METRICS	142
6.5. DESIGN OF EXPERIMENTS FOR SELECTIVE EVALUATION	142
6.5.1. <i>Selecting Attack Test-cases</i>	142
6.5.2. <i>Baseline Experiments</i>	143
6.5.3. <i>Background Experiments</i>	143
6.6. RESULTS	144
6.7. CONCLUSION ON OUR EXPERIMENTS	146
VII. CHAPTER 7: CONCLUSIONS	147
7.1. AN OVERVIEW	147
7.2. SUMMARY OF CONTRIBUTIONS	148
7.3. FUTURE WORK	149
7.3.1. <i>Research Work</i>	149
7.3.2. <i>Development and Empirical Work</i>	150
APPENDIX A: MALWARE ANALYSIS	153
APPENDIX B: DATASETS AND DATASET GENERATION TOOLS	161
1. READY-MADE DATASETS	161
2. DATASET GENERATION TOOLS	162
A) BACKGROUND TRAFFIC GENERATORS	162
B) IDS STIMULATORS	162
C) VULNERABILITY SCANNERS AND NETWORK MAPPERS	163
D) PENETRATION TESTING FRAMEWORKS	163
E) ATTACK MUTATION TOOLS	164
APPENDIX C: GLOSSARY	165
REFERENCES	167

List of Figures

PART I:

FIGURE 1 : NOTRE MÉTHODOLOGIE POUR L'ÉVALUATION DES IDS.....	28
FIGURE 2 : MODÈLE DE DÉFAILLANCE DES IDS	32
FIGURE 3 : ARBRE DE FAUTE GÉNÉRIQUE POUR LES IDS BASÉS SIGNATURE.	33
FIGURE 4 : NOUVEAU SCHÉMA DE CLASSIFICATION D'ATTAQUE	40
FIGURE 5 : SCHÉMA DE SÉLECTION D'ATTAQUE BASÉE SUR L'ARBRE DE CLASSIFICATION	41
FIGURE 6. MODÈLE DE PROCESSUS D'ATTAQUE	45
FIGURE 7 : MACHINE À ÉTAT REPRÉSENTANTE LE PROCESSUS D'ATTAQUE	46
FIGURE 8 : L'ARCHITECTURE GLOBALE DE NOS OUTILS D'ÉVALUATION	47

PART II:

FIGURE I.1: THE ROADMAP OF THE THESIS.....	54
FIGURE II.1: GROWTH IN THE NUMBER OF INCIDENTS HANDLED BY US-CERT FROM 1993 TO 2003.	56
FIGURE II.2: GROWTH IN NUMBER OF VULNERABILITIES CATALOGUED BY US CERT.....	57
FIGURE II.3: THE CONTINUOUS GROWTH OF THE NUMBER OF CAPTURED MALWARES.....	57
FIGURE II.4: IDS COMPONENTS.	62
FIGURE II.5: INTRUSION DETECTION PROCESS.	62
FIGURE III.1: AN EXAMPLE OF A ROC CURVE FOR TWO IDSes.....	69
FIGURE III.2: THE FOUNDATIONS OF THE EVALUATION FRAMEWORK.....	74
FIGURE III.3: IDS EVALUATION METHODOLOGY.....	75
FIGURE III.4: IDS FAILURE MODEL.	80
FIGURE III.5: GENERIC FAULT TREE ANALYSIS FOR INTRUSION DETECTION SYSTEMS.	83
FIGURE III.6: PROCESS OF IDS EVALUATION.	84
FIGURE III.7: ENHANCED PROCESS FOR IDS EVALUATION.	84
FIGURE IV.1: INTERSECTION BETWEEN ATTACK AND NORMAL USE ACTIVITIES.	88
FIGURE IV.2: A CLASSIFICATION SCHEME OF ELEMENTARY ATTACK.	97
FIGURE IV.3: TEST CASES PRODUCED BY THE CTE TOOL.	99
FIGURE IV.4: A SIMPLIFIED FLOWCHART FOR CODERED I. FIGURE IV.5: A SIMPLIFIED FLOWCHART FOR CODERED II.	102
FIGURE IV.6: A SIMPLIFIED FLOWCHART FOR SASSER.....	103
FIGURE IV.7: A SIMPLIFIED FLOWCHART FOR TRINOO.....	103
FIGURE IV.8: ATTACK PROCESS MODEL.....	105
FIGURE IV.9: CLASSIFICATION-BASED CHARACTERIZATION OF <i>METASPLOIT</i> 'S EXPLOITS.	112
FIGURE IV.10: AN EXTRACT FROM APACHE ACCESS-LOG ENTRIES (SOURCE: WWW.MONITORWARE.COM).....	115
FIGURE IV.11: AN EXTRACT FROM APACHE ERROR-LOG ENTRIES (SOURCE: WWW.MONITORWARE.COM).	115
FIGURE V.1: EVOLUTION IN THE NUMBER OF EXECUTED COMMANDS DURING ATTACK SESSIONS ON THE 1 ST MACHINE.....	122
FIGURE V.2: EVOLUTION IN THE NUMBER OF EXECUTED COMMANDS DURING ATTACK SESSIONS ON THE 2 ND MACHINE.	122
FIGURE V.3: MOST FREQUENTLY EXECUTED COMMANDS.	123
FIGURE V.4: PERCENTAGE OF COMMANDS AS ATTACK STEPS.....	123
FIGURE V.5: GRAPH REPRESENTING CONNECTIONS AND POSSIBLE PATHS BETWEEN ATTACK STEPS (I.E., ATTACK CYCLE).	125
FIGURE V.6: AN OVERVIEW OF THE EVALUATION KIT STRUCTURE.	129
FIGURE V.7: DATA STRUCTURE THAT STORES THE INFORMATION OF THE GRAPH DESCRIBED IN FIGURE V.5.	130
FIGURE V.8: AN EXCERPT FROM THE SCENARIO CALCULATION SCRIPT WRITTEN IN MOZART LANGUAGE.	131
FIGURE V.9: MAIN COMPONENTS OF AIT THAT CORRESPOND TO MAIN ENTITIES INVOLVED IN ATTACK SCENARIOS GENERATION.	133
FIGURE V.10: MODIFIED COMPONENTS AND CLASSES USED OF METASPLOIT/REX.	134
FIGURE V.11: ARCHITECTURE OF TrTR.	136
FIGURE V.12: A SIMPLIFIED CLASS DIAGRAM FOR CLASSES USED IN TrTR.	136
FIGURE V.13: THE WORK-AROUND SOLUTION FOR LIMITATION IN JPCAP.	138
FIGURE VI.1: A SCREENSHOT OF METASPLOIT CONSOLE AFTER LOADING OUR PLUGIN.....	140
FIGURE VI.2: THE CONFIGURATION PANEL OF THE EVALUATION MANAGER.	140
FIGURE VI.3: THE EVALUATION TEST BED.	141

List of Tables

PART I:

TABLE 1 : CARACTÉRISTIQUES PRINCIPALES DES TECHNIQUES D'ÉVALUATION DES IDS.....	27
TABLE 2 : PROPOSITION ET CARACTÉRISATION DE MÉTRIQUES	31
TABLE 3 : AMDE GÉNÉRIQUE DES IDS	34

PART II:

TABLE III.1: A COMPARISON OF IDS EVALUATIONS.	72
TABLE III.2: MAIN FEATURES OF IDS EVALUATION TECHNIQUES.	77
TABLE III.3: EXAMPLE OF SUGGESTED METRICS.	78
TABLE III.4: GENERIC FMEA FOR IDS.....	80
TABLE IV.1: A COMPARISON BETWEEN DIFFERENT EVALUATION TOOLS/TECHNIQUES.....	91
TABLE IV.2: AVAILABLE PAYLOADS FOR EACH PLATFORM (METASPLOIT VERSION 3.1).	110
TABLE IV.3: VARIANTS OF SHELL PAYLOADS FOR WINDOWS (METASPLOIT VERSION 3.1).	111
TABLE IV.4: LIST OF SOME AVAILABLE FACILITIES.	114
TABLE IV.5: LIST OF POSSIBLE LOGGING LEVELS.	115
TABLE V.1: MOST FREQUENTLY USED USERNAME AND PASSWORDS.	122
TABLE V.2: CONNECTION MATRIX.	126
TABLE V.3: EXAMPLES OF ATTACK TOOL CHARACTERIZATION.....	128
TABLE V.4: INFORMATION ON THE SIZE OF SEARCH SPACE.	130
TABLE VI.1: EXAMPLE OF CALCULATED METRICS.....	143
TABLE VI.2: LIST OF ATTACKS (METASPLOIT EXPLOITS) INCLUDED IN THE EVALUATION DATASET AND THEIR DETECTION STATUS.....	144
TABLE VI.3: COMPARISON OF THE MAIN FEATURES BETWEEN OUR TOOLKIT AND THE OTHER TOOLS.....	146

PART I

Évaluation des Systèmes de Détection d'Intrusion

Introduction

Les systèmes de détection d'intrusion (IDS)¹ sont parmi les outils de sécurité les plus récents. On peut les classer en différents types selon leurs caractéristiques, par exemple selon leurs techniques de détection, leur architecture ou la portée de détection {Debar00}, {Debar05}. Malheureusement, malgré leur utilité, en pratique la plupart des IDS souffrent plus ou moins de deux problèmes : le nombre important de faux positifs et de faux négatifs. Les faux positifs (c'est-à-dire les fausses alertes) sont générés lorsque l'IDS identifie des activités normales comme des intrusions, alors que les faux négatifs correspondent aux attaques ou intrusions qui ne sont pas détectées (aucune alerte n'est générée).

Les concepteurs des IDS essayent de surmonter ces limitations en développant de nouveaux algorithmes et architectures. Il est donc important pour eux d'évaluer les améliorations apportées par ces nouveaux dispositifs. De même, pour les administrateurs réseau et systèmes, il serait intéressant d'évaluer les IDS pour pouvoir choisir celui qui répond le mieux à leurs besoins avant de l'installer sur leurs réseaux ou systèmes, mais aussi de continuer à évaluer son efficacité en mode opérationnel.

Malheureusement, un nombre élevé de faux positifs et de faux négatifs persiste dans les nouvelles versions des IDS, ce qui laisse à penser que les améliorations apportées ne sont pas à la mesure des efforts continus de recherche et développement dans le domaine de la détection d'intrusion. Ceci est essentiellement dû à l'absence de méthodes efficaces d'évaluation des outils de sécurité en général, et des IDS en particulier.

Il est vrai que ces dernières années, de nombreuses tentatives d'évaluation ont eu lieu {Puketza96}, {Puketza97}, {Lippmann00a}, {Lippmann00b}, {Debar98}, {Debar02}, {Alessandri04}. Cependant, elles souffrent pour la plupart de limitations sérieuses {Mell03}, {McHugh00a}. En particulier, le comportement de l'IDS pendant la phase de l'évaluation est souvent très différent de son comportement en environnement réel. Les conclusions qui peuvent être tirées sont donc incorrectes ou tout au moins biaisées.

Il n'est donc pas étonnant de constater que plusieurs études menées dans le domaine de la détection d'intrusion ont clairement identifié l'évaluation des IDS comme une thématique de recherche prioritaire {Mukherjee94}, {Axelsson98a}, {Jones00}, {Allen00}, {Lundin02}.

Malheureusement, l'évaluation des systèmes de détection d'intrusion s'avère une tâche difficile, qui exige notamment une connaissance profonde de techniques relevant de disciplines différentes, en particulier la détection d'intrusion, les méthodes d'attaques, les maliciels², les réseaux et systèmes, les techniques de test et d'évaluation, etc.

La tâche est d'autant plus difficile que les IDS doivent non seulement être évalués en conditions normales, mais aussi et surtout en environnement malveillant, en tenant compte notamment de modes d'utilisation inattendus et parfois même inconnus (ceci est également vrai pour pratiquement tous les outils dédiés à la sécurité comme les pare-feux, les IPS³ et les antivirus). Toutes ces considérations rendent difficile la tâche de construction de données représentatives pour l'évaluation.

Dans ce contexte, l'objectif ultime de ce travail est d'améliorer la qualité des systèmes de détection d'intrusions en fournissant des procédures rigoureuses d'évaluation, mais aussi des outils de génération de données représentatives de test. Notons que les outils d'évaluation que nous proposons seront non seulement au service des développeurs d'IDS, mais également des utilisateurs d'IDS pour leur permettre,

1 Pour *Intrusion Detection Systems* en anglais.

2 Programmes malveillants, ou *malware* en anglais.

3 Pour *Intrusion Prevention Systems* en anglais.

par exemple, de comparer différents IDS et de choisir le plus adapté à leur réseau ou système. Nous nous concentrons essentiellement sur l'amélioration de la procédure d'évaluation elle-même et sur la création de données représentatives. Ceci aidera notamment à réduire le temps nécessaire à la construction des données d'évaluation tout en privilégiant leur représentativité, et permettra à l'évaluateur d'accorder plus d'attention à la conception et la mise en place des expérimentations.

En suivant la même structure que la partie en anglais, le reste de ce résumé en français est organisé comme suit : la section I discute l'état de l'art. La section II présente notre méthodologie d'évaluation. Les sections III et IV sont dédiées à la caractérisation et la modélisation des données réelles traitées par les IDS. Dans notre contexte, ces données sont de deux natures : des données de fond qui correspondent à des activités normales d'opération (ex. trafic réseau) et des données d'attaques qui correspondent à des actions exécutées par les attaquants. Nous concentrons notre étude sur la génération des données d'attaques, même si nous avons également abordé le problème de la génération de trafic de fond. Nous introduisons une nouvelle classification des attaques et nous développons des approches complémentaires destinées à la sélection des attaques pour évaluer les IDS. Ensuite, nous présentons un nouveau modèle de processus d'attaques. La Section V décrit notre approche pour générer des données représentatives d'évaluation en nous appuyant sur le modèle présenté dans la section IV. Enfin, nous présentons la conclusion ainsi que les perspectives de ce travail.

I. État de l'art : Techniques d'évaluation des IDS

On peut distinguer deux grandes classes d'évaluation des IDS : l'évaluation analytique et l'évaluation par test. Généralement, la première technique se base sur une modélisation du système étudié, et peut être appliquée à toute étape du cycle de développement, tandis que la deuxième injecte des données réelles à une implémentation (ex. un prototype) du système sous test. Ces deux techniques sont détaillées dans le reste de cette section.

I.1. Évaluation par test

Nombreux sont les travaux antérieurs qui se sont intéressés à l'évaluation par test des IDS. Citons, à titre d'exemple, l'un des plus anciens, celui de Puketza *et al.* {Puketza96}, {Puketza97}. Ce travail utilise un ensemble de scripts pour simuler des cas de test (pour des sessions normales et intrusives) en se basant sur la politique de sécurité de l'organisation. Il met en œuvre, non seulement des intrusions séquentielles provenant d'une seule session d'attaque, mais aussi des intrusions simultanées provenant de plusieurs sessions. La procédure de test suivie vise trois objectifs majeurs : identification des intrusions, utilisation des ressources et test aux limites ou *stress testing*.

Un autre travail important a été réalisé par IBM Zurich en vue d'une évaluation comparative de plusieurs IDS. Pour cela, une plateforme de test générique a été créée en utilisant plusieurs clients et serveurs contrôlés par une station unique. Le trafic de fond (trafic normal sur le réseau et événements non intrusifs du système) est généré en utilisant des suites de tests construits par les développeurs du système d'exploitation, tandis que les attaques sont sélectionnées dans une base de données propre à IBM. Le rapport publié indique seulement que quatre IDS intégrés sur hôte (*Host-based Intrusion Detection Systems*) ont été testés contre des attaques FTP, mais ne dit rien sur les métriques utilisées ni sur les détails des résultats obtenus.

L'un des projets les plus ambitieux a été celui sponsorisé par la DARPA (en 98 et 99), en collaboration avec le laboratoire Lincoln du MIT {Lippmann00a}, {Lippmann00b}. Le but était de fournir un ensemble significatif de données de test, comprenant trafic de fond et activités intrusives (c'est-à-dire du trafic intrusif ou des événements systèmes causés par des attaques). Le trafic de fond était déduit des données statistiques collectées sur le réseau des bases de l'*Air Force* alors que les attaques étaient générées par des scripts créés spécialement, mais aussi par des scripts collectés à travers des sites spécialisés et des listes de diffusion. Les données collectées concernaient à la fois des HIDS (par exemple données d'audit de stations Solaris, *disk dump* de machines UNIX et BSM « *Sun Basic Security Module* ») et des IDS dédiés réseau, communément appelés NIDS, pour *Network IDS*.

Le jeu de test DARPA'98 utilisait environ 300 attaques (classées en 38 types d'attaques), tandis que DARPA'99 utilisait environ 50 types d'attaques.

John McHugh a fortement critiqué les évaluations de la DARPA. Il a surtout insisté sur trois problèmes essentiels respectivement liés à la génération des données de test, aux métriques utilisées et à la présentation des résultats [McHugh00a].

1.2. Évaluation analytique

Contrairement à l'évaluation par test, l'évaluation analytique porte sur la définition de méthodes permettant de maîtriser le modèle. Le travail le plus important dans ce domaine est celui d'Alessandri, qui a proposé un modèle descriptif de l'IDS [Alessandri04] visant à pallier les problèmes cités précédemment et à fournir une documentation aux concepteurs. L'évaluation consiste à :

- classer les attaques selon leurs caractéristiques observables par l'IDS ;
- décrire le comportement de l'IDS, et notamment la manière de collecter et d'analyser les informations ;
- déterminer si un certain type d'attaques sera détectable par l'IDS ou pas.

Notons que globalement, chaque catégorie d'évaluation des IDS, qu'elle soit analytique ou par test, possède des avantages mais aussi des inconvénients. Le tableau suivant dresse une comparaison globale des principales caractéristiques des deux techniques d'évaluation des IDS.

Table 1 : Caractéristiques principales des techniques d'évaluation des IDS

	<i>Évaluation par test</i>	<i>Évaluation analytique</i>
Phase du cycle de vie	Après l'implémentation	Spécification et conception
Cible	Prototype, implémentation	Modèle de l'IDS
Entrées	Données réelles ou synthétisées	Modèle et classes d'attaques
Activités normales de fond	Peuvent être considérées	Ne sont pas considérées
Caractéristiques évaluées	Performances, capacités de détection	Capacités de détection
Effets de l'environnement	Peuvent être considérés	Ne sont pas considérés
Niveau de connaissance requis	Connaissance sur l'IDS non obligatoire, compatible avec une évaluation boîte noire	Bonne connaissance sur la structure et la conception de l'IDS, correspond à une évaluation boîte blanche

1.3. Discussion

Globalement, nous identifions plusieurs faiblesses dans les méthodes classiques d'évaluation. Le premier point à noter est l'*utilisation d'approches non systématiques*. En effet, la plupart des méthodes de test des IDS sont des approches plutôt *ad hoc* : la sélection des paramètres du système, des facteurs, des métriques et des données de test est souvent arbitraire et non justifiée.

Le deuxième point est la *non représentativité* des données de test. Ni le trafic de fond, ni les données d'attaques ne correspondent à la réalité d'Internet. L'IDS évalué se comporte ainsi différemment sous test et quand il est déployé dans un environnement réel. Si on prend, par exemple, les données de test de la DARPA, le trafic généré (de quelques Kbits/s) est considérablement plus faible que celui attendu (plusieurs Mbits/s). De plus, les données de test (trafic de fond) ont été basées sur des statistiques prises dans différents réseaux, mais leurs caractéristiques, en particulier celles liées à la génération de fausses alertes, n'ont pas été validées.

Le troisième point est la *sensibilité des données de test aux variations de l'environnement*. Malheureusement, le comportement des algorithmes de détection d'anomalies (basés sur des méthodes probabilistes, des réseaux de neurones, des arbres de décision, etc.) est étroitement lié à l'environnement. Par conséquent, la nature, la régularité et la variation des données collectées lors de la phase d'apprentissage (ou des données de test) auront un impact important sur les performances de l'IDS.

La dernière faiblesse que nous soulevons est la *non-pertinence des métriques*. On a souvent tendance à sélectionner des métriques faciles à mesurer, sans voir l'étendue réelle de leur efficacité. Parmi les métriques qui sont souvent utilisées dans l'évaluation des IDS, on peut citer le taux de détection, le rapport de détection (*detection ratio*), le taux de fausses alertes, le rapport de fausses alertes (*false alarm*

ratio), etc. Clairement, ces métriques ne sont pas toujours adéquates. Si on prend le taux de fausses alertes par exemple, il peut avoir plusieurs définitions selon la nature du dénominateur. Selon les différentes études, ce taux peut être défini comme le nombre de fausses alertes divisé par le nombre total d'alertes, ou par le nombre de sessions, ou par le nombre de paquets. Néanmoins, même si le taux de fausses alertes par paquet peut avoir un sens pour un IDS qui applique une simple analyse de chaque paquet, il n'en a pas de même pour un IDS qui analyse des sessions et contrôle l'état de connexion.

Dans les sections suivantes nous proposons des solutions pour pallier ces limitations. Dans cette optique, la section suivante commence par proposer une nouvelle méthodologie systématique d'évaluation des IDS.

II. Nouvelle méthodologie pour l'évaluation des IDS

II.1. Vue globale

La figure 1 présente, de manière globale, notre méthodologie, dont l'objectif principal est le développement d'une évaluation systématique suivant des étapes bien structurées. Deux grandes phases peuvent être identifiées : une phase de *préparation* et une phase d'*expérimentation*.

La procédure commence par identifier les besoins de l'utilisateur final (crédibilité, taux de détection, réactivité, facilité de mise en œuvre, adaptabilité, performances, diversité des canaux d'alerte, etc.). Ensuite il conviendrait d'identifier les caractéristiques de l'environnement et du contexte de test (dans lequel l'IDS sera déployé) ainsi que celles de l'IDS cible de l'évaluation.

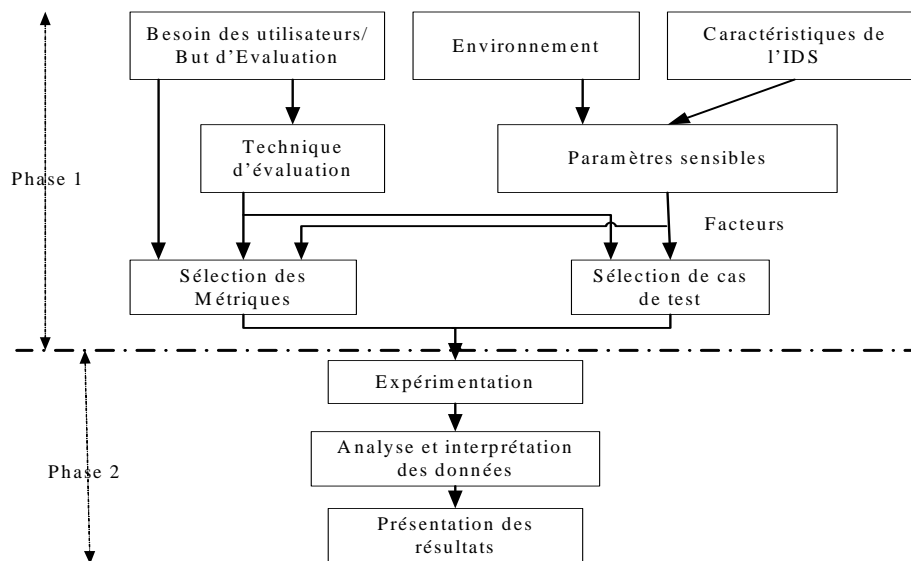


Figure 1 : Notre méthodologie pour l'évaluation des IDS

Après ces étapes, l'évaluateur sera en mesure de sélectionner la technique d'évaluation la plus appropriée : évaluation analytique ou par test. Les facteurs (paramètres contrôlables et ajustables par l'évaluateur) du système et des données de test sont également spécifiés. Par exemple, la bande passante et les tailles de paquet de trafic de fond sont des facteurs liés aux données de test des NIDS⁴. Bien évidemment, les facteurs ainsi que la technique d'évaluation ont un impact sur la sélection des métriques et des données de test.

Il est important de noter que, souvent, il suffit de quelques modifications mineures pour pouvoir réutiliser dans plusieurs évaluations les mêmes besoins des utilisateurs ainsi que les caractéristiques de l'environnement et/ou de l'IDS. Une partie des étapes précédentes peut donc être factorisée et réutilisée moyennant quelques paramétrages mineurs.

4 Pour *Network-based Intrusion Detection System* en anglais.

Ensuite, vient la deuxième phase, la phase d'expérimentation, qui consiste à :

- préparer et exécuter les expériences ;
- prendre les mesures et calculer les résultats ;
- analyser et interpréter les résultats obtenus ; et finalement
- présenter les résultats.

Les étapes de la figure 1 seront détaillées dans la suite de cette section.

II.2. Besoins des utilisateurs et objectifs de l'évaluation

Les IDS s'intègrent dans un contexte et une architecture qui impose des contraintes pouvant être très diverses. C'est pourquoi il n'existe pas de grille d'évaluation unique pour ce type d'outils. Pourtant l'utilisateur peut dégager un certain nombre de critères, qui devront nécessairement être pondérés en fonction du contexte de l'étude. On peut distinguer :

- les besoins fonctionnels, qui concernent essentiellement la qualité de la détection ;
- les besoins de performances, qui traitent des aspects liés à la vitesse, l'utilisation de la mémoire, la charge CPU ;
- les besoins d'utilisation, qui concernent les problèmes liés à la facilité de configuration, la mise à jour et utilisation, l'ergonomie, etc.

Les besoins des utilisateurs doivent être traduits en exigences pour les IDS. Une exigence pourrait être par exemple :

- avoir une bonne détection (détecter toutes les attaques connues, mais aussi certaines attaques non encore connues) ;
- générer peu de fausses alertes ;
- résister aux attaques visant l'IDS lui-même, notamment les attaques par déni de service ;
- nécessiter peu de ressources ;
- tolérer les fautes (avec redondance de certains composants de manière à continuer à fonctionner même en cas d'intrusions).

On peut également approfondir l'analyse en spécifiant l'objectif de l'évaluation : s'agit-il d'une évaluation pour des administrateurs de systèmes ou pour des développeurs d'IDS. En effet, un administrateur pourrait tout simplement viser une étude comparative afin de sélectionner un IDS ou pour comparer les IDS déployés dans son propre système, tandis que le développeur pourrait exiger plus de détails et chercher à comprendre et interpréter le comportement de l'IDS. Ainsi, si une évaluation du type « boîte noire » peut suffire à une évaluation orientée utilisateur, elle ne peut pas satisfaire les besoins du développeur.

II.3. Environnement

Bien évidemment, l'environnement peut différer d'une évaluation à une autre. Par exemple, les caractéristiques d'un réseau académique sont différentes de celle d'un réseau militaire, elles-mêmes différentes d'un environnement commercial. Des connaissances sur le système d'exploitation, les serveurs, les plateformes, les fichiers ainsi que les bases de données spécifiques à chaque environnement, peuvent être pertinentes pour la procédure d'évaluation, par exemple, pour aider à choisir le type de trafic de fond, les attaques à considérer ou à injecter à l'IDS sous test.

II.4. Caractéristiques de l'IDS

Les caractéristiques architecturales, algorithmiques ou les caractéristiques d'implémentation peuvent influencer la méthode d'évaluation.

Au niveau architectural, la plupart des IDS obéissent au modèle CIDF (*Common Intrusion Detection Framework*) de l'IETF [Chen98]. En effet, un IDS peut être un programme monolithique installé sur une seule machine ou distribué sur plusieurs hôtes qui gèrent plusieurs processus : captures et prétraitement d'événements, analyse et génération des alertes. Plusieurs sondes peuvent être déployées sur le réseau ou dans différents hôtes.

Concernant les algorithmes et l'implémentation, on distingue plusieurs implémentations des algorithmes de détection, par exemple les algorithmes statistiques, les réseaux de neurones, les algorithmes génétiques. Si l'IDS étudié est basé sur des agents, d'autres caractéristiques comme l'intelligence, l'autonomie et la mobilité peuvent être prises en considération.

II.5. Paramètres du système et des données de test

Étant donné qu'il est pratiquement impossible de calculer tous les aspects liés à l'environnement, les évaluateurs ne peuvent ajuster et modifier que les paramètres contrôlables, nommés ici *facteurs*. Parmi l'ensemble des facteurs, il faut en identifier ceux qui vont nous aider à comprendre le système évalué. Par exemple, pour les NIDS, ces facteurs peuvent être : la composition du trafic (type de trafic, longueur des paquets, contenu de la charge utile, utilisation de la bande passante). Pour les HIDS⁵, on peut identifier des facteurs comme le système (plateforme, version, ...) où l'IDS est déployé, les applications et services qui tournent sur la machine, les vulnérabilités non encore corrigées, etc.

On peut également considérer des facteurs spécifiques à l'IDS lui-même, notamment les règles de signature, l'algorithme de détection, etc. Le type d'algorithme d'apprentissage ainsi que le seuil du profil sont deux exemples de facteurs associés aux IDS basés sur la détection d'anomalie.

II.6. Techniques d'évaluation

Le choix de la technique d'évaluation (analytique ou par test) dépend de l'objectif de l'évaluation ainsi que de l'étape dans laquelle on l'applique. En effet, comme indiqué dans Table 1, alors que l'évaluation analytique peut être appliquée dès les phases avancées de conception, le test n'est possible que sur des IDS déjà implémentés. Par ailleurs, l'évaluation analytique nécessite une bonne connaissance de la structure et du fonctionnement de l'IDS, tandis que le test peut se contenter de considérer l'IDS comme une boîte noire.

II.7. Sélection des données de test

Le choix et la construction des données de test (activités normales et intrusives) peuvent prendre en considération plusieurs points notamment :

- le type des fonctions et services qui seront testés, par exemple, la capacité du NIDS à détecter les attaques par fragmentation ;
- le niveau de détail : en effet, certains algorithmes de détection sont plus sensibles au contenu de la charge des paquets {Antonatos04} ;
- l'impact des composants externes : par exemple, si un pare-feu est installé derrière l'IDS, il bloque une partie du trafic et l'IDS ne reçoit donc plus ce trafic pour l'analyser (puisque'il est déjà bloqué par le pare-feu) ;
- la reproductibilité : les résultats doivent être facilement reproductibles.

Notons, qu'en plus des données de la DARPA, d'autres générateurs plus performants (de trafic de fond et d'attaques) sont actuellement disponibles {Sommers04}, {Marty02}. Néanmoins, ces outils doivent être bien choisis, configurés, ajustés et adaptés au système étudié.

II.8. Sélection des métriques

La définition des métriques est la pierre angulaire du processus d'évaluation. En effet, si elles sont mal définies, les résultats de l'évaluation peuvent être faux ou biaisés. Dans la table 2, nous proposons et nous décrivons un ensemble de métriques pouvant être utilisé dans ce domaine.

Ce tableau propose et classe les métriques selon qu'elles sont reliées aux ressources ou à la détection. Cette dernière classe est divisée en métriques microscopiques (reliées aux composants) et macroscopiques (au niveau système). La première sous-classe donne une idée globale sur les fonctions de l'IDS ; tandis que la deuxième vise à mesurer les capacités de l'IDS selon les types d'attaques et les fonctionnalités des composants de l'IDS, par exemple, pour mesurer les capacités des sondes à capturer les événements, nous suggérons la métrique « *Intrusive event drop ratio* ».

Bien entendu, certaines hypothèses générales doivent être prises en compte dans cette étape de la procédure, en l'occurrence :

- il n'y a pas de métriques absolues, mais seulement des métriques relatives à l'ensemble des cas de test ; l'importance des métriques dépend de l'objectif et de la cible de l'évaluation ;

5 Pour Host based Intrusion Detection System en anglais

- pour plus d'expressivité, les résultats de l'évaluation doivent être spécifiés à travers plusieurs métriques ;
- on ne considère que les métriques mesurables et ayant un sens contribuant à la compression, nous évitons donc les métriques dénuées de sens (ex., taux de détection) ou génériques et ambiguës (ex., résistance aux attaques de déni de service) ;
- pour les métriques microscopiques, il est important de tracer les événements intrusifs dans les données de test.

Table 2 : Proposition et caractérisation de métriques

Métriques macroscopiques reliées à la détection	Définition
<i>Ratio de détection</i>	nombre d'attaques détectées / nombre total d'attaques
<i>Ratio de fausse alarme</i>	nombre de fausses alertes générées / nombre total d'alertes générées
Métriques microscopiques reliées à la détection	Définition
<i>Ratio de détection par type d'attaque</i>	nombre d'attaques détectées d'un certain type / nombre total d'attaques de ce type
<i>Ratio de fausse alarme par type d'attaque</i>	nombre de fausses alertes générées pour un certain type d'attaques / nombre total de fausses alertes générées pour ce type
<i>Événements capturés / attaques non détectées</i>	nombre d'attaques non détectées dont les événements ont été capturés / nombre total d'attaques non détectées
<i>Événements non capturés / attaques détectées</i>	nombre d'attaques dont les événements n'ont pas été capturés / nombre total d'attaques non détectées
<i>Ratio d'événements intrusifs non capturés</i>	nombre d'événements intrusifs non capturés / nombre total des événements intrusifs
Métriques reliées à l'utilisation des ressources	Définition
<i>Utilisation CPU</i>	Pourcentage du CPU utilisé par IDS
<i>Utilisation de la mémoire</i>	Pourcentage de la mémoire utilisée par IDS

La méthodologie ainsi développée sera recentrée dans la section suivante sur les besoins des développeurs. En effet, dans ce cas, la première étape de notre méthodologie – l'identification des besoins de l'utilisateur et du but de l'évaluation – fournira la possibilité d'identifier et d'éliminer les erreurs introduites dans la phase d'expérimentation (ceci, en vue de l'amélioration de la capacité de détection, des performances, etc.).

II.9. Évaluation diagnostique

Afin d'aider le développeur à évaluer son IDS, à interpréter son comportement, à découvrir les causes de défaillance et à les localiser au niveau des composants, nous proposons dans cette section une évaluation diagnostique. En effet, cette tâche nous paraît importante, d'autant plus qu'actuellement, excepté quelques travaux s'intéressant aux algorithmes de détection, très peu de travaux se sont intéressés à relier le comportement de l'IDS au composant défaillant.

L'idée principale est de combiner les techniques d'évaluations (par test et par analyse du modèle) et les méthodes de sûreté de fonctionnement, en particulier, les arbres de défaillance, appelés aussi arbres de fautes (AdF ou AdD).

Un AdF n'est rien d'autre qu'un diagramme logique utilisant une structure arborescente pour représenter les causes de défaillances et leurs combinaisons conduisant à un événement redouté (racine de l'arbre). La réduction des arbres de faute à partir du calcul des coupes minimales, permet d'identifier les chemins critiques. On en déduit les éléments matériels et logiciels du système dont la défaillance contribue le plus à la réalisation de l'événement redouté. Les arbres de faute peuvent être quantifiés, permettant ainsi de calculer la disponibilité et la fiabilité du système modélisé.

Ci-dessous, nous construisons un AdF générique des IDS qui pourrait aider à mieux interpréter le comportement de l'IDS et de localiser la source de défaillance ; les fautes d'implémentation peuvent ainsi être découvertes et corrigées. Par ailleurs, l'AdF pourrait guider la sélection des cas de test et le processus d'expérimentation.

II.9.1. Modèle de défaillance de l'IDS

De manière globale, quelque soit la technique de détection ou l'architecture de l'IDS, les composants sont similaires et donc le processus de détection peut être décrit comme suit :

- une ou plusieurs sondes (du même type ou pas) pour capturer les événements douteux ;
- un ou plusieurs préprocesseurs pour prétraiter et extraire les informations pertinentes à partir des données collectées par les sondes ;
- un ou plusieurs détecteurs pour analyser les informations fournies par le préprocesseur et éventuellement générer des alertes ;
- une base de données contenant les signatures (pour les IDS à base de signatures) ou les profils (pour les IDS à détection d'anomalies).

Dans cette section, nous allons analyser cette structure générique afin de chercher les causes des défaillances. La figure 2 montre le modèle de défaillance de l'IDS, où nous considérons la faute comme étant :

- l'introduction d'une action intentionnelle (ex. déni de service, fragmentation) afin de contourner l'IDS ou le désactiver ; ou
- l'occurrence d'une activité bénigne non-malveillante qui affecte l'IDS, ex. une surcharge/saturation ; ou
- une faute d'implémentation ou de configuration (absence d'une signature, par exemple).

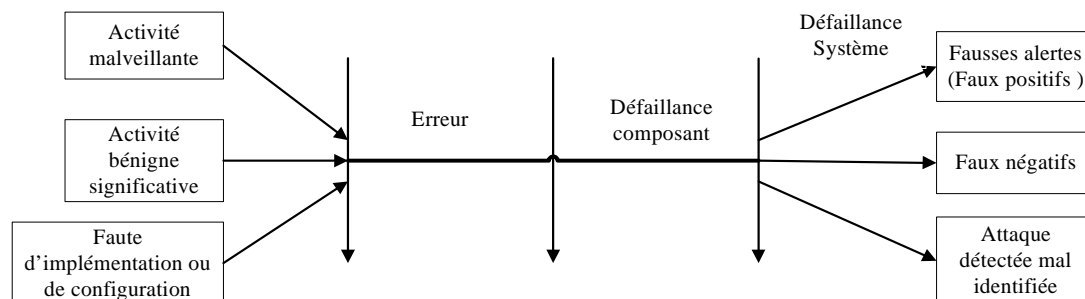


Figure 2 : Modèle de défaillance des IDS

L'IDS est donc défaillant si un de ses composants l'est ou s'il y a une rupture de communication entre les composants de l'IDS. Ceci pourrait se traduire par une fausse alerte, la non-détection d'une attaque (faux négatif) ou une mauvaise identification de l'attaque détectée.

Une analyse qualitative des défaillances de l'IDS peut être effectuée par l'Analyse des Modes de Défaillances et de leurs Effets (AMDE). L'AMDE est une méthode d'élimination et de prévision de faute. Elle vise principalement à faire l'analyse qualitative de la fiabilité d'un système par la définition, en termes de gravité, des effets de chaque mode de défaillance (leur criticité) sur d'autres éléments et/ou fonctions du système {Avizienis04}, {Bouti94}.

Ainsi, dans notre étude, l'AMDE permettrait d'identifier les défaillances potentielles de chaque composant de l'IDS et de donner une idée sur la cause, alors qu'une analyse par arbre de fautes (AdF) facilite l'identification des combinaisons de défaillances des composants qui pourrait conduire à une défaillance totale de l'IDS.

La table 3 fournit une structure générique de l'AMDE pour les IDS où les défaillances, les causes, les effets ainsi que les actions correctives possibles sur chaque composant sont identifiées.

Par ailleurs, étant donné que la construction de l'arbre de défaillance repose sur l'étude des événements entraînant un événement redouté, nous avons réalisé les étapes suivantes :

- dans un premier temps, il faut définir l'événement redouté et l'analyser en spécifiant ce qu'il représente et dans quel contexte il peut apparaître ;
- puis dans un deuxième temps, il faut représenter graphiquement les relations de cause à effet par des portes logiques (ET, OU) qui permettent de spécifier le type de combinaison entre les événements intermédiaires qui conduisent à l'événement analysé.

Le résultat de ces deux étapes est présenté par la figure 3, qui montre que l'événement redouté (défaillance de l'IDS) survient quand l'IDS ne détecte pas l'attaque ou s'il génère de fausses alertes ; la détection de l'attaque sans l'identifier correctement peut également être considérée comme une défaillance.

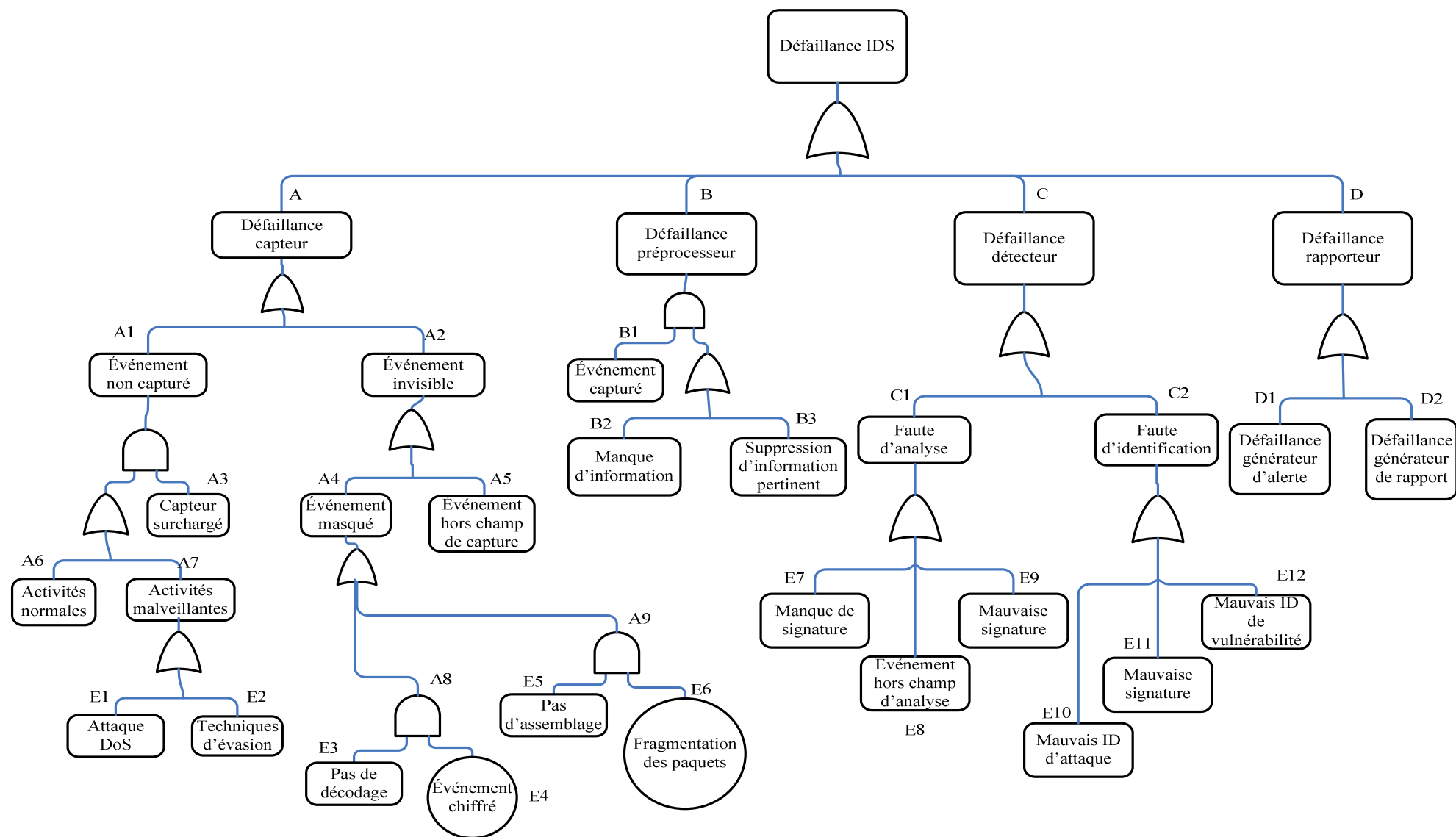


Figure 3 : Arbre de faute générique pour les IDS basés signature.

La défaillance de l'IDS peut être due à une défaillance de la sonde, du préprocesseur, du détecteur ou du générateur d'alertes.

Comme indiqué dans la table 3, la sonde défaille si elle ne capture pas l'événement, c'est-à-dire si l'événement est hors de la portée de la sonde ou si l'attaquant arrive à masquer le caractère malveillant de l'attaque et faire en sorte qu'elle passe inaperçue. Néanmoins, l'invisibilité de l'événement n'est pas la seule chose qui peut conduire à la défaillance de la sonde, elle peut également défailir si elle est saturée, suite à une attaque par déni de service, mais aussi suite à une surcharge du réseau par des activités normales.

La défaillance du préprocesseur survient si les informations pertinentes (pour le traitement et la détection) sur les événements sont supprimées, omises ou manquantes.

Le détecteur peut défailir s'il ne détecte pas les intrusions dont les événements ont été capturés, s'il identifie une activité normale comme intrusive ou s'il n'identifie pas correctement l'attaque détectée.

Bien évidemment, l'arbre de fautes de l'annexe A peut être développé davantage, par exemple, le déni de service dans la branche de la sonde peut être étendu pour inclure les attaques par déni de service distribué ou non, etc. Par manque d'espace, nous nous restreignons dans ce manuscrit à ce niveau d'abstraction.

Table 3 : AMDE générique des IDS

<i>Composant</i>	<i>Mode de défaillance</i>	<i>Cause</i>	<i>Effet</i>
Sonde/Capteur	Non capture des événements intrusifs	<ul style="list-style-type: none"> • Événements malicieux hors de la portée de la sonde • Événements masqués 	Intrusion non vue
		Suppression d'événements	Intrusion non vue ou vue partiellement
Préprocesseur	Suppression d'informations utiles	<ul style="list-style-type: none"> • Format non approprié • Information insuffisante de la part de la sonde 	Défaillance du détecteur
Détecteur	Non-détection des événements intrusifs capturés	<ul style="list-style-type: none"> • Défaillance du préprocesseur • Défaillance de l'algorithme de détection 	Faux négatif
	Événements non intrusifs considérés comme intrusifs	Défaillance de l'algorithme de détection	Faux positif
	Mauvaise identification		Rapport incorrect
Générateur d'alerte	Alerte non générée	Mauvaise configuration	Faux négatif

Notons par ailleurs que l'exploitation de l'AdF est basée sur le calcul des coupures minimales (*minimal cut sets*), c'est-à-dire l'ensemble minimal des événements (nœuds) qui, s'ils se produisent, peuvent conduire à l'événement redouté (racine).

En notation algébrique on a :

$$\text{Défaillance IDS} = A+B+C+D$$

A = Défaillance dans la capture des événements

B = Défaillance du préprocesseur

C = Défaillance du détecteur

D = Défaillance du générateur d'alertes

En substituant les portes du niveau supérieur par le niveau en dessous, on obtient la formule :

$$\text{Défaillance IDS} = (A3.A6) + (A3.E1) + (A3.E2) + (E3.E4) + (E5.E6) + A5 + (B1.B2) + (B1.B3) + E7 + E8 + E9 + E10 + E11 + E12 + D1 + D2$$

La coupure minimale est donc :

$$(A3.A6), (A3.E1), (A3.E2), (E3.E4), (E5.E6), A5, (B1.B2), (B1.B3), E7, E8, E9, E10, E11, E12, D1, D2$$

Une simple analyse des résultats précédents (en particulier, la coupure minimale) indique qu'une attaque peut passer inaperçue seulement en cas de défaillance d'un ou deux composant au plus. Ceci confirme la nécessité de diversifier les techniques de capture et de détection ou même la combinaison et le déploiement de plusieurs IDS sur un même système ou réseau et d'utiliser les techniques de tolérance aux intrusions dans les IDS eux-mêmes. En fait, les conclusions tirées de cette analyse sont abstraites et

déjà connues car l'analyse est portée sur un arbre de faute générique. Cependant, en prenant l'arbre de faute détaillée d'un IDS particulier on peut en sortir des conclusions plus fines.

Il est important de rappeler qu'il s'agit ici d'une évaluation qualitative dont le but est d'identifier et de visualiser les combinaisons d'événements qui peuvent mener à la défaillance de l'IDS. Pour une évaluation quantitative, ce travail peut bien évidemment être enrichi par les probabilités d'occurrence des événements basiques (par exemple, en exploitant des statistiques fournies par les réseaux de pots de miel à grande échelle).

Dans les sections précédentes, nous avons identifié les limites des évaluations existantes des IDS et nous avons proposé notre méthodologie d'évaluation comme une première étape pour pallier ces limitations. Dans la section suivante, nous améliorons notre processus d'évaluation en proposant une nouvelle classification des attaques, que nous avons identifiée comme nécessaire à l'évaluation des IDS.

III. Caractérisation des attaques élémentaires : classification des attaques

Le nombre et la complexité des attaques ont connu une augmentation significative au cours des dernières années. Ceci pose de sérieux problèmes pour les évaluateurs des IDS. En effet, comment tester efficacement et avoir la certitude (prouver) que l'IDS se comporte correctement (par exemple, génération d'une alarme pour toute tentative d'intrusion, pas de fausse alarme, etc.) pour toutes les attaques existantes voir inconnues ?

Puisqu'il est impossible de tester les IDS vis-à-vis de toutes les attaques, il est indispensable de trouver une manière de sélectionner un ensemble des cas de test représentatifs.

Une solution qui peut paraître triviale consiste à construire des classifications pertinentes et représentatives de toutes les attaques. L'idée est basée sur le concept de *classe d'équivalence* bien connu dans le domaine du test de logiciel. Il consiste de réduire considérablement les cas possibles en construisant des classes d'attaques de telle manière qu'un scénario de test ne prendra qu'un élément de chaque classe. Cette technique part du principe que n'importe quelle instance d'attaque d'une classe donnée produira les mêmes effets, et donc générera les mêmes résultats.

Pour traiter ce problème, il nous semble nécessaire de commencer par une analyse approfondie des classifications existantes, ceci est l'objet de la section suivante.

III.1. Analyse des classifications d'attaques existantes

Le but de cette section est de dresser un état de l'art des taxonomies existantes, mais surtout de voir (cf. section suivante) si elles peuvent être pertinentes pour l'évaluation des IDS.

Commençons par la taxonomie de Bishop {Bishop99} ; même si celle-ci concerne les vulnérabilités plutôt que les attaques, il peut être intéressant de regarder les attributs (également appelés axes dans cette taxonomie) qu'elle considère : *nature de la faille* (ex. débordement de tampon), *phase de l'introduction de la vulnérabilité* (ex. pendant l'étape de conception ou d'implémentation), *domaine d'exploitation* (c'est-à-dire comment l'exploiter), *domaine des effets* (ce qui est affecté), *nombre minimum des composants* nécessaires à l'exploitation de cette vulnérabilité et *source* de son identification (le site ou la liste de diffusion où la vulnérabilité a été publiée).

Kumar a proposé une classification des attaques selon quatre attributs du schéma ou de la signature de l'attaque : *existence*, *séquence*, *intervalle* et *durée* {Kumar95}.

La taxonomie de Hansman considère quatre dimensions reliées aux attaques : ce qu'il appelle le *vecteur* ou le type (c'est-à-dire le moyen utilisé par l'attaquant pour arriver à ses fins, comme les virus, les vers, le déni de service), la *cible* (ex. système d'exploitation, protocole réseau), les *effets* de l'attaque ainsi que la *vulnérabilité exploitée* {Hansmann03}.

Un autre travail intéressant est celui de Lindqvist et Jonsson {Lindqvist97} qui étendent la taxonomie de Newman et Parker {Neumann89}. Ces derniers considèrent une seule dimension, la *technique*, tandis que Lindqvist et Jonsson ajoutent le *résultat* comme dimension supplémentaire. Cette classification s'inscrivait dans le cadre d'expériences menées par des utilisateurs internes (étudiants d'une classe d'informatique) afin d'améliorer les capacités de détection d'IDS qui utilisent le filtrage par reconnaissance de forme (*pattern matching*, en anglais). D'ores et déjà, on remarque que ce travail ne considère que des attaques lancées par des étudiants. On peut ainsi constater qu'elle ignore une grande

partie de l'espace des attaques, notamment des attaques plus sophistiquées (non imaginées ou non accessibles aux étudiants de cette classe).

Weber a présenté une taxonomie basée sur trois dimensions : le *niveau de privilège requis* pour mener l'attaque, le *moyen utilisé par l'attaquant* (ex. exploitation d'un bug logiciel) ainsi que *l'effet souhaité* (ex. déni de service) {Weber98}.

La taxonomie de la DARPA est en fait une version réduite de celle de Weber. Elle ne considère que *l'effet* de l'attaque comme dimension. Les attaques sont divisées en cinq catégories : "distant vers local" (ou R2L pour *Remote to Local*), "utilisateur vers super-utilisateur" (ou U2R pour *User to Root*), "sondeur" (*scan*) et "déni de service" {Kendall99}, {Lippmann00a}. Là encore, on peut remarquer que cette classification considère des niveaux différents d'abstraction, ce qui pose des problèmes, notamment l'exclusion mutuelle des classes résultantes.

À la différence des classifications déjà citées, la taxonomie de Howard est centrée sur le processus de l'attaque, plutôt que sur l'attaque elle-même {Howard98}. Elle tient compte de *l'attaquant* (qui est-il ?), de *l'outil* qu'il a utilisé, de la *vulnérabilité* exploitée, de *l'accès obtenu*, des *résultats* de l'attaque (c'est-à-dire divulgation, altération) ainsi que ses *objectifs* (c'est-à-dire obtenir ou détruire une information).

Un autre travail intéressant {Killourhy04} présente une taxonomie prenant un point de vue défensif. Le but était de fournir des informations pour aider les administrateurs à mieux défendre leurs systèmes. Les attaques ont ainsi été classifiées selon leurs manifestations (opérations visibles) telles qu'elles sont vues par des HIDS (*Anomaly Host-based IDS*, en anglais). Les quatre dimensions de cette taxonomie sont :

1- *signes extérieurs* : il s'agit d'appels systèmes qui apparaissent suite à l'exécution de l'attaque, mais qui n'apparaissent jamais dans les opérations normales (c'est-à-dire dans les activités non intrusives) ;

2- *séquence minimale* : c'est la plus petite séquence qui apparaît dans l'attaque, mais qui n'apparaît jamais dans des opérations normales ;

3- *séquence dormante* : c'est une séquence qui correspond (partiellement) à une sous-séquence d'opérations normales ;

4- *séquence normale* : c'est une séquence dans l'attaque qui ne se distingue pas des activités non intrusives.

Enfin, regardons de plus près l'importante taxonomie d'Alessandri {Alessandri04}. Celle-ci a été élaborée à des fins d'analyse des modèles d'IDS. Au lieu de catégoriser directement les attaques, elle classifie plutôt toutes les activités (de manière plus globale) qui peuvent être pertinentes pour l'IDS. Une évaluation analytique a été ensuite établie pour déterminer les capacités de détection de l'IDS vis-à-vis de telle ou telle classe d'attaques.

Plus concrètement, le modèle correspondant à cette classification fait la différence entre les caractéristiques dynamiques et les caractéristiques statiques d'une activité observable par l'IDS. Les activités statiques sont divisées en caractéristiques reliées aux objets-interfaces⁶ et en celles reliées aux objets affectés (corrompus) par l'attaque. Les caractéristiques dynamiques sont développées selon trois critères : caractéristiques de la communication (ex. unidirectionnelle, bidirectionnelle), méthode d'invocation (ex. création, suppression, lecture) ainsi que d'autres attributs additionnels qualifiés de mineurs (ex. l'attaque provient de plusieurs origines ou elle contient des événements répétitifs).

L'attaque, quant à elle, est décrite selon cinq paramètres : *l'objet-interface*, *l'objet affecté*, la *communication*, la *méthode d'invocation* ainsi que d'autres *attributs mineurs*. Au total, cette taxonomie contient 25 objets-interfaces, 10 objets affectés, 3 caractéristiques reliées à la communication, 5 méthodes d'invocations ainsi que 4 attributs additionnels mineurs.

Il faut constater que chacune des classifications existantes a été développée dans un but particulier (par exemple, comprendre les vulnérabilités pour renforcer les mesures correctives et défensives, appréhender les processus d'attaque ainsi que le comportement des attaquants, etc.). Il en résulte que les attributs identifiés dans une étude ne sont pas forcément pertinents pour une autre ayant un objectif différent.

6 Un objet interface est un objet (ex. fichier, processus) qui contient une vulnérabilité ou qui propose une fonctionnalité qui a servi pour attaquer d'autres objets (entités du système).

Avant de présenter notre classification, la section suivante va d'abord discuter les limites des classifications présentées et va analyser les attributs qu'elles proposent afin de ne retenir que les plus pertinents pour l'évaluation des IDS.

III.2. Discussion

Les différentes taxonomies existantes adoptent différents points de vue, et sont basées sur des attributs liés aux attaques ou aux vulnérabilités. Même s'il est impossible de citer dans ce mémoire toutes les taxonomies existantes, on peut globalement identifier les attributs les plus importants :

- *type de l'attaque* : virus, vers, cheval de Troie, déni de service, etc. ;
- *technique de détection de l'attaque* : approche statistique, filtrage, reconnaissance de motif, etc.
- *signature de l'attaque* : motif (*pattern*) ou séquence de motifs observés ;
- *outil utilisé par l'attaquant* : boîte à outils (*toolkit*), script, commande utilisateur, etc. ;
- *cible de l'attaque* : système d'exploitation, protocole réseau, application, service, ... ;
- *résultat de l'attaque* : modification illicite ou divulgation d'informations, déni de service, ... ;
- *accès visé par l'attaque* : accès en super-utilisateur, accès en utilisateur normal ;
- *préconditions de l'attaque* : existence de versions particulières d'un certain logiciel, ... ;
- *vulnérabilité exploitée par l'attaque* : débordement de mémoire, mauvais choix de mot de passe, mauvaise configuration, etc. ;
- *objectif de l'attaque* : gain financier, terrorisme, autosatisfaction, etc. ;
- *localisation de l'origine de l'attaque* : interne, externe ;
- *propriété de sécurité violée ou visée par l'attaque* : confidentialité, intégrité, disponibilité.

On peut noter que la plupart des travaux existants souffrent d'un manque de clarté dans la distinction entre les attributs, et donc entre les attaques. Par exemple, certaines classifications regroupent le débordement de tampon et le déni de service sous le même attribut ; ce choix nous semble abusif car une attaque qui exploite un débordement de tampon peut aussi causer un déni de service.

Par ailleurs, nous constatons que la plupart des classifications sont centrées sur l'attaquant, c'est-à-dire adoptent le point de vue de l'attaquant (*attacker-centric*, en anglais). Or ce type d'approches souvent ignore (ou masque) certaines caractéristiques importantes des attaques, telles qu'elles sont vues par l'IDS ou les administrateurs système, alors que ces aspects sont importants dans notre contexte.

À l'inverse, la classification d'Alessandri a été principalement créée pour l'analyse des modèles d'IDS {Alessandri04}. Elle considère plus de détails reliés à l'attaque en termes de caractérisation des IDS, ce qui la rend plus pertinente pour l'évaluation et le test des IDS. Néanmoins, elle présente quelques limites. Tout d'abord, elle s'est centrée sur les manifestations des activités intrusives qui peuvent être observables par l'IDS, mais elle ignore certains attributs intéressants pour l'évaluation des IDS, notamment les privilèges requis ou obtenus, les conséquences, etc.

De plus, nous trouvons son niveau de dimensionnement très fin, au point que le niveau de détail atteint n'est pas nécessairement très utile pour le test des IDS. Pour ne prendre qu'un exemple simple, analysons la dimension "*object-interface*" ; celle-ci contient 24 types dont cinq sont directement reliés à la couche "application" : *App. layer-connectionless* ; *App. Layer single-connection single-transaction* ; *App. layer single-connection multiple-transaction* ; *App. layer multiple-connection single-transaction* ; et *App. layer-multiple-connection multiple-transaction*. Avec ce niveau fin de granularité, il n'est pas rare de trouver des classes contenant seulement une ou deux attaques, alors que le résultat d'un point de vue de l'analyse de l'IDS est pratiquement le même.

Par ailleurs, les combinaisons des différents cas possibles (compte tenu de cette classification très fine) conduit à 9600 cas de test, alors que, par exemple, la fusion des classes reliées au niveau de l'application, permet de réduire ce nombre à 8000. On peut donc obtenir un gain considérable de temps, sans pour autant pénaliser la procédure de test. En effet, le niveau de détail caché lors du regroupement (proposé dans l'exemple) peut être investi ultérieurement à travers une analyse complémentaire, si l'IDS sous test est sensible à certains types de communication au niveau applicatif.

Pour résumer cette discussion, force est de constater que les taxonomies existantes ne sont pas réellement adaptées pour l'évaluation des IDS. Les raisons peuvent globalement être résumées dans les points suivants :

- dans leur majorité, elles considèrent la vision de l'attaquant et non celle de l'IDS ; il n'est donc pas étonnant que les attributs résultants soient moins pertinents pour le test des IDS ;

- parfois, la définition des attributs est quelque peu ambiguë voir incohérente ; ceci peut poser des problèmes d'exclusion mutuelle, et donc de classification ;
- le nombre de classes résultantes est parfois très grand, sans que la complexité qui en résulte soit justifiée par une efficacité accrue du test des IDS ;
- ces classifications ne sont malheureusement pas accompagnées de schéma de sélection et génération des cas de test.

Dans la section suivante, nous proposons une nouvelle classification qui cherche à dépasser ces limites.

III.3. Nouvelle classification

Nous nous baserons sur les attributs que nous avons identifiés dans la section précédente, en éliminant ceux qui sont ambigus ou qui ne sont pas pertinents pour l'évaluation des IDS. Les attributs retenus seront accompagnés par une définition claire.

La définition d'une taxonomie systématique devrait passer, en réalité, par une identification judicieuse des principaux objectifs à respecter.

Tout d'abord, la classification, au même titre que la sélection des cas de test, doit être réfléchie et bien structurée. En effet, dans une sélection plus ou moins "aléatoire" des cas de test, les évaluateurs testent souvent leurs systèmes de manière ad hoc en utilisant quelques scripts disponibles sur Internet ou dans des listes de diffusion. Néanmoins, les scripts récupérés ne couvrent pas certains types d'attaques critiques et ne reflètent pas une distribution cohérente des attaques.

En outre, l'expression des résultats de l'évaluation en termes de *classes* d'attaques peut contribuer certainement à une meilleure compréhension de l'évaluation ainsi qu'à une représentation et interprétation plus précises de ses résultats. En effet, il est plus intéressant de dire que l'IDS est faible (ou robuste) vis-à-vis de la détection de tel ou tel type d'attaques. Au contraire, lorsqu'on exprime les résultats en distinguant chaque attaque prise individuellement (et non de manière générique à travers les classes d'attaques), les conclusions peuvent être interprétées de manière biaisée.

Ceci étant, les classes résultantes ainsi que le processus de classification doivent respecter autant que possible les propriétés suivantes :

1. *complétude* (c'est-à-dire *exhaustivité*) : un schéma de catégorisation doit tenir compte de toutes les attaques possibles (connues et inconnues) ;
2. *extensibilité* : quand de nouvelles attaques apparaissent, le schéma de catégorisation doit permettre de les classer.
3. *clarté des critères* : le schéma et les règles de classification doivent être bien établies de manière à ce qu'une attaque puisse être classifiée en prenant une et une seule classe à partir de chaque dimension ;
4. *répétitivité* : la ré-application du processus de classification doit toujours produire les mêmes résultats ; autrement dit, si on répète les étapes suivies pour la classification d'une certaine attaque, on doit toujours la placer dans la même catégorie ;
5. *conformité avec les standards* et terminologies existants, notamment avec les bases de données et dictionnaires des vulnérabilités comme CVE {Cve08} et OSVDB {Osvdb08}, qui sont actuellement largement utilisés ;
6. *exclusion mutuelle* : être sûr qu'une attaque ne fait pas partie de deux catégories différentes, une dimension n'aura donc que des classes mutuellement exclusives ;

Dans le cas de notre étude, il faudrait, en plus, garder une vision "évaluateur" (et non "attaquant") tout au long du processus de classification. Ceci va considérablement influencer la procédure de sélection de cas de tests nécessaires.

Commençons par analyser les attributs mentionnés dans la section précédente, afin de n'en retenir que les plus pertinents d'un point de vue "évaluateur", ceux qui sont invisibles par l'IDS ou dénués de sens étant donc écartés.

Par exemple, des dimensions comme l'objectif de l'attaquant, ne seront pas considérés dans notre classification, d'autant plus qu'il est à la fois difficile et inutile dans notre contexte d'imaginer l'intention de l'attaquant. Dans notre vision, toute tentative d'attaque est considérée comme une menace, quelque soit l'objectif visé (terrorisme, vandalisme, vol ou autre).

Dans le même sens, des dimensions comme le résultat de l'attaque ou la propriété de sécurité sont également peu pertinents dans notre étude. En effet, une fois que l'attaquant prend la main dans un système (en particulier s'il obtient l'accès *root*), il peut généralement modifier, détruire ou divulguer les informations, et donc porter atteinte à la fois aux propriétés de confidentialité, d'intégrité et de disponibilité. Par ailleurs, nous estimons que les dimensions "type" et "technique de détection" ne servent pas à définir une catégorisation claire.

Après avoir écarté toutes les dimensions non-pertinentes pour notre étude et adapté celles qui peuvent être utiles, nous avons abouti à la classification de la figure 4. Notre classification repose sur cinq dimensions. Ces dimensions sont sélectionnées de manière à couvrir les sources, les cibles et les manifestations des attaques, informations qui nous semblent nécessaires et suffisantes pour le test et l'évaluation des IDS. Nous définissons ces dimensions comme suit :

- **Source** : indique l'endroit d'où l'attaque a été lancée. Elle possède deux classes : locale et distante.
- **Privilège obtenu** : nous distinguons cinq classes de privilèges visés par l'attaquant, les classes "root" et "utilisateur" signifient respectivement que l'attaquant a réussi à obtenir l'accès "root/administrateur" ou "utilisateur" ; la classe "système" qui permet l'exécution de processus avec les privilèges "systèmes" ; la quatrième classe "variable" identifie les attaques qui fournissent l'accès en fonction des privilèges de l'utilisateur de l'application vulnérable exploitée. La classe "aucun" couvre les attaques qui n'ont besoin d'aucun privilège d'accès au système, comme les attaques de reconnaissance (*scans*).
- **Vulnérabilité** : du point de vue de l'évaluateur, il est intéressant de cibler le système de test le plus pertinent, de bien paramétrer la plateforme de test, d'exprimer la relation entre les attaques et les vulnérabilités exploitées ; ceci va en particulier aider à choisir (lors de la phase de test) les attaques qui peuvent exploiter ces vulnérabilités (et qui sont d'ailleurs répertoriées et disponibles dans des bases de données standardisées de vulnérabilités), mais aussi à identifier les failles du système pour une éventuelle correction.
- **Porteur** ou moyen par lequel l'attaque est lancée : il peut s'agir du trafic réseau ou d'action exécutée directement sur la machine cible et qui n'apparaît donc pas sur l'interface réseau.
- **Cible** : ce peut être la mémoire, le système d'exploitation, la pile réseau, le système de fichier ou un processus.

Remarquons que contrairement aux classifications existantes, notre taxonomie tient compte, non seulement des caractéristiques observables de l'attaque (comme c'est le cas des classifications orientée IDS {Alessandri04} ou orientée défense {Killourhy04}), mais aussi des aspects opérationnels, qui sont importants pour l'évaluateur.

En effet, la classification que nous proposons fournit les informations essentielles pour la génération des attaques et l'analyse des cas de test. Par exemple, la dimension "source" donne une idée sur l'endroit d'où l'attaque doit être générée pour le test ; de même, la dimension "vulnérabilité" donne une information sur la configuration à avoir (ou à l'inverse, à éviter) pour le test. Dans le même sens, la sévérité des attaques est implicitement décrite à partir de la dimension "privilège".

Il est également important de noter que notre classification respecte les objectifs (règles de bonne pratique) déjà identifiés ci-dessus :

- Tout d'abord, les cinq attributs que nous proposons sont choisis de façon à avoir une caractérisation exhaustive, couvrant différentes facettes des attaques. Ainsi, n'importe quelle attaque peut être caractérisée, c.-à-d. classifiée [propriété 1: *complétude*].
- Par ailleurs, l'extensibilité des dimensions "cible" et "moyen" permet de classifier les nouvelles attaques (ex. notamment celles qui utilisent de nouveaux moyens ou visent de nouvelles cibles) [propriété 2 : *extensibilité*].
- De plus, les définitions des dimensions que nous proposons aident amplement à déterminer (facilement) la classe de chaque attribut qui caractérise l'attaque à inclure dans l'ensemble des cas de test [propriété 3 : *clarté des critères*].
- Cette clarté des critères aide également à placer une certaine attaque dans la même catégorie si on réapplique le schéma de classification [propriété 4: *répétitivité*].
- En outre, la dimension "vulnérabilité" peut établir un lien direct entre l'attaque et une ou plusieurs entrées dans les bases des données standardisées des vulnérabilités (ex. CVE ou OSVDB) [propriété 5 : *conformité aux standards*].

- Enfin, puisque les attributs de notre classification sont mutuellement exclusifs, une attaque ne peut, a priori, faire partie de deux catégories différentes [propriété 6 : exclusion mutuelle] ; cette propriété sera davantage démontrée lors de la classification d'attaques réelles existantes.

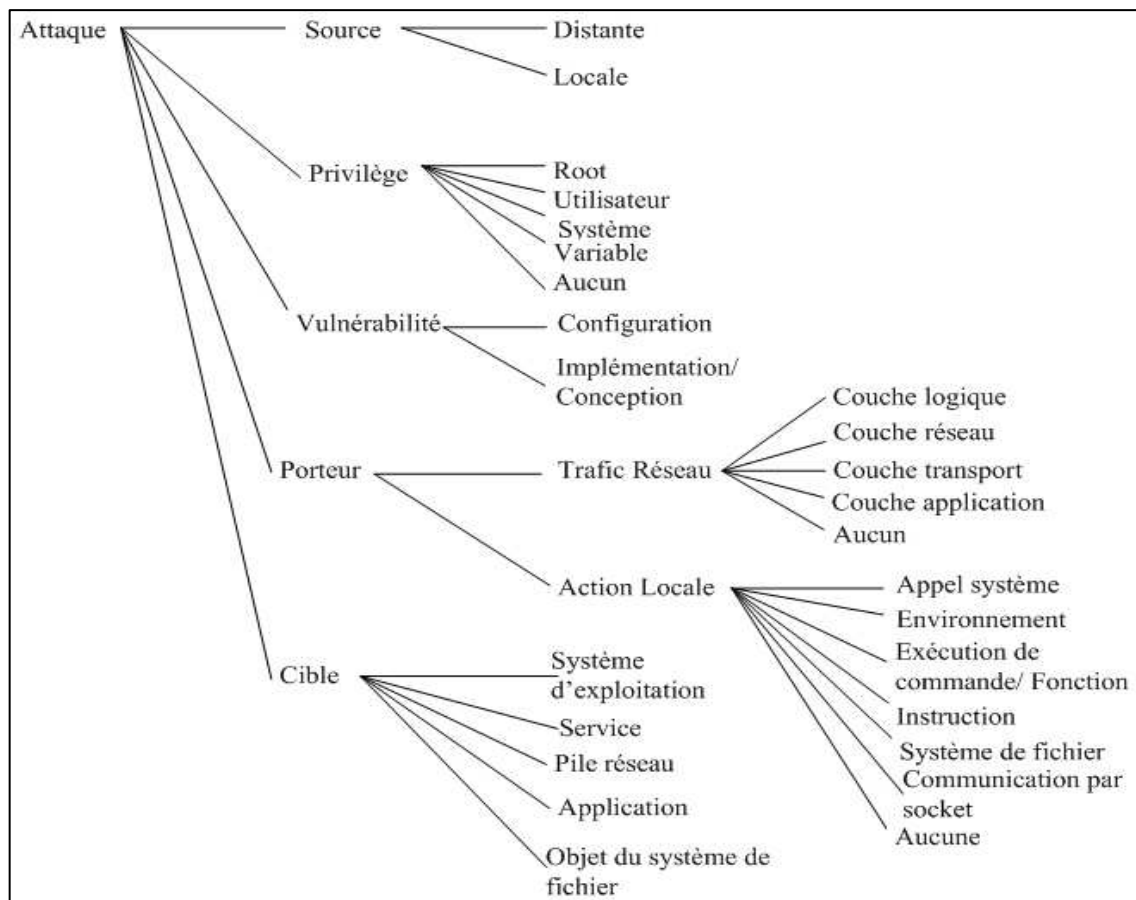


Figure 4 : Nouveau schéma de classification d'attaque.

III.4. Schéma pour la sélection des cas de test

Dans cette section nous présentons un schéma de sélection de cas de test basé sur l'arbre de classification (CTM pour *Classification-Tree Method*) qui a été développé par Grochtmann et Grimm dans le domaine du génie logiciel [Grochtmann95]. Comme son nom l'indique, cette méthode représente graphiquement les partitions du domaine d'entrée sous forme d'arbre. Dans notre cas, le but est de pouvoir former des cas de test en combinant des classes appartenant à différentes dimensions.

Dans une première étape, le domaine des entrées du test est d'abord considéré selon divers aspects ; pour chaque aspect, des classifications complètes et disjointes sont formées. Les classes résultantes sont, à leur tour, divisées en sous-classes. Dans la deuxième étape, une grille est dressée au-dessous de l'arbre. Chaque colonne de la grille contient les feuilles de l'arbre de classification (figure 5).

Un cas de test correspond en fait à une sélection d'une seule classe-fille de chaque attribut/dimension de niveau supérieur ; en d'autres termes, chaque ligne de la grille indique un cas de test distinct. Néanmoins, tous les cas de test possibles théoriquement par cette méthode ne sont pas forcément valides ou intéressants. La personne qui planifie le test doit donc identifier les cas valides et éliminer les autres, en se basant notamment sur les contraintes déjà définies ainsi que d'autres informations concernant le système comme l'explique l'exemple de la section suivante.

L'arbre de classification, présente plusieurs avantages. Tout d'abord, l'identification de tous les cas possibles ainsi que la sélection des cas de test pertinents se fait de manière systématique, ce qui facilite sa gestion et aide à réduire ou éliminer certaines erreurs. De plus, sa représentation graphique améliore la visualisation et facilite la communication entre les personnes qui font la spécification, celles qui s'occupent du développement et celles qui gèrent les tests.

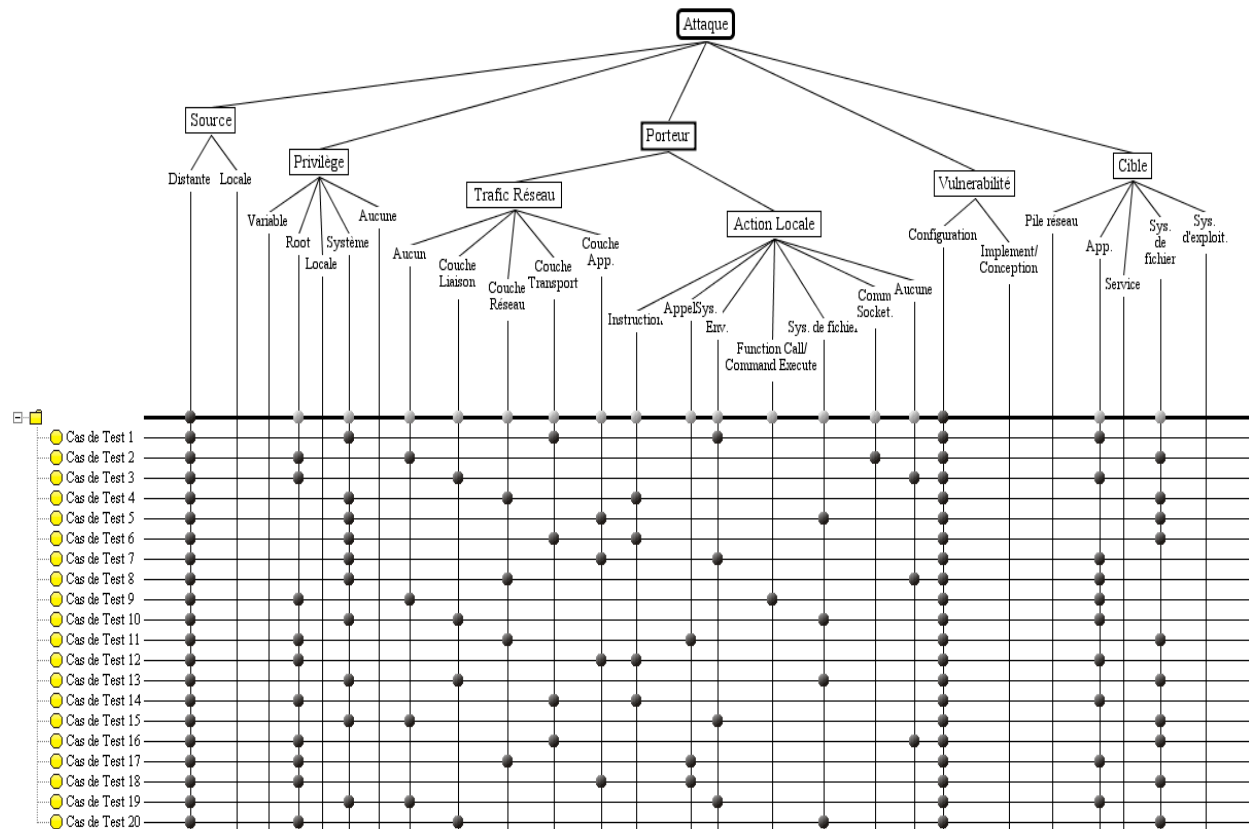


Figure 5 : Schéma de sélection d'attaque basée sur l'arbre de classification.

III.4.2. Exemple de sélection des cas de test

Une fois tous les cas de test possibles énumérés, il faut identifier les cas de test pertinents. Pour cela, nous avons utilisé l'outil CTE, pour *Classification Tree Editor* [Cte08]. Cet outil permet, entre autres, l'automatisation de la génération et l'identification des cas de test.

Ainsi, à partir de notre schéma de classification et en utilisant la CTM, l'outil CTE génère toutes les combinaisons possibles des sous-classes. Ces combinaisons représentent les cas de tests d'attaques possibles.

Afin de couvrir l'espace des attaques, le nombre de combinaisons possibles sans considérer les contraintes est 3500 (dont 3400 combinaisons valides), au lieu de 9600 si on utilise la classification d'Alessandri [Alessandri04].

L'outil CTE permet l'application des contraintes sur l'arbre de classification. Cela aide à réduire davantage, à regrouper ou à réordonner les cas de test afin de n'en retenir que les plus pertinents pour l'évaluation en cours. Plus précisément, CTE offre un formalisme simple et puissant pour l'expression des contraintes en combinant des règles contenant des sous règles entre parenthèses (sous formes de prédicats), des connecteurs interpropositionnels tels que et (*), ou (+), non (NOT), etc.

Par exemple, la contrainte suivante :

*Distant * (root + système) * Vul_configuration * Trafic réseau * (Sys. de fichier + App)*

génère des cas de test qui fournissent des accès "root" OU "système", ET qui concernent les attaques distantes pouvant exploiter des vulnérabilités introduites lors de la configuration, ET qui sont visibles sur le trafic réseau, ET qui visent des applications OU le système de fichiers.

Comme indiqué dans la figure 5, l'application de cette contrainte réduit les cas de test de 3400 à 20 seulement. Par exemple, le premier cas de test concerne les attaques qui :

- sont lancées à distance,
- fournissent des accès "système",
- sont visibles sur le trafic réseau au niveau transport,
- modifient les variables de l'environnement,
- exploitent des vulnérabilités introduites lors de la configuration, et
- visent des applications.

Dans cette section, nous avons présenté une classification qui caractérise chacune des attaques au niveau élémentaire. Or, dans les cas réels, le test de l'IDS devrait tenir compte de scénarios qui peuvent être composées de plusieurs attaques élémentaires. La section suivante s'intéresse aux scénarios d'attaques.

IV. Caractérisation et modélisation des scénarios d'attaque

Ces dernières années ont vu une augmentation massive du nombre des virus, vers et autres programmes malveillants (malicieux). Et même si les rapports se contredisent parfois, on parle d'une multiplication par cinq en 2007 par rapport à l'année précédente. Pour ne citer qu'un exemple représentatif, le très sérieux projet *AV-Test* (15 ans d'expérience dans l'analyse des programmes malveillants) a répertorié en 2007, 5,5 millions d'échantillons de programmes malveillants nouveaux, contre moins d'un million en 2006 [Avtest08]. Ce nombre est non-cumulatif, il ne tient pas compte des programmes malveillants précédemment détectés.

Cette analyse est alarmante pour ce qui concerne la détection d'intrusions. En effet, une légère modification d'un programme malveillant connu suffit pour tromper la plupart des mécanismes de détection à base de signature. De plus, le processus de génération de signatures est généralement fastidieux, même pour des variantes de malicieux connus, alors qu'il est facile de le modifier automatiquement. Il n'est donc pas étonnant de voir de nouvelles mutations d'un cheval de Troie chaque jour, voire parfois chaque heure.

De toute façon, la croissance actuelle du nombre de malicieux ne peut que provoquer l'effondrement des performances des outils de sécurité qui ont besoin d'une signature spécifique du malicieux, que ce soit un nouveau programme malveillant ou une variante d'un malicieux connu. Il est donc important de trouver une solution efficace pour ce problème, plus que jamais d'actualité. Pour aider à le résoudre, le travail présenté dans cette section propose de modéliser les processus d'attaque sous forme de scénarios composés de séquences de primitives d'exécution. La définition de ce modèle repose sur l'analyse d'incidents de sécurité résultant aussi bien d'outils d'attaque automatiques que d'attaques interactives. Ce modèle peut alors servir à générer automatiquement des scénarios d'attaques représentatifs des attaques réelles, et ces scénarios peuvent être utilisés pour tester les outils de détection d'intrusion (IDS) et en évaluer l'efficacité.

IV.1. Approche

Afin de traiter cette question de la multiplication des variantes d'attaques, et contrairement à d'autres travaux qui visent à générer une signature pour chaque variante, notre approche vise plutôt à décrire les attaques par des caractéristiques qui soient moins sensibles au polymorphisme des variantes.

Pour cela, nous avons analysé les modèles d'attaques {Cheung03}, {Dacier94}, {Dahl06}, {Schneier99}, {Templeton00} qui décrivent le comportement des utilisateurs et programmes (potentiellement malveillants) accédant à et/ou s'exécutant sur un système. Mais nous avons constaté que ces modèles sont généralement spécifiques de l'environnement d'exécution, et nécessitent donc une connaissance précise et détaillée de l'architecture, de la topologie et des vulnérabilités du réseau et du système considérés. De plus, ces modèles se basent essentiellement sur les vulnérabilités connues et ignorent les attaques susceptibles d'exploiter des vulnérabilités encore inconnues.

Dans notre contexte, ceci constituerait une limite sérieuse, dans la mesure où la robustesse des IDS dépend également des vulnérabilités inconnues et des nouvelles attaques. De plus, le modèle qui répondrait à nos attentes devrait être facilement adaptable et extensible notamment lors de l'ajout ou de la suppression d'un utilisateur ou d'une machine, lors de l'installation ou la mise à jour d'un logiciel, ou encore lors de l'application de rustines (*patches*) pour corriger ses vulnérabilités.

Pour pallier ces limites, nous proposons dans ce travail un modèle suffisamment abstrait pour couvrir un maximum de classes ou de types d'attaques, et qui soit le plus possible indépendant de l'environnement.

Pour établir un tel modèle, il s'avère nécessaire d'analyser un nombre suffisant de données sur les attaques réelles, ce qui constitue en soi un problème car les données disponibles pour la communauté scientifique sont limitées, voire parfois biaisées et non représentatives.

C'est pour cela que nous avons basé notre analyse préliminaire sur les attaques de type virus et vers les plus répandues. En effet, étant donné que les vers sont autonomes, ils doivent comporter toutes les étapes d'un processus d'attaque. De plus, les virus comme les vers peuvent être vus comme une classe d'attaques automatiques développées par des attaquants habiles ; cela peut donc aider à comprendre comment les attaques interactives peuvent être menées.

Dans la section suivante, nous présentons le résultat de l'analyse d'environ 40 virus, vers, chevaux de Troie ainsi que d'autres incidents liés à des attaques, dans le but d'identifier les types et séquences d'actions qu'un attaquant exécute. Une description plus détaillée de cette analyse peut être trouvée dans {Gadelrab08a} ou dans l'annexe A.

IV.2. Analyse de maliciels

Nous avons analysé les primitives d'exécution des 39 maliciels de la liste CME (*Mitre's Common Malware Enumeration list*) {Cme08}, qui sont représentatifs des attaques les plus dangereuses et les plus répandues. Nous avons également utilisé d'autres données intéressantes disponibles sur des sites spécialisés comme <http://research.eeye.com>, et <http://www.viruslist.com>.

Le premier résultat surprenant que nous avons pu constater est que, malgré la diversité de ces maliciels, les étapes suivies pour ces attaques peuvent être classées en seulement 8 primitives. Nous avons identifié chaque primitive par un symbole, comme indiqué ci-dessous :

- *R*: Reconnaissance
- *VB*: Fouille des machines ou des réseaux victimes (*Victim Browsing*)
- *EP*: Exécution de programme (*Execute Program*)
- *GA*: Gain d'accès (*Gain Access*)
- *IMC*: Implantation de code malveillant (*Implant Malicious Code*)
- *CDI*: Compromission de l'intégrité (*Compromise Data Integrity*)
- *DoS*: Dénégation de service (*Denial of Service*)
- *HT*: Effacement des traces (*Hide Traces*)

Notons qu'au lieu d'analyser les détails des commandes et des instructions de bas niveau, nous nous sommes plutôt intéressés au processus d'attaque dans sa globalité. Cela nous permet d'identifier les primitives communes aux différents types d'attaques et être le plus indépendant possible de la plateforme ou de l'environnement.

D'ailleurs, pour ne pas biaiser l'étude et afin d'avoir des résultats plus généraux, nous prêtons peu d'attention à l'aspect propagation, caractéristique plutôt spécifique aux vers. En réalité, nous considérons l'étape de propagation comme un gain d'accès (*GA*) ou une implantation de code malveillant (*IMC*).

De plus, étant donné que plusieurs étapes d'attaques peuvent être considérées, de manière globale, comme une exécution de programme (*EP*), nous préférons les différencier en considérant une relation d'ordre partielle (dénnoté par $>$), le but étant ainsi de déterminer (et ne considérer que) la primitive qui exprime et reflète le mieux l'étape de l'attaque en cours. Ainsi, nous considérons que :

- $IMC > CDI > EP$
- $HT > CDI > EP$
- $[R/VB] > EP$
- $HT > DoS$

Ces relations peuvent être interprétées de la manière suivante :

- Si l'étape d'attaque exécutée contribue à l'installation d'un code malveillant, elle est classifiée comme *IMC*. Sinon, si elle modifie le système de fichiers, les fichiers de configuration, les clefs d'enregistrement (*Windows registry*), ou les variables d'environnement, on la considère plutôt comme un *CDI*. Autrement, on la considère comme *EP*.
- Si l'étape d'attaque exécutée cache des informations ou bloque l'accès aux informations qui montrent l'existence d'un code malveillant, elle est considérée comme *HT*. Sinon, si elle modifie le système de fichiers, les fichiers de configuration, les clefs d'enregistrement (*Windows registry*), ou les variables d'environnement, sans cacher des informations liées à l'attaque, on la considère comme un *CDI*. Autrement, nous la considérons comme *EP*.
- La recherche à distance des informations reliées aux victimes potentielles est une étape de reconnaissance (*R*). Si l'attaquant cherche des informations localement stockées sur la victime on l'identifie comme une étape d'exploration (*VB*).

- Si l'étape d'attaque conduit à bloquer/arrêter/compromettre l'accès aux services qui fournissent des informations reliées aux activités malveillantes, il s'agit en fait d'une étape d'effacement des traces (HT). Sinon, si le service bloqué/arrêté/compromis ne cache pas des informations sur les activités malveillantes, nous le considérons comme une étape de déni de service (DOS).

Dans la suite de cette section, nous construirons progressivement un modèle de processus d'attaques.

IV.3. Modèle de processus d'attaques

Cette analyse nous a permis de construire le modèle décrit par la figure 6. Le premier résultat surprenant que nous pouvons souligner est que, quelque soit la nature de l'attaque et l'expérience des attaquants, ces derniers suivent généralement le même type de processus (étapes), le niveau de l'attaquant se reflète à travers la sophistication et la finesse de l'attaque, la qualité du code, les effets et les dommages causés, etc. En général, les attaquants expérimentés utilisent des techniques plus avancées et des outils plus personnalisés, alors que les débutants (*script kiddies*) utilisent des exploits et des outils souvent développés par d'autres.

Notre modèle distingue les étapes suivantes (figure 6) :

- *Reconnaissance* : il est logique pour un attaquant de chercher les informations nécessaires sur les victimes potentielles avant de les cibler avec les outils d'attaques les plus appropriés (codes d'exploits, *toolkits*, etc.).
- 2) *Gain d'accès (Gain Access)* : afin d'atteindre leurs objectifs, les attaquants ont généralement besoin d'avoir un accès aux ressources des victimes ; le niveau d'accès requis dépend évidemment de l'attaque. Notons toutefois que certains types d'attaques, comme les attaques en déni de service, n'ont pas, en général, besoin d'accès sur la machine victime.
- 3) *Augmentation de privilèges (Privilege Escalation)* : l'accès obtenu initialement par l'attaquant est parfois insuffisant pour réaliser l'attaque ; dans ce cas, l'attaquant essaie d'augmenter ses privilèges pour avoir plus de pouvoir (par exemple, passer du mode utilisateur au mode administrateur pour pouvoir accéder aux ressources systèmes).
- 4) *Fouille de la machine victime (Victim browsing)* : après avoir acquis suffisamment de privilèges, l'attaquant essaie généralement d'explorer la machine ou le réseau cible (par exemple, en fouillant les fichiers et les répertoires), pour rechercher un compte particulier (comme un compte invité ou un compte ftp anonyme), pour identifier les composants matériels, pour identifier les programmes installés, pour rechercher les hôtes de confiance (typiquement, ceux ayant des certificats installés sur la machine de la victime), etc.
- 5) *Actions principales (Principal Actions)* : comme indiqué dans la figure 6, cette étape peut prendre différentes formes ; par exemple, l'attaquant peut exécuter une attaque en déni de service, installer un code malveillant, compromettre l'intégrité des données ou exécuter un programme.
- 6) *Cacher les traces (Hiding Traces)* : les attaquants les plus expérimentés utilisent généralement cette dernière étape pour effacer leurs traces et rendre ainsi la détection plus difficile.

Il est important de noter que, du point de vue de la détection d'intrusion, le nombre d'étapes qui apparaît dans une certaine session d'un processus d'attaque est arbitraire. En effet, afin d'empêcher la détection, les attaquants peuvent procéder lentement, en plusieurs étapes, sur plusieurs jours, voir même sur plusieurs semaines. Ainsi, quand ils reprennent leur attaque avec les étapes qui suivent, cela pourrait apparaître comme une nouvelle attaque pour l'outil de détection d'intrusion (*IDS*) ; le plus souvent, l'attaquant recommence directement par une augmentation de privilège ou par exécuter des actions intrusives sans reproduire les étapes précédentes (telles que la reconnaissance, la fouille, etc.). Par ailleurs, dans bien des cas, l'attaquant peut être un utilisateur interne qui possède un compte valide et qui a déjà suffisamment d'information et de privilège ; il n'a donc pas besoin de passer par certaines étapes comme la reconnaissance. D'un autre côté, un processus d'attaque peut être interrompu délibérément par une simple décision de l'attaquant, par exemple s'il estime qu'il est difficile de réussir son attaque ou s'il a trouvé une autre cible plus facile ou plus intéressante.

Notons également que même si notre modèle ne tient pas compte directement des attaques dites multi-sauts (*multi-hop attacks*) qui passent par plusieurs victimes, il est tout à fait possible de représenter ces attaques par une concaténation de plusieurs scénarios (c'est-à-dire avec un scénario pour chaque victime).

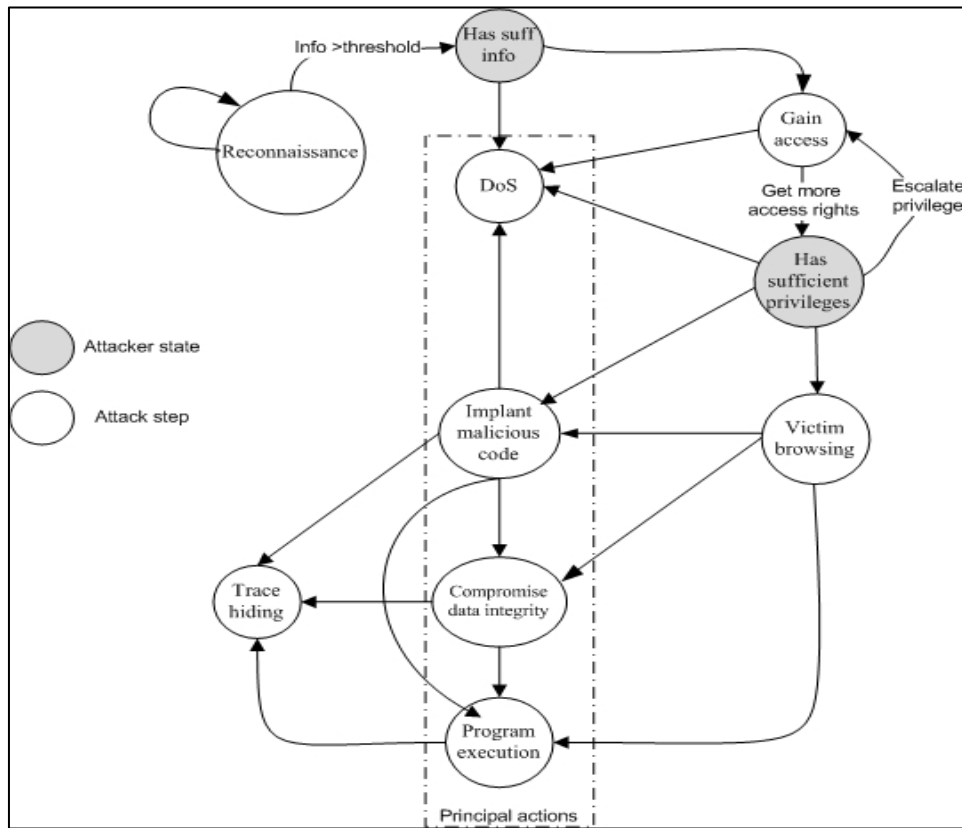


Figure 6. Modèle de processus d'attaque

V. Génération des scénarios d'attaque

Dans notre contexte, c'est-à-dire l'évaluation des IDS, notre modèle de processus d'attaque (figure 6) est simplifié par la machine à états illustré par la figure 7. Ce graphe permet de générer des scénarios d'attaques à un niveau abstrait. En appliquant des contraintes sur les chemins entre les nœuds et la répétition consécutive de la même action (les boucles), nous pouvons trouver un ensemble des scénarios abstraits valides, comme par exemple :

Scén_1 = (R, GA, VB, CDI, EP, HT)
 Scén_2 = (R, DoS)
 Scén_3 = (GA, IMC, EP)

Néanmoins, il est clair que cette vue "de haut niveau" n'est pas suffisante pour générer des traces réelles d'attaques. C'est pourquoi nous l'utilisons conjointement avec deux autres modèles: un modèle pour le comportement des attaquants et un autre qui classe les outils d'attaques (en particulier, nous avons répertorié les outils et commandes qui permettent de réaliser chacune des primitives de la figure 7). L'ensemble de ces trois modèles nous permet d'instancier les scénarios abstraits, et donc de générer des scénarios exécutables réels.

En particulier, la transformation est faite en associant les étapes abstraites avec des commandes concrètes ou des instances d'exécution d'outils qui réalisent et implémentent ces étapes. Par exemple,

- **R** est associé avec (c'est-à-dire peut être exécuté de manière concrète par) : *nessus, nmap, ping, traceroute*, etc.
- **VB** peut être exécuté de manière concrète par : *ls, ps, uname*, etc.
- **AG** peut être associé avec : *SSH, telnet, exécuter/exploiter une vulnérabilité metasploit*, etc.
- **CDI** peut être associé avec : *cp, rm, mv, éditer un fichier de configuration, changer les variables d'environnement*, etc.
- **EP** peut être associé avec : *crontab, lynx, nc*, etc.
- **HT** peut être associé avec : *rm log, kill syslog, tuer un processus antivirus*, etc.

- *DoS* peut être associé avec : shutdown -halt, crash system, arrêt d'un service (stop service), etc.
- *IMC* peut être associé avec : scp malveillant, ftp malveillant, exécution de metasploit avec une charge utile malveillante, etc.

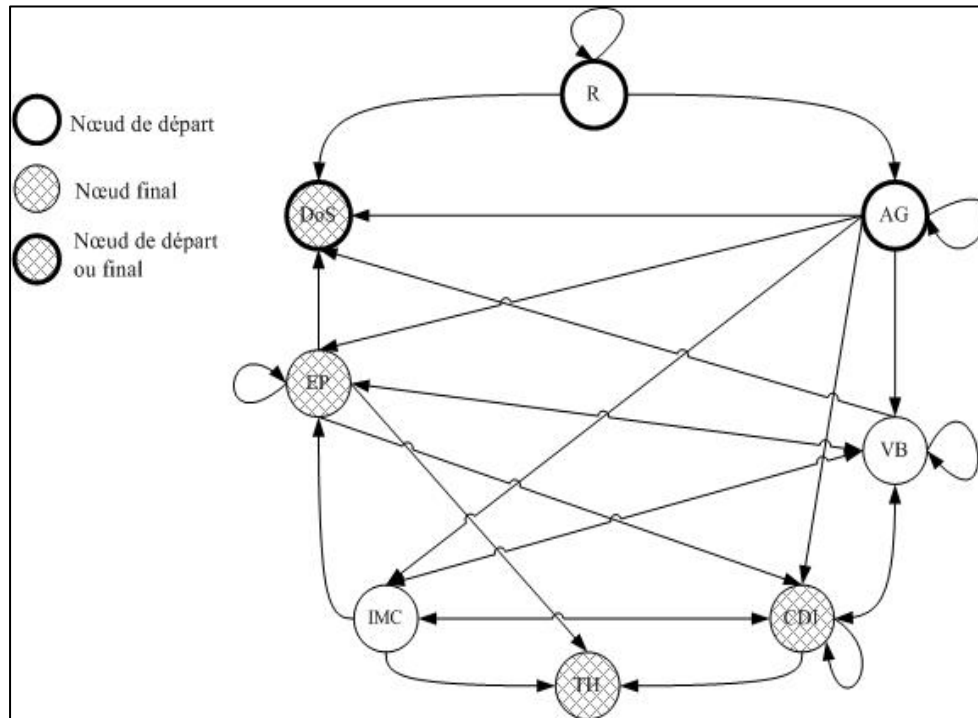


Figure 7 : Machine à état représentant le processus d'attaque

Un scénario abstrait est donc transformé en scénarios exécutables par la traduction de ces actions primitives à des séquences d'actions exécutables. Chaque action primitive est remplacée par une ou plusieurs commandes capables de la réaliser. La transformation peut être faite selon différentes techniques : de manière exhaustive, en choisissant aléatoirement une des commandes qui peuvent implémenter chaque étape du scénario, ou en utilisant des algorithmes de transformation plus sophistiqués qui prennent en considération des paramètres comme les résultats des étapes précédentes et le contexte de l'attaque.

On peut éventuellement paramétrer la transformation par un modèle de compétence de l'attaquant. Pour cela, nous caractérisons l'attaquant par son niveau de compétence (*Débutant*, *Intermédiaire*, *Avancé*), son profil (*Hésitant*, *Agressif*), l'ensemble d'outils qu'il possède, et son adresse IP. Un attaquant peut être donc représenté comme suit :

$$\text{Attaquant} \equiv (\text{Niveau}, \text{Profil}, \{\text{outils}\}, \text{IP})$$

Notons qu'il peut également être intéressant de paramétrer les scénarios d'attaques selon des statistiques issues des données collectées à partir de réseaux de pots de miel. Ces statistiques peuvent fournir des informations intéressantes telles que la fréquence d'utilisation de certains outils d'attaques, la fréquence d'occurrence de chaque attaque, les adresses IP les plus utilisées comme source d'attaques, etc.

Il est important de noter que cette approche itérative de génération des scénarios d'attaques nous a permis de pallier le problème de l'explosion combinatoire, problème intrinsèque aux approches classiques de génération des scénarios d'attaques. En effet, dans notre modèle, le nombre de cas possibles est fortement réduit, grâce notamment à des contraintes sur les boucles et les relations de précedence (déduites à partir de notre graphe).

VI. Mise en œuvre et expérimentation

Les idées et modèles présentés dans les sections précédentes constituent la base d'un ensemble d'outils que nous avons développés pour l'évaluation des IDS. Notre implémentation consiste en trois parties principales : un gestionnaire d'évaluation, un générateur d'attaques et un générateur de trafic de

fond. Dans la version actuelle, ce dernier est implémenté séparément comme un logiciel *Java*. Nous avons opté pour une implémentation du gestionnaire d'évaluation et du générateur d'attaques (en utilisant le langage *Ruby*) comme un plugin de *metasploit*. D'ailleurs, nous avons développé un autre plugin de *metasploit* qui fait l'interface avec la base de données dans laquelle nous stockons les informations reliées à la classification des attaques élémentaires. L'architecture globale de notre outil *toolkit* d'évaluation et des interactions entre les différents outils sont illustrées dans la figure 8.

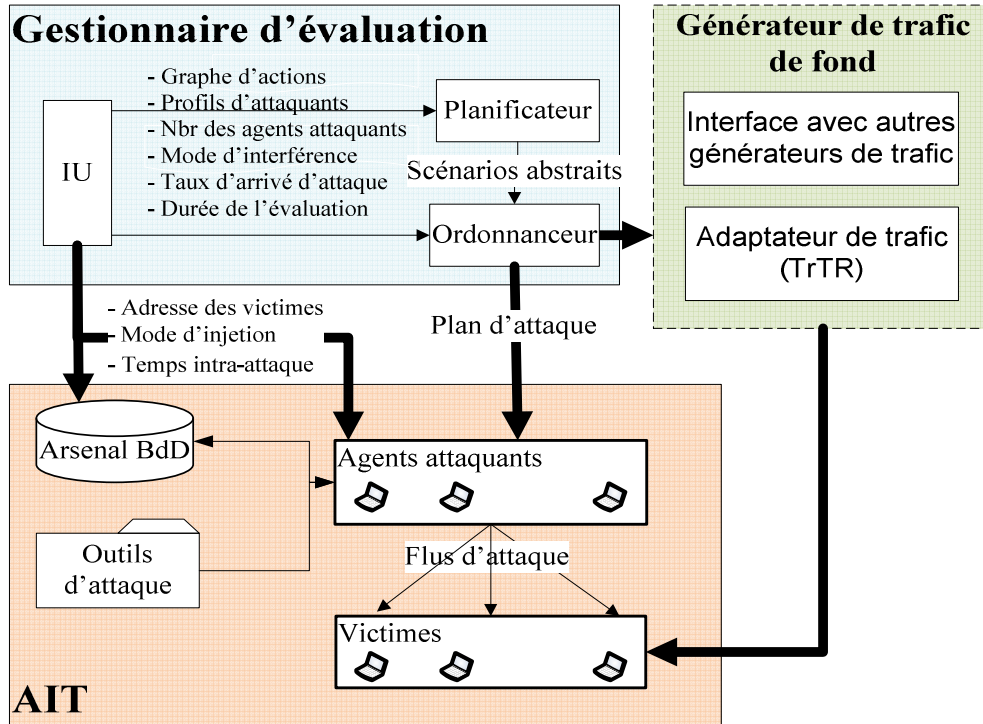


Figure 8 : L'architecture globale de nos outils d'évaluation

Contributions et perspectives

Les principales contributions de ce travail peuvent être résumées par les points suivants :

- 1) Nous avons proposé une méthodologie systématique pour l'évaluation et le test des IDS.
- 2) Nous avons fait une analyse profonde pour caractériser les données d'entrées traitées par les IDS. Cette analyse nous permet de concevoir, construire et produire des données d'évaluation qui soient le plus possible représentatives des données réelles.
- 3) Nous avons créé un schéma de classification des attaques qui peut être employé pour choisir les cas de test représentatifs d'attaques, mais aussi pour analyser et présenter les résultats d'évaluation.
- 4) Nous avons conçu et développé des outils d'évaluation qui permettent l'injection des attaques et la génération du trafic de fond. Ces outils peuvent être adaptés selon les besoins des évaluateurs, les caractéristiques de l'IDS évalué ou l'environnement d'évaluation.
- 5) Finalement, en appliquant notre approche et en utilisant les outils d'évaluation que nous avons développés, nous avons illustré comment produire des données représentatives d'attaques.

Il est également important de noter que ce que nous proposons dans ce mémoire ne représente qu'une petite partie des applications possibles de notre approche. Le fait de travailler sur un sujet très riche nous a inspiré beaucoup d'idées qui augurent d'une suite prometteuse sur l'évaluation des IDS. Les travaux futurs consistent non seulement à améliorer les outils développés, mais aussi à approfondir travail de recherche initié dans ce domaine à travers cette thèse. À court terme nous envisageons de :

- 1) raffiner le schéma de classification
- 2) classifier plus d'outils d'attaques
- 3) raffiner le modèle de processus d'attaques

-
- 4) faire des évaluations plus exhaustives des IDS
 - 5) développer une interface graphique unique pour gérer le processus d'évaluation et afficher les résultats
 - 6) améliorer l'intégration entre les différents composants de notre plateforme de test
 - 7) étudier la possibilité d'utiliser les plateformes d'émulation de réseaux, tels que ReAssure et PlanetLab.

PART II

I. Chapter 1: Introduction

In a highly connected world, people worry about the security of their connected devices. In order to protect information circulating on networks or stocked on storage media, several defense lines have to be put in place. Examples include access control mechanisms, firewalls, encrypted communication channels, antivirus tools, etc. Each of such tools contributes to the enforcement of security policies defined by the organization or the computer system owner. Unfortunately, even with the implementation of several lines of defense, a perfect security cannot be guaranteed. For this reason, complementary mechanisms have been added: the Intrusion Detection Systems (IDSes). The philosophy behind intrusion detection is that even with the use of several protection mechanisms, attacks as well as intrusion incidents are always possible. Therefore, if we cannot provide a complete protection to our networks and computer systems, we should at least be aware of the occurrence of attacks, to react with appropriate responses and take corrective actions. The detection of intrusion occurrences is as important as protection because ignoring security breaches may result in a continuous leakage of information and thereby a continuous significant loss.

Nevertheless, even if detecting intrusions is an interesting concept, for the moment, implementations of the intrusion detection concept stay far below the expectations. Many ideas and algorithms were proposed since the creation of the first IDS, and several of them could be innovative. Unfortunately, the value of these ideas cannot be fairly assessed because IDS researchers and developers lack for effective methods and tools that allow evaluating IDS precisely.

1.1. Motivation

Intrusion detection systems are relatively recent. There are different types of IDS that are characterized by various detection techniques, architectures and scope {Debar00}, {Debar05}. Almost all IDS types suffer, with different degrees, from two common problems: the huge number of *false positives* and of *false negatives*. A false positive occurs when the IDS signals an intrusion for normal activities (false alarm), while false negatives occur when attacks or intrusions pass without detection (i.e., no alarm is generated).

IDS developers try to overcome these limitations of IDSes by developing new algorithms and architectures. However, they need to evaluate the improvements provided by these new features. Similarly, network administrators and security analysts need to evaluate IDSes either to select the best ones before acquisition or to assess their efficiency after the installation in the heart of their networks. Unfortunately, the enhancements of IDS are often disappointing, since large numbers of false positives and false negatives persist in new versions of IDS with little improvement over previous versions. Although we see continuous research and development effort {Mé01}, changes do not appear to represent significant breakthroughs, neither in technology nor in usability. This is partially due to the lack of effective evaluation and testing methods for such type of tools.

Several evaluation attempts took place in the last years {Puketza96}, {Puketza97}, {Lippmann00a}, {Lippmann00b}, {Debar98}, {Debar02} and {Alessandri04}. However, they have serious shortcomings and unfortunately, most of them suffer from serious limitations {Mell03}, {McHugh00a}. Consequently, under evaluation, IDSes exhibit a behavior that is different from their behavior when installed in real environments, which thus leads to misleading conclusions.

Many surveys of the research on intrusion detection, ancient ones as well as the more recent, have reported IDS evaluation and testing as a high-priority open research area {Mukherjee94}, {Axelsson98a}, {Jones00}, {Allen00} and {Lundin02}.

Contrarily to what may be thought, evaluating Intrusion Detection Systems is a non-trivial task, because it requires a deep understanding of the evaluated tool (i.e., the IDS) as well as of attacks and malware, to find out the best way to evaluate these tools. Furthermore, security-related tools such as firewalls, IDS, IPS, antivirus, etc., have special particularities, as they should deal with unexpected and probably unknown use patterns, with abuse of the tools themselves and with vulnerabilities of the surrounding systems. These considerations make the construction of appropriate (i.e., representative) evaluation datasets extremely tedious and time-consuming.

In addition to that, evaluating security-related tools such as IDSes involves issues from various domains, such as networking, operating systems, software testing and, of course, security. Having a good knowledge of all these areas is essential to be able to evaluate or test such tools properly.

Since IDS evaluations have intersections with the aforementioned domains, we have dived into several sub domains to pickup techniques and methods that can be applicable for the evaluation of security-related tools in general and more particularly of intrusion detection systems. This allowed us to benefit from advances in other fields that are more mature and helped us in not reinventing the wheel.

1.2. Research goal

The ultimate goal of this work is to improve the quality of intrusion detection systems by providing enhanced evaluation procedures, datasets, tools and metrics. This is expected to serve not only IDS developers, but also IDS users in comparing different IDS products. We focus mainly on the improvement of the evaluation process itself and on the creation of realistic datasets. This will hopefully reduce the time required by the evaluator (either as an IDS developer, a network administrator, or third party assessor) to construct datasets of good quality and allows giving more attention to the experimentation design.

1.3. Approach

IDS research has often focused on autonomous response and other “advanced” issues, at the expense of addressing how to detect and diagnose attacks more accurately. Improving these features is meaningless unless the underlying mechanisms of analysis and detection become reliable and credible. For this reason, we focus mainly on the detection and diagnostic capabilities of IDSes as opposed to some previous evaluations that have distracted the attention towards organizational and non-technical issues such as cost of acquirement ease of installation and ease of use. We do not deny the importance of organizational aspects and ease of deployment, but given the current state of intrusion detection systems, we believe that these have a lower priority.

Figure I.1 illustrates the roadmap and the milestones of the work presented in this thesis. First, we surveyed the literature to establish the state of the art in intrusion detection and to analyze the previous evaluations as well as their limitations. This allowed us to draw a map of the field, and to identify the problem and the mistakes committed in previous evaluations. Amongst those, the most critical are the following: (1) the use of ad-hoc approaches, (2) the use of non-representative datasets without sensitivity analysis and finally (3) the use of incorrect metrics.

Our prime objective when addressing the IDS-evaluation problem is to correct and avoid these mistakes or reduce their effects. Thus, the first part of our approach relies on improving the evaluation process itself, to provide a well-established and systematic evaluation. The second step is to characterize the data inputs of an IDS and determine their main features. As a result, we can make an idea about datasets representativeness and how they should be constructed to match real-world data. Besides that, we treated the last part (i.e., the metrics) by defining various new metrics that measure specific aspects or functionalities of IDSes. The broad range of the suggested metrics allows evaluators to select those metrics that fit the best their evaluation goal.

1.4. Contributions

The main contributions of this work can be summarized in the following points:

- We propose an engineered approach to make systematic evaluations,
- We made a thorough analysis and characterization of IDS input/workload that, consequently, allows us to design, construct and generate evaluation datasets as close as possible to the reality.
- We created a classification scheme of attacks that can be used for the selection of representative attack test cases as well as the analysis and the presentation of evaluation results.
- The design and the implementation of a flexible evaluation tool that provides attack injection as well as background traffic generation. Both can be customized to fit the evaluator’s needs concerning the characteristics of the evaluated IDS and of the target system environment.

- Finally, by applying our approach and by using the evaluation tools that we developed, we have illustrated how effective attack datasets can be generated.

The chapters of this document are dedicated to detailed descriptions of each of these contributions.

1.5. Thesis Outline

The remainder of the dissertation is organized as follows:

Chapter 2 introduces the main concepts of security while focusing on those related to intrusion detection. It begins by describing briefly some of the common attack threats, security countermeasures and concludes by a detailed description of intrusion detection systems: their types, input data sources, detection methods as well as their strengths and weaknesses.

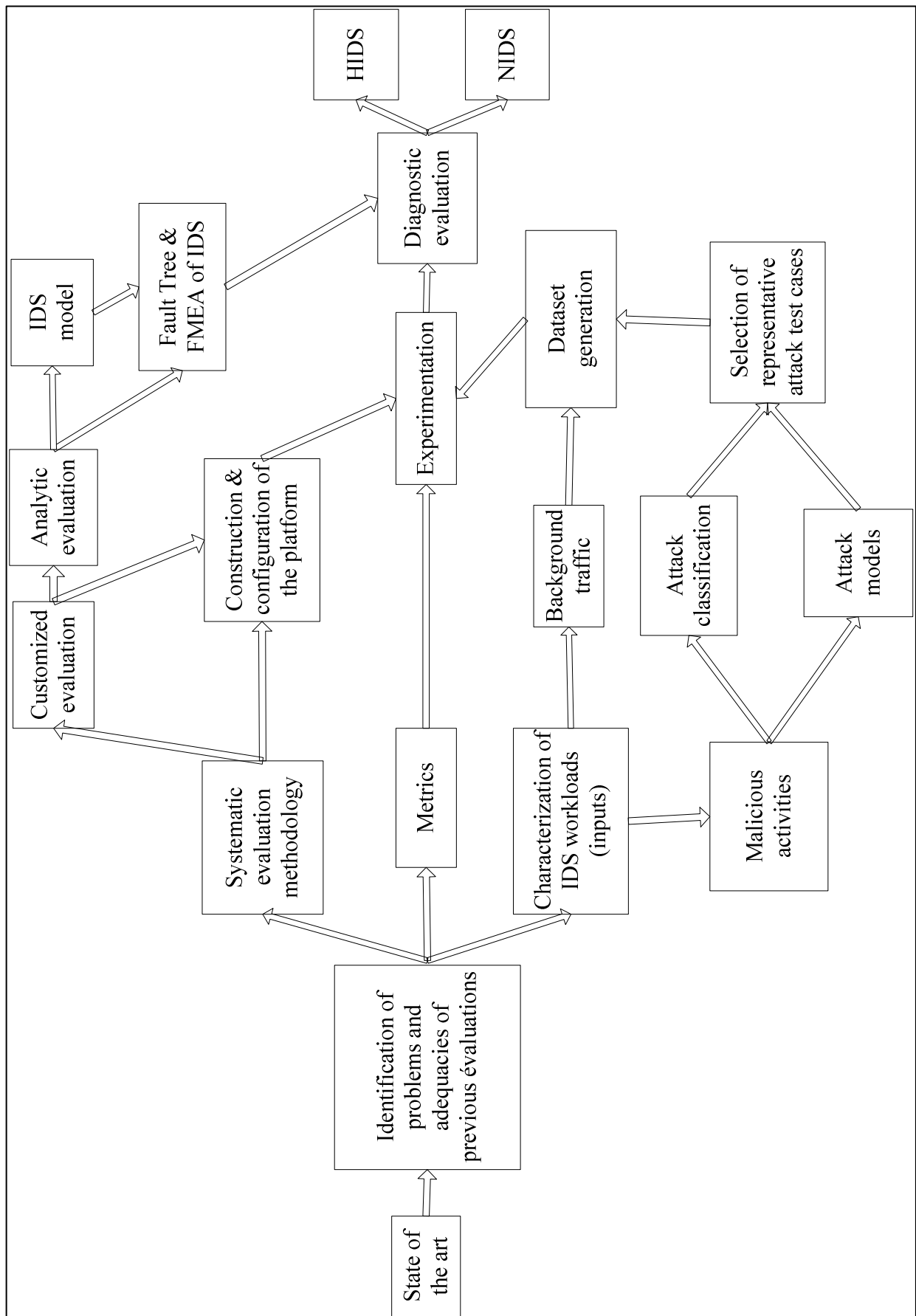
We present in Chapter 3 a brief survey of the literature that describes the current state of the art in IDS-evaluation as well as an overview of existing evaluations: describing the most important ones, distinguishing the evaluation types and discussing the limitations of such evaluations. Then, we draw the outlines of a global evaluation framework and its main parts, which include an engineered evaluation methodology and the underlying models of our approach and explain the benefits of using such a framework. Finally, we identify the main challenges inherent in IDS evaluation that hinder break through improvements in this field.

Chapter 4 is dedicated to the characterization of IDS workloads because we identified the representativeness of evaluation datasets as one of the most significant limitations and any attempt to generate a representative dataset requires first to characterize the real-world datasets (i.e, workloads). Therefore, we describe both the attack and the background components that exist in the real-world workloads. We give more focus to the attack component, which we describe it in more details at two levels: the elementary attack level and the attack-scenario level. Moreover, we characterize *metasploit*, which is one of the tools that are increasingly used by IDS evaluators, in attempt to figure out whether it is representative of the real-world attacks and if it can be sufficient to perform a rigorous IDS evaluation. Finally, we close the chapter by discussing the limitations that may persist in attack generation and selection approach.

In Chapter 5, we propose a model-driven approach for generating evaluation datasets. We explain how the theoretical concepts and models presented in the previous chapters can be transformed into a consistent method to generate representative datasets. Moreover, we outline the architecture and some implementation details of the tools that we have developed to implement the discussed ideas. Mainly, there are two tools: one for generating attack scenarios and the other for replaying captured network traffic. A third tool is concerned with the management and the configuration of the whole evaluation platform that allows the customization of evaluation environments to produce datasets with various characteristics.

Chapter 6 presents the experimentations that we made with the tools and an explanatory case study on how to apply our evaluation approach.

Finally, Chapter 7 states the conclusions and the future work including some ideas to further improve the evaluation of intrusion detection systems as a pursuit to the work on such a rich subject.



II. Chapter 2: Background

In this chapter, we introduce the basic concepts related to intrusion detection and to the security problem that we are addressing, in order to provide a casual reader with the background necessary for this dissertation.

The evaluation of IDSes spans several domains such as information security, software testing, networking, and performance modeling and analysis. Sometimes, the same term may have different or even inconsistent uses in the different domains. To prevent confusion, we explicitly define, where necessary, these terms to clarify what we mean by each term. We have adopted in most cases the definitions stated by the project MAFTIA {Maftia03}, with some additions and slight modifications, as we find MAFTIA's work strongly relevant to our context (i.e., IDS evaluation). In the second part of this chapter, we present the security problem we are addressing, as well as the most common attacks and the main security tools that may have a relation with intrusion detection. Then, we dedicate a separate section to IDSes with their types, detection techniques, limitations, etc. All this background is necessary to introduce the next chapter, which will present a brief overview of the most significant work previously done on IDS evaluation and testing.

2.1. Introduction

Security of computers, networks and information has become an important subject both as a research topic and in the public media where we hear terms such as cyber attacks, intrusions, cybercrimes, etc. This is a natural consequence to the spread and the implication of computers and computer-based systems in all details of our modern life. The dependence on computers and on the Internet to process, exchange and store huge amounts of personal, public or business information makes the protection of such systems and of their data a very critical mission.

Indeed, *security* is defined in the ITSEC {ITSEC91} as a composite notion, namely the “*combination of confidentiality, integrity and availability*”, with *confidentiality* defined as “*the prevention of unauthorized disclosure of information*”, *integrity* defined as “*the prevention of modification or unauthorized suppression of information*” and *availability* defined as “*the prevention of unauthorized retention of information*”. Accordingly, a computer system or a network is assumed secure, if these properties are maintained.

As explained in {Maftia03}, other security properties, such as *privacy*, *non-repudiation* and *authentication*, can be derived from *confidentiality*, *integrity* and *availability* of data or meta-data. For example, *privacy* means respecting the liberty of individuals and protecting their private lives. It has a direct relation to the *confidentiality* of personal data and meta-data such as the identity of users who carried out particular operations or transactions.

Authenticity is the property of being true. The *authenticity* of a message, by example, is equivalent to the integrity of the message contents (data integrity) and the integrity of its origin (meta-data) and optionally the integrity of other meta-data such as transmission date and time. The *non-repudiation* property guarantees that the person who carried out an action within the system cannot deny this fact. Thus, it corresponds to the availability and the integrity of meta-data such as the identity of users (e.g., who sent the message).

To protect computer systems against hackers and malicious users, various security mechanisms have been implemented to maintain security properties according to some predefined *security policy*⁷. In particular, a part of the *security policy*, namely the *authorization policy*, determines who is authorized to do what. A security policy does not provide protection by itself; it needs security mechanisms to be enforced. The policy can be described by various forms of access control models (e.g., MAC, DAC, RBAC, etc.) and enforced by access control mechanisms such as hardware implemented capabilities {Abouelkalam03}.

Unfortunately, access control can often be subverted or bypassed by attackers because the underlying security policy can be sometimes incomplete, imprecise or badly implemented. Therefore, other

⁷ A *security policy* is a description of 1) the security properties to be fulfilled by a computing system; 2) the rules according to which the system security state can evolve.

countermeasures such as *firewalls*, *antivirus tools*, *intrusion detection and prevention systems*, etc. have been invented but none of these techniques is perfect.

Indeed, computer intrusions⁸ have been occurring since at least the 1960s and continue to occur despite of the deployed counter-measures. With industry and governments are becoming increasingly dependent on networks for doing business, computer intrusions with the goals of obtaining economic/competitive advantage, political/military intelligence, and financial gain have become more prevalent.

The growth of incidents on the Internet reflects the growth of the Internet itself as well as the number of attackers but it also indicates how attacking culture and tools had become more accessible. Figure II.1 illustrates this growth of the number of incidents reported to US CERT/Coordination Center {Cert08} over the years from 1993 to 2003. By the end of 2003, they have stopped counting the number of incidents because “Given the widespread use of automated attack tools, attacks against Internet-connected systems have become so commonplace that counts of the number of incidents reported provide little information with regard to assessing the scope and impact of attacks. Therefore, we stopped providing this statistic at the end of 2003” {Cert08}.

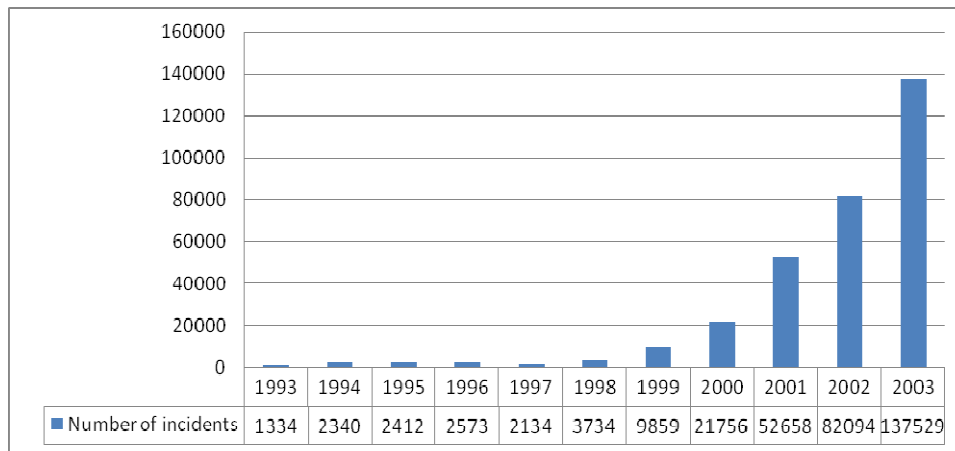


Figure II.1: Growth in the number of incidents handled by US-CERT from 1993 to 2003.

The continuously growing number of attack⁹ incidents as well as their severity can be explained by the increased connectivity and complexity and by the increased publicity of vulnerability¹⁰ information and attack scripts via the Internet. Figure II.2 shows the number of vulnerabilities discovered during the period from 1995 to 2007.

Consequently, much more viruses, worms and other malware than ever have appeared in the last years. Reports vary, but some estimates suggest that there were five times as many variants of malicious programs in circulation in 2007 as compared with 2006. For example, the security software testing organization *AV Test* {Avtest08} reported that it identified 5.49 million unique samples of malicious software in 2007 – over five times more than the 972,606 in 2006, (see Figure II.3).

Even if many samples do indeed correspond to the same malware¹¹, the broad trend shows a steep rise. Variants are often created to defeat security tools. For instance, the same Trojan can mutate sometimes hourly or daily just to try to escape detection by virus scanners. It follows that security analysts cannot cope with the frequent discovery of new malware instances to produce the corresponding signatures for each new variant. Moreover, creating variants of variants will result, shortly, in new

⁸ *Intrusion*: a *malicious*, externally or internally induced *fault* resulting from an *attack* that has succeeded in exploiting a *vulnerability*. A *fault* is the adjudged or hypothesised cause of an *error*, which cause is intended to be avoided or tolerated. {Maftia03}

⁹ *Attack*: a malicious technical interaction *fault* aiming to exploit a *vulnerability* as a step towards achieving the final aim of the attacker. {Maftia03}

¹⁰ *Vulnerability*: a fault created during development of the system, or during operation, that could be exploited to create an *intrusion*. {Maftia03}

¹¹ AV-Test counted the number of files with different MD5 hashes (fingerprints). This includes many samples of the same malware that is packed using a different run-time packer or that is differently encrypted.

malware instances that have less common features with the original ones. Nevertheless, whether an attack instance is a variant or a completely new malware, security tools could lose the fight against malware producers if they continue to use one signature per variant.

In the next section, we describe briefly various forms of the most common attacks. Then, in Section 2.3, we discuss security countermeasures that are supposed to provide protection against attack threats.

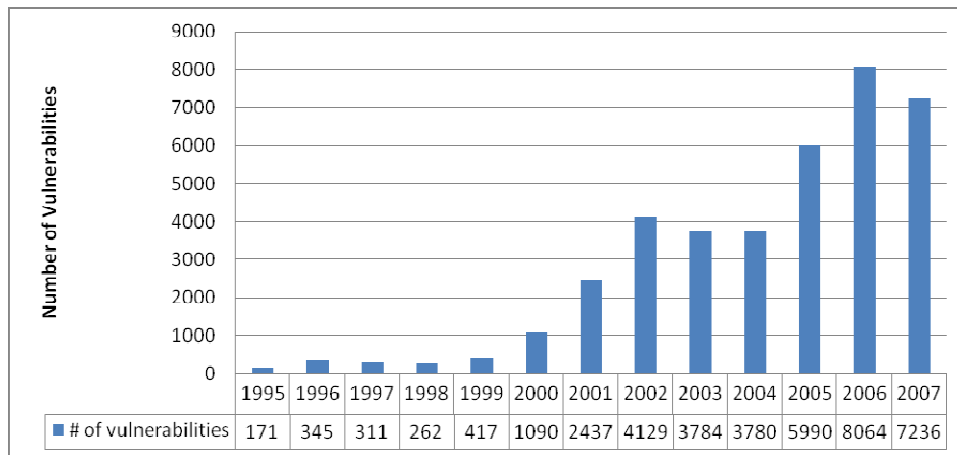


Figure II.2: Growth in number of vulnerabilities catalogued by US CERT.

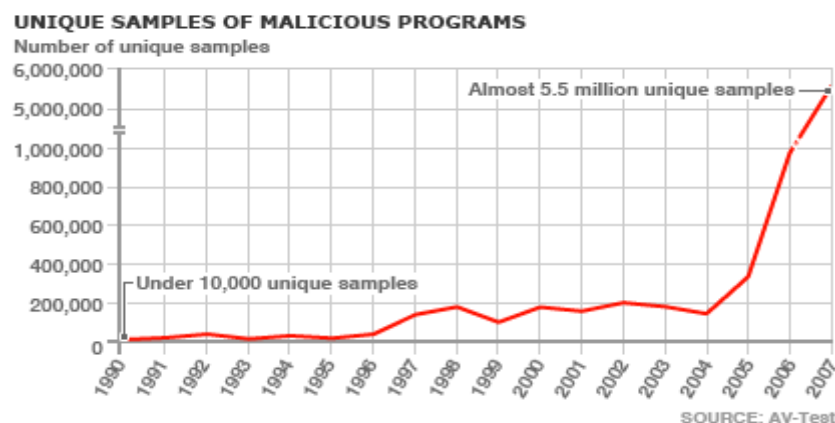


Figure II.3: The continuous growth of the number of captured malwares.

2.2. Examples of Common Attacks

Attacks take several forms to break one or more of the security properties. The attack space consists of a wide range of attack activities that enable attackers to achieve their goals. Attacks differ in scope, complexity and popularity. They can be grouped according to their functionality as described in the following subsections.

2.2.1. Gathering Security-relevant Information

Before experiencing an attack, a hacker tries to obtain necessary information that is probably sensible about the targeted system, which can be employed later to obtain access to this system. Useful information can be obtained by different ways such as network scanning and vulnerability scanning or even by using public search engines such as *Google* or social engineering methods.

A) Network Scanning

A port scanner is a piece of software designed to search for network hosts with open ports. Some port scanners only scan the most common or the most commonly vulnerable port numbers on a given host or a set of hosts or sub networks. The information gathered by a port scan may have many legitimate uses, including the ability to verify the configuration of a network by administrators. Port scanning can

however be used by those who intend to compromise security. Automatic malware such as viruses and worms rely also on embedded port scanners to find open ports. *Nmap* {Nmap08} is an example of network scanner. It can scan a network and print various information such as a list of all live hosts, the open ports (i.e., services), as well as the type of operating systems running on these machines.

B) Vulnerability Scanners

A vulnerability scanner goes a step further. It searches for open ports and checks whether they correspond to vulnerable services. In order to get over the problems related to firewalls and low bandwidth connections, a distributed vulnerability scanner can have agents on various networks, controlled by (and reporting to) a central server. Further, a large-scale vulnerability scan can be divided across multiple distributed scanners. For example, *Nessus* {Nessus08} can work in client-server mode: clients can analyze a part of a network and send all information to the server. Many malware programs incorporate a scanning engine in order to scan the local network after infection for a specific vulnerable service. An attacker can connect back to retrieve the scan result, or the malware itself can post it via an IRC connection by example.

C) Social Engineering

Social engineering {Anderson08} is the practice of obtaining confidential information by manipulating legitimate users. A social engineer will commonly use the telephone or mail to trick people into revealing sensitive information or getting them to do something against typical policies. Rather than exploiting technical computer security vulnerabilities, social engineering exploits the natural tendency of a person to trust people. It is generally agreed upon that "*users are the weakest link*" in security and this principle is what makes social engineering possible.

By this means, malicious individuals could penetrate into well-designed, secure computer systems by taking advantage of the carelessness of trusted individuals, or by deliberately deceiving them. Perhaps the simplest, but still effective attack is tricking a user by impersonating an administrator and requesting a password for various alleged purposes.

2.2.2. Access Gain Attacks

With information gathered by the above methods, attackers try to obtain a privileged access on a system by exploiting vulnerabilities in the services or the applications installed on this system or a bad configuration of the network. This kind of attacks primarily grants unauthorized access to the targeted system.

For example, one of the configuration problems is the use of weak passwords in systems where a bad policy of password definition allows users to choose simple and easy guessable passwords. Otherwise, an attacker can use cracking tools such as "*john the ripper*" {John07} to obtain passwords by brute-force. Buffer-overflow attacks are another example that allows attackers to execute arbitrary code on the targeted hosts.

2.2.3. Denial of Service

Denial of service (DoS) attacks differ slightly from those listed above, in that they are not primarily intended to gain unauthorized access or control of a system. Instead, they are designed to overload or disable the capabilities of a machine or a network, and thereby render it unusable or inaccessible (i.e., compromise the *availability* of the service(s) provided by this system or network).

A) Traditional Denial of Service

A denial of service typically leads to a complete loss or degradation of services by consuming the bandwidth of the victim network or by overloading the computational resources of the victim host. *Smurf* attacks, for example, consist in sending ICMP requests to the broadcast address of badly configured networks, with the faked, or spoofed, IP source address of the targeted victim. As a result, all hosts reached by the broadcast address will send their ICMP replies to the victim. Another classic example is *SYN flooding*, in which too frequent bogus SYN requests to a service (often HTTP) cause a server to exhaust its open connection table entries. In both cases, if the packets received by the victim exceed its capability of processing, it will get slow or even crash.

B) Distributed Denial of Service

Distributed DoS attacks (DDoS) consist in deploying coordinated denial-of-service attacks through hosts that have been previously compromised by viruses, worms or Trojan horse programs. In such attacks, the perpetrator controls the attack process remotely from the infected machines (called zombies) typically through IRC or peer-to-peer channels. Such an array of infected computers is called a botnet. With a large enough number of such zombie hosts, the services of even the largest and most well connected websites can be disrupted. A single attacker can carry out a DDoS alone, but the effect of the attack is greatly multiplied by the use of many zombies.

The simplest form of a DDoS attack is merely to send a very large quantity of request packets, to a service on the victim machine. Unless something (e.g., a firewall) between the attacking machines and the victim drops those packets, the victim will spend resources attempting to receive and properly handle the requests. With a sufficient number of such packets, all of the machine's resources will be spent trying to serve fake requests.

2.2.4. Malware Attacks

This category of attacks is used for several purposes and has variable consequences. They can result in damages as simple as displaying a simple flicker to catastrophic damages such as completely formatting hard disks.

A) Virus and Worm

A virus is a sequence of instructions that attach itself to programs. When an infected program is run, the virus is executed and tries to replicate itself by creating (possibly modified) copies of itself in other programs. The main criterion for classifying a piece of executable code as a virus is that it spreads by “contaminating an host program” in analogy to biologic viruses. A virus can only spread from one computer to another when its hosting program is executed on another, previously uninfected computer, or when the hosting program is able to modify remote programs. Virus infections usually occur by sending infected program files over a network or carrying them on removable storage media. Viruses are sometimes confused with worms. However, they differ in replication and propagation methods.

Worms are standalone programs that spread themselves to other computers without needing to be hosted by another program. Since many personal computers are frequently connected to the Internet or to local-area networks, worms can spread quickly. A worm such as *Slammer* {Slammer07} can infect thousands of hosts all over the globe just within a few minutes.

B) Trojan horse

A Trojan horse is a computer program that performs some illicit activity without the user knowledge when it is run to perform an apparently useful function. It secretly runs commands, and usually enables attackers to access the infected computer running it by opening a backdoor. A Trojan horse is not necessarily a virus, as its primary goal is not to reproduce itself to infect other machines. However, some viruses may have Trojan features (i.e., they might spread like viruses and perform illicit actions on infected machines). An anti-malware program can detect Trojans if it recognizes their signatures. Also, as for worms and viruses, firewalls can help to protect from them by restricting access to the only needed services and ports.

C) Spyware

The word spyware refers to programs that gather information on users of the computer on which they are installed and then send the gathered information to the software provider so that Internet users can be profiled and in some cases they can even contribute to identity theft. It gathers interesting data such as the URLs of the visited websites, keywords entered in search engines, passwords, payment information (credit/debit cards), or any other personal information.

Spyware programs are generally installed along with other software (mostly freeware or shareware). This enables the creators of such software to gain money by selling gathered information or statistical information about users.

Although some spyware could be legal as the license of the accompanying software may state clearly that third party programs are installed, divulging personal information and degrading the performance of

the infected machines makes it, at least, annoying. Performance degradation is due to overloading host and network resources (e.g., RAM, disk space, taking up processor cycles, network bandwidth, etc.).

D) Rootkit

According to Hoglund *et. al.* {Hoglund05}, “a rootkit is a set of programs and code that allows a permanent or consistent, undetectable presence on a computer”. Rootkits have two main features, namely hiding code and data, and providing remote access. They can employ various tricks and techniques to hide code and data on a system. For example, many rootkits use hidden files and directories. Other rootkit features are implemented for remote access and eavesdropping (e.g., for sniffing packets from the network).

E) Spam

Spamming is the use of any electronic communication medium to send unsolicited messages in bulk. While its definition is usually limited to bulk mailing and not targeted marketing, the term "spam" can refer to any commercially oriented, unsolicited bulk messages perceived as being excessive and undesired.

Messaging spam makes use of instant messaging systems, such as *AOL* instant messenger or *ICQ*. Newsgroup spam targeting *Usenet* newsgroups predates email spam, which is currently the most important vector of spam. Mobile phone spam is directed at the text messaging service of a mobile phone. *Spamdexing* (a combination of spamming and indexing) refers to the practice on the World Wide Web of deliberately modifying HTML pages to increase the chance of being ranked high on search engine relevancy lists.

2.3. Security Countermeasures

In order to eliminate or reduce the exposure to security threats, a set of security countermeasures are recommended. These countermeasures are not only technical solutions but also cover user awareness and training as well as clearly defined practices. A wide range of technical countermeasures and administrative tools can be employed to enforce security. This includes log analyzers, password auditors, network sniffers, antispyware, port scanners, vulnerability scanners, storage and communication encryption, etc. Describing all these tools and techniques is out of the scope of this chapter. However, we present, hereafter, a brief description of some of the most common security tools that have a direct relation with intrusion detection. We will provide more details about *Intrusion Detection Systems* throughout the next section (2.4) because this dissertation is dedicated to their evaluation.

2.3.1. Firewalls

A firewall is a system that allows users to protect a computer from unauthorized connections from the network or to protect a LAN from attacks from an external network or the Internet). It also allows controlling connections made by applications installed on local machines to outside networks or the Internet. Filtering network connections in both directions can be based on different criteria such as source and destination IP addresses, transport protocols, application protocols, etc.

2.3.2. Antivirus

Antivirus (AV) tools are programs that can detect the presence of viruses, worms or Trojans on a computer and remove them. Eradicating a virus is the term used for cleaning out a computer. There are several methods of eradication: (1) Clean the infected file by removing the malicious code from it; (2) Removing the infected file entirely; and (3) Quarantining the infected file, which involves moving it to a location where it cannot be run. Antivirus tools often apply signature-based detection techniques and have many similarities with intrusion detection systems. Both tools (IDS and AntiVirus) are eligible for more convergence in the near future {Morin07}.

2.4. Intrusion detection systems

According to {Mukherjee94}, {Debar05}, and {Debar04} intrusion detection is the process of detecting and identifying malicious and unauthorized use, misuse, and abuse of computer systems. Thus,

it concerns the set of practices and mechanisms that contributes to the diagnosis of attacks and/or the detection of errors that may lead to security failure¹² (adapted from the definition found in {Maftia03}).

An *Intrusion Detection System (IDS)*: is an implementation of practices and mechanisms of intrusion detection. IDSeS include all software or hardware systems that automate the process of monitoring events occurring in a computer system or network and analyzing them for clues of security breaches (i.e., compromising confidentiality, integrity, or availability, or bypassing security mechanisms of a computer or network).

Early IDS implementations have appeared since the beginning of 1980s {Anderson80}, {Denning87}. Since then, a number of research and open source IDSeS were created such as: STAT family (USTAT, NSTAT, NetSTAT) {Ilgun95}, {kemmerer98}, {Vigna99}, EMERALD {Porras97}, Bro {Paxson99}, and Snort {Roesch99}. Commercial IDSeS started to emerge starting from 1990s, for example, Cisco secure IDS (previously known as NetRanger) {Earl01} and ISS RealSecure {Realsecure08}.

Despite different implementations, all intrusion detection systems' major task is to collect data from computer systems or computer networks; analyze them to find security-relevant events and raise alarms if they find any. According to the Common Intrusion Detection Framework (CIDF) model {Chen98}, any IDS is composed of the following components:

- *E-Box*: Event-box, which collects data from the information source (e.g., network traffic, host logs), and feeds interesting data to the IDS.
- *D-Box*: Database-box in which the relevant events are stored after some preprocessing (e.g., normalization of different logs in a common format).
- *A-Box*: Analysis-box, the core unit of any IDS that manipulates the event data and contains the detection engine.
- *R-Box*: Response-box, this component is concerned with responsive actions that can be taken upon detection of intrusions. The response can be an administrative action such as modifying the firewall rules to block the intruder traffic, ending the TCP connection or simply generating an alert.

IDSeS have evolved much in the last decade and now they tend to be architecturally distributed and can integrate various sensors from different sources. Furthermore, centralized management consoles, correlation engines and reporting front ends have been proposed to facilitate the use of several heterogeneous but complementary IDSeS. We have derived the model shown in Figure II.4 that reflects the new tendencies in modern IDSeS. Through the dissertation, we base our discussion and analysis on this model, which focuses more on the functional units of the IDS. According to this model, an IDS consists mainly of at least one detector unit, at least one alarm/report generator, one or more sensor units and optionally includes preprocessing and correlation units. An IDS must have at least one sensor either of its own or it must import information from other sources such as IDS audit.

Normally, the intrusion detection process, shown in Figure II.5, starts by collecting events. It then passes them either to a preprocessing unit to normalize data or directly to the detector unit. The later analyzes the gathered data and decides whether they correspond to signs of an attack or not. If this is the case, the reporter unit generates an alarm indicating the occurrence of the attack. If the IDS includes a correlation unit, it aggregates alarms that belong to the same scenario, or extracts more information from the gathered data.

We consider *Intrusion Prevention System (IPS)* as a kind of IDS that not only detects attacks but also prevents their occurrence. It extends the functionality of IDS by a response unit to prevent attacks or limit their effects. Typical responses to intrusions may include reconfiguring the firewall to drop suspicious traffic, denying user access to resources that exhibit anomalous behavior, etc. The following section presents different types of IDS.

¹² *Security failure*: violation of a security goal of the intended security policy {Maftia03}.

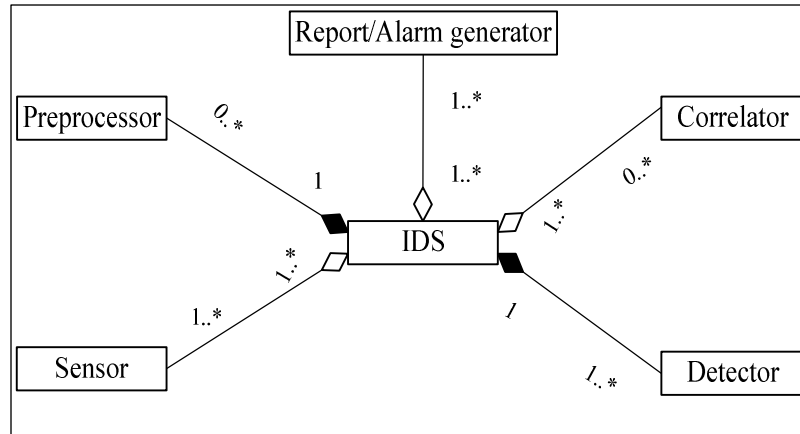


Figure II.4: IDS components.

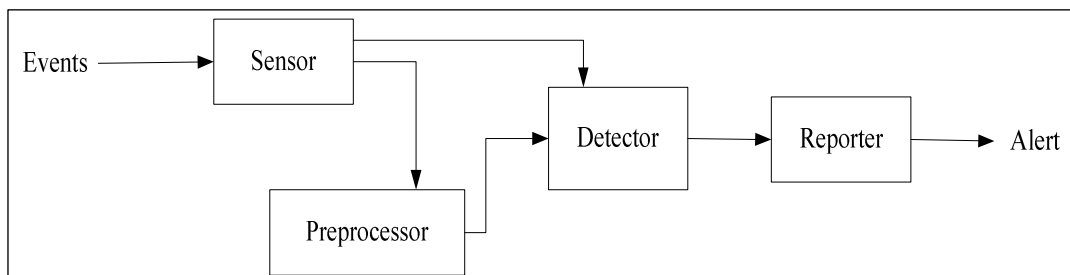


Figure II.5: Intrusion Detection process.

2.4.1. IDS types

There are several taxonomies of IDSes such as that of Debar *et al* {Debar99}, {Debar00} and Axelsson *et al* {Axelsson00} based on various criteria. Examples of criteria include:

- *The time of detection*: Two main groups can be identified: those that analyze events on line and attempt to detect intrusions in *real-time* or near real-time, and those that process audit data with some delay or offline (*non-real-time*), which in turn delays the time of detection.
- *The granularity of data processing*: this criterion distinguishes IDSes that process data continuously from those that process event data in batch mode.
- *The source of event data*: there are two major categories: (1) network-based IDSes (NIDSes), which typically read event data directly off a multicast network such as Ethernet and (2) host-based IDSes (HIDSes), which collect and analyze event data collected on the host. The host data are typically logs such as operating system kernel logs, application program logs or even firewalls logs, etc.
- *The detection method or technique*: Two categories can be distinguished: (1) behavior-based (also known as anomaly detection) and (2) knowledge-based (namely signature-based or misuse detection). Both types will be described in more details in upcoming sections.
- *The behavior on detection* (response to detected intrusions): an IDS can be classified as *passive* or *active*. Passive systems notify the proper authority, and they do not try to mitigate the damage by themselves. Contrarily, active IDS react to stop the attack (e.g., terminates the attack session)
- *The location of data collection*: audit data for the processor/detector can be collected from many different sources in a *distributed* manner, or from a single point using a *centralized* approach.
- *The location of data processing*: collected data can be processed and analyzed centrally even if it was collected from many different sources. Otherwise, it can be processed and analyzed at the same place where it was collected.

In the next section, we concentrate on the basic IDS features regarding the source of event data, the detection approach and the location of data collection and processing.

2.4.2. The source of event data

The events analyzed by an IDS can take several forms. Generally, event sources include network traffic, log files and audit data gathered locally on the host machine. Accordingly, we have two categories of IDS: *Host-based IDS (HIDS)* and *Network-based IDS (NIDS)*.

A) Host-based IDS

A *host-based* intrusion detection system monitors the host on which the sensor is installed. The event stream can be system call sequences, log records from one or more services, operating system logs, or any other log for activities within the monitored machine. Normal activities as well as intrusions may consist of a single event or of a series of events. For example, an ftp session might generate log records on the host that runs the FTP server indicating the start of the session, successful authentication, transferred files, examined directories and termination of the session. These records may be mixed with the records of other simultaneous ftp sessions as well as records from other services.

The main advantage of *HIDS* is that it can theoretically detect intrusions where a local legitimate user tries to perform some illegal actions and can help detecting attacks such as Trojan or other attacks that may involve software integrity breaches without leaving traces on network traffic. Although the *HIDS* has the advantage of not requiring additional hardware, it can cause a significant degradation in the performance of its host due to the overhead of the *HIDS* operations. Another limitation is the difficulty to port it from one platform to another. A large variety of host-based IDS ranges from integrity checkers such as *Tripwire* {Tripwire08} to multi-platform *HIDS* such as *samhain* {Samhain07} and *OSSEC* {Ossec08} that perform log analysis, integrity checking, *Windows* registry monitoring, rootkit detection, real-time alerting and active response.

For systems or applications that use logs as the primary source of information, security log analysis can also be called LID - Log-based Intrusion Detection.

B) Network-based IDS

A *network-based IDS* is an appliance that monitors the whole traffic that passes on the network segment or monitors only the traffic directed to or from the host on which it is installed. This type of IDS has the advantage that a single sensor, properly placed, can detect attacks that target multiple hosts. However, it has its own limitations. For example, it cannot detect attacks carried out locally that have no manifestations on the network card (e.g., attacks executed by a local user from the console). Besides that, it is difficult to analyze encrypted connections and impossible to see traffic that does not pass on the monitored segment or that use other links such as modem connections. Moreover, on high-speed networks, analyzing all packets in real-time may require processing capacities that exceed those available on most computers.

To overcome these limitations, *network-based IDSes* are often organized as a set of single-purpose sensors or hosts placed at various points in the network. These units monitor network traffic, perform local analysis of that traffic and report attacks to a central management console. That way, a few well-placed network-based IDSs can monitor a large network.

Snort {Roesch99}, Bro {Paxson99}, and Cisco Secure IDS {Earl01} are examples of network-based IDS.

2.4.3. Detection method

The detection method is the technique used by an IDS to determine whether an intrusion has occurred or not. There are two broad categories of detection methods: *anomaly-based* or *signature-based* (also known as *behavior-based* and *misuse-based* respectively).

A) Signature-based IDS (Misuse detection)

Misuse or signature-based detectors analyze system activities, looking for events (or sets of events) that match a predefined pattern of events describing a known attack. This implies the analysis of signatures that represent a known pattern of attack. A signature can be the interpretation of series of packets or a piece of data contained in those packets. It may also manifest in audit records, logs, or in changes in files or memory of the compromised system.

This type of IDSes can only detect previously known attacks. Therefore, they must be constantly updated with signatures of new attacks. Signature definition is a critical task. If signatures are loosely defined, the IDS will detect a broader range of attacks at the expense of generating more false alarms. On the other hand, if signatures are tightly defined, this will reduce the number of false alarms but the IDS will be unable to detect variants of common attacks.

B) Behavior-based IDS (Anomaly detection)

Anomaly detection identifies any unacceptable deviation from the expected behavior on a host or a network. It assumes that attacks are different from “normal” (legitimate) activity and can therefore be detected by systems that identify these differences. Expected behaviors of users, hosts or network connections are constructed, in advance. Profiles can be created manually or automatically based on historical data collected over a period of normal operation (supposed free of attacks). An automatically developed profile is created by software that collects and processes characteristics of system behavior over time and forms a statistically valid sample of such behavior. Note that unexpected behavior is not necessarily an attack; it may represent new, legitimate behavior that needs to be included in the profile.

The measured features that may comprise a profile include the number of failed login attempts to the system, the time or location of login, the number of files accessed by a user in a given period of time, etc. Several techniques are used to determine whether the behavior is abnormal such as statistical techniques, rule-based techniques, genetic algorithms, neural networks as well as immune system models {Somayaji98}.

Unfortunately, *behavior-based IDSes* often need a training period and are sensitive to the training dataset. Therefore, they often produce a large number of false alarms, as normal patterns of user and system behavior can vary widely. Despite this shortcoming, researchers assert that *behavior-based IDSes* are able to detect new attack forms, unlike *signature-based IDSes* that rely on matching patterns of past known attacks. Contrarily, alarms generated by *behavior-based IDSes* are less precise than those generated by its signature-based counterpart. The later often identifies the detected attack and provides rich information such as references to the exploited vulnerabilities and even advices to correct them.

2.4.4. Locations of data collection and data processing

A monolithic network IDS deployed on a single host cannot see or handle all traffic passing, neither on switched LANs nor on networks with high data rates. Moreover, it is no longer able to treat massive volumes of heterogeneous security data. For these reasons, Distributed Intrusion Detection Systems such as DIDS {Snapp91}, EMERALD {Porras97}, GrIDS {Staniford-Chen96} have been created to monitor more hosts and several points within the network.

Basically, two architectures of distributed IDSes have to be considered: first, an architecture with distributed sensors but centralized analysis like *DIDS* {Snapp91} and *Prelude* {Prelude08} where sensors that support different detection techniques could be integrated. For example, *Prelude* integrates *Snort* {Snort08} as a NIDS, *prelude-lml* as a HIDS that analyzes system log files and *Samhain* as another HIDS that checks file integrity. Although this architecture allows monitoring several points, it exhibits a single point of failure once an intruder manages to get the central analyzer down. It suffers also from the poor scalability due to the limited capacity of the analyzer and the excessive data transmission between sensors and analyzers.

The second architecture is hierarchical where the IDS is structured in several layers and redundant components such as EMERALD or GrIDS. Thus, there is no single point of failure and the scalability is improved as the analysis burden is distributed over many hosts. However, reconfiguring such systems is difficult and not flexible. Distributed IDSes can be implemented following the client/server model as well as by using agent-based approaches.

2.5. Limitations of Intrusion Detection Systems

As partially seen in the previous sections, each IDS type has its own strengths and weaknesses. However, intrusion detection, in general, suffers from common limitations, which we summarize in the following points:

- 1) ***The excessive number of false alarms*** is the most persistent obstacle that prevents intrusion detection systems from playing effectively the expected role in preventing attacks. Generally, the

number of false alarms generated by behavior-based IDSes is higher than the number of false alarms generated by signature-based IDSes.

- 2) **Weak and imprecise identification:** even when IDSes detect attacks, they sometimes badly identify them. Consequently, this affects the diagnosis capability, which is essential for restoring the compromised systems as well as for taking corrective and preventive actions {Debar02}. This problem is more obvious in behavior-based IDSes.
- 3) **Limited correlation:** alarm correlation had become better in *recent* versions *of* either HIDS or NIDS. Simple alarm correlations (e.g., alarm aggregation by source IP address) are now possible in most IDSes. However, both correlation *of various events* by *one* IDS and heterogeneous correlation of alarms generated by different IDSes (*including both* HIDS and NIDS) *are* still limited{Debar04}.
- 4) **Evasion techniques:** in addition to the limitations that are inherent to the nature of attack events, evasion techniques aim to make detecting attacks more difficult or impossible. To avoid detection, attackers develop novel methods to render their activities stealthier, invisible or not analyzable by the IDS.
- 5) **Novel attacks** (also known as zero-day attacks) are attacks that have never been seen before or are unknown previously. These attacks exploit either newly discovered vulnerabilities or old ones in a new way. A signature-based IDS assumes a minimum knowledge about how the attack manifests in the information source. It requires a signature or some kind of model that describes this attack to be included in the knowledge base. This implies that the attack is already “known” for the IDS. On the other hand, behavior-based IDS is supposed to detect zero-day attacks because they observe any deviation of the normal behavior profile not the manifestations of particular attacks. Unfortunately, not all new attacks deviate significantly from normal behavior, and in any case it is difficult to test behavior-based IDSes against unknown attacks!
- 6) **Attack variation:** as we have seen in Section 2.1, this is a serious challenge to the intrusion detection technology that is directly related to (and amplifies) the previous limitations. In the near future, we can expect a significant impact on signature-based IDSes, due to the rise we have seen in the variety of captured malware. Even if the number of completely new malware has increased, most new attacks are variations of already existing malware. Even with minor variations, it requires a new signature to be added for signature-based IDS. Fortunately, behavior-based IDSes are less sensitive *to* attack variations if they can detect the original malware and if the new variation causes similar deviations from the normal behavior.

2.6. Conclusion

Given the increased cyber threats, we need to find more effective and efficient security counter-measures either by creating new security-mechanisms or by improving existing technologies.

Otherwise, security countermeasures present so serious limitations that they might lose the fight against attackers. Limitations of current intrusion detection technology prevent both automatic responses and strong forensics to be practically feasible. Unless they produce more accurate and consistent alarms, outputs from intrusion detection systems will lack enough credibility to initiate post-detection actions (i.e., attack prevention, active responses or forensics).

Of course, there is no miraculous solution for these problems. These limitations are not only due to weaknesses in implementations of existing tools but also due to fundamental problems inherent to detection algorithms and underlying attack models. The bad news is that aggressors can take advantage of these limitations to make detection fail.

We aim by this work to promote the improvement of intrusion detection technology. We opted to work on the evaluation of intrusion detection systems because testing and evaluation of systems is a central activity in the development or the amelioration of systems. We believe that whenever we want to examine the correctness or the effectiveness of designs, implementations or detection algorithms of IDSes, IDS evaluation is the key tool. This is why, throughout the next chapters, we will try to create an adequate evaluation methodology.

III. Chapter 3: An Evaluation Framework for Intrusion Detection Systems

This is an introductory part for the evaluation of intrusion detection systems. In this chapter, we treat the evaluation as a whole process. First, we present some definitions and we explain why we need to evaluate intrusion detection systems and how such an evaluation can be carried out. Then, we present briefly the state of the art on the most relevant IDS evaluations and analyze them to find out what goes wrong with these evaluations. From this point, we can identify their limitations and consequently determine the requirements for a satisfactory IDS evaluation. Accordingly, we propose a systematic methodology to evaluate intrusion detection systems independently from their types. Finally, we give an overview of the main components of this methodology, which will be sharpened in the next chapters and thereby applied in Chapter 6.

3.1. Introduction

The evaluation phase is a fundamental activity in the development and the acquirement of computer systems and products. It allows both developers and users to judge the effectiveness, the efficiency and the robustness of their systems. Generally, progressive enhancements in evolving technologies are often the result of unbiased evaluation methodologies and techniques. Inversely, fields that have a slow progress often lack of robust evaluation methodologies, tools and/or metrics. Instead, they often use heuristic assessment methods and/or unclear metrics. The intrusion detection field represents a clear example for the lack of well-established evaluation methods and tools. This may explain why intrusion detection systems are still failing to play effectively their expected role in detecting and preventing computer and network attacks. We argue that this is partially due to the absence of trusted evaluation methodologies. Even worse, a biased evaluation could be misleading for both developers and users.

Before discussing existing IDS evaluation techniques, let us first recall some important definitions. The aim here is to clarify common ambiguities and to help understanding this chapter.

The term *Evaluation* is generally defined as “the act of placing a value on the nature, character, or quality of something” {Webster}. Evaluation may take several forms, use different techniques and aim at various goals. Considering the techniques applied to evaluate computing systems¹³ {Jain91}, we can cite three main techniques:

- 1) *Analytical evaluation*: is usually based on some abstract model of the system under study (not the system itself) and can be performed at any stage during the development cycle.
- 2) *Simulation*: is defined as “the imitative representation of the functioning of one system or process by means of the functioning of another. It usually refers to: (a) computer simulation of an industrial process, (b) examination of a problem often not subject to direct experimentation by means of a simulating device” {Webster}. This technique is applicable at any stage when system behavior, interactions between system components and system inputs/outputs can be represented in the simulation,
- 3) *Test or measurement* by which an actual implementation or a prototype of the system is evaluated against real or synthetic inputs (workloads or test dataset) in order to study system behaviors and reactions. *Benchmarking* is a specific kind of test: it is the process of obtaining representative measurements to compare two or more systems. Alternatively, a series of experiments can be performed on one system using a reference set of benchmarks (datasets/programs).

Each of these three techniques involves the presence of three elements, a *Target of Evaluation (ToE)* (i.e., the system to be evaluated), an *evaluation methodology* that provides a detailed plan describing all the steps of the evaluation process and some *metrics* that correspond to the criteria used to evaluate and judge the system. Any metrics should have both a definition and a unit of measure. For example, the

¹³ By “computing system”, we mean any computer system or computer-based system that is comprised either in software, hardware or in both forms.

throughput of CPUs is a metric defined as the rate (i.e., number of instructions per unit of time) at which instructions can be executed; the unit of measure is MIPS (Millions of Instructions per Second).

The term *measure* (noun) has several different meanings: in metrology and performance evaluation literature, it refers to the output of the measurement process, whereas quite often in security it represents means or mechanisms used to enforce security, such as firewalls, IDSs, etc. To avoid any confusion, we prefer using the term *countermeasure* for security mechanisms while keeping the term “*measure*” for its original use in metrology or use it as a verb that stands for the action of taking measures or measurements. In the case of empirical evaluations (i.e., test or benchmarking), two more elements are also involved: a *Test-bed* or *workbench* and test datasets. The test-bed comprises the platform of test (software/hardware/network architecture) on which the test will be carried out, and the test dataset consists of the inputs that will be fed to the evaluated system (i.e., the ToE).

3.2. An Overview of Existing IDS Evaluations

According to the definitions stated above, most of the previous IDS evaluations such as the evaluation that was carried out by Puketza *et al* at University of California-Davis {Puketza96}, {Puketza97}, the early evaluations by IBM Zurich {Debar98} or the evaluations DARPA98 {Lippmann00a} and DARPA99 {Lippmann00b} are considered as evaluations by test or simply IDS testing.

By contrast, the evaluation of Alessandri that is described in {Alessandri04} is an evaluation by analysis. As far as we know, this is the only analytic evaluation found in the literature of IDS.

In addition to that, there are magazine evaluations and other evaluations carried out by vendor-independent laboratories {Nss08}. In this section, we will focus more on “academic” evaluations rather than “commercial” evaluations, because more information is usually available in technical reports or research papers published by academic evaluators. In the following, we will present a brief description of the most significant evaluations. First, test evaluations will be briefly described: their procedures, metrics, workloads (attacks and normal activities), test case selection, etc. Then, the analytic evaluation will be described to show its strengths and shortcomings when compared to test evaluations. Then, we identify the problems and the common mistakes found in existing IDS evaluations of both types.

3.2.1. Evaluation by Test

1) Evaluation by University of California-Davis (USA)

Even if there have been some unpublished comparison experiments between several early IDS prototypes, the test made by Puketza *et al* at California-Davis University {Puketza96} is, to our knowledge, the first published IDS evaluation. The authors claimed that the selection of test cases was based on the organization’s security policy. The test procedures were crafted to address three performance objectives: intrusion identification, resource usage and stress testing. They took into account both sequential intrusions that are executed over a single session and concurrent intrusions originating from several sessions either from the same attacking machine or from different attacking machines.

A simple software platform based on *TCL-DP*¹⁴ programming package and *Expect*¹⁵ package was developed to launch attack sessions automatically by means of a limited set of scripts that simulate a number of selected test cases for both normal and intrusive sessions. Besides, some highly interactive attacks and sessions for GUI applications were executed manually.

2) IBM Evaluation

Another real-time test was carried out by IBM Zurich research Division {Debar98}. Its goal was mainly to create a generic test-bed suitable for comparative evaluations of IDSes using several client and server machines controlled by a single workstation. The authors found that modeling users’ behavior by using generic session generators would be more complex. Therefore, they decided to generate the background traffic by using test suites developed by operating system developers and pre-recorded “live

¹⁴ *TCL-DP* is an extension of *TCL/TK* programming language. It provides a suite of commands for creating client-server systems.

¹⁵ *Expect* package provides commands for controlling interactive programs.

data". Attack test cases were selected from the vulnerability database, which is maintained internally by IBM. The published report {Debar98} indicates that only host-based IDSes (HIDS) were tested and only against FTP attacks. The workbench was developed using scripts written in *Expect* and *Perl* languages to record user sessions. Four *HIDS* were compared but unfortunately, the report details neither which metrics were used nor which results were obtained.

3) DARPA Evaluations

In 1998 and 1999, DARPA sponsored an ambitious project for IDS evaluation in cooperation with MIT's Lincoln laboratory. These evaluations are known as DARPA 1998 and DARPA 1999 evaluations. Both DARPA evaluations share the main objective: to provide a dataset or "*corpora*" for testing and comparing IDSes and to analyze their strengths and weaknesses easily.

To achieve this, a network test-bed was implemented to create live traffic, which contained various traffic types similar to what may be generated by hundreds of users on thousands of hosts. Seven weeks of training data, containing background traffic and labeled attacks, plus two weeks of unlabeled test data were recorded.

The background traffic was synthesized according to statistics collected from computer networks in several air force bases (about 50 air bases). The attack part of the dataset was generated by attack scripts collected from specialized sites and mailing lists on the Internet or written by hand. In addition to that, some live attacks were executed manually during the evaluation.

As an evaluation criterion, two metrics were defined and used: *Detection Rate* and *False Alarm Rate*. The results were presented in the form of *Receiver Operational Curves (ROC)*. The ROC curves were initially used in domains and applications concerned with signal detection such as communication and radar and then applied successfully to other fields. To draw the *ROC* curve, the *false alarm rate* is plotted on the horizontal axis and the *detection rate* on the vertical axis, as illustrated in Figure III.1.

DARPA evaluations were criticized on several points {Mell03}, {McHugh00a}, {Zanero07}, which can be summarized in three groups: 1) critiques related to dataset generation (background and attack), 2) those related to metrics and 3) the presentation of results by ROC curves. As we will see later, these critiques similarly hold for other evaluations and in some cases more severely.

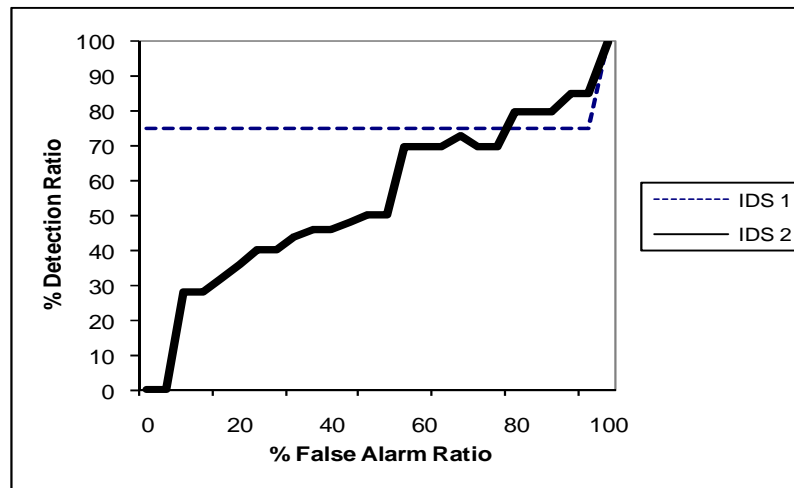


Figure III.1: An example of a ROC curve for two IDSes.

Beside DARPA evaluations, the DARPA datasets have been frequently used by IDS evaluators as a *de facto* standard dataset. Because of the importance of DARPA evaluations and the impact they had on subsequent evaluations, we will further discuss the characteristics of their datasets in more details in Chapter 4.

4) Evaluation by University of Notre Dame (USA)

The aforementioned evaluations have often focused on the quality of the detection provided by IDSes. On the contrary, the evaluation made by Schaelicke *et al* {Schaelicke03} is an effort to study the performance requirements of signature-based NIDS sensors across a variety of platforms. Performance

bottlenecks were then attributed to specific system features. In particular, they measured the effects of the number of rules (signatures) and the packet size.

The results showed that the required processing load is constant for header-related rules but are very payload-dependent for payload-related rules. In both cases, processing load is highly dependent on NIDS algorithms. Moreover, the results showed also that a combination of factors contributes to the overall performance. In particular, memory, bandwidth and interrupt handling mechanisms of the OS have a more significant effect than processor speed.

5) Evaluation by France Telecom

As explained in {Debar02}, the authors aimed to evaluate the diagnostic capabilities of network-based IDSes. Four commercial NIDSes were compared with each other and with respect to *Snort* as a baseline. The comparison was established on measured false positive and false negative rates. Attack test cases included IP denial of service attacks, Trojans and various HTTP-based attacks. The HTTP attacks were generated by the Whisker tool {Whisker08}, which allows the application of various evasion techniques. The importance of this evaluation stems from its result as it showed the very poor diagnostic quality of the evaluated IDSes and surprisingly *Snort*, which is an open source tool, outperforms commercial IDSes in several tests.

6) Maxion's Evaluation

Maxion *et. al.* have evaluated probabilistic algorithms that are often used in anomaly-based IDS {Maxion00}. The goal of this evaluation was to test the hypothesis that the intrinsic regularity of the dataset influences the performance of probabilistic-based anomaly detectors. The conditional relative entropy was defined as a metric for characterizing the structure of data environments.

A series of experiments was carried out on an anomaly-detection algorithm using a suite of 165 anomaly-injected datasets of varying structures. Results showed a strong relationship between detector accuracy and regularity. The authors observed that the false-alarm rate rises as the regularity index grows (i.e., the data become more and more random). They concluded that, in contrast to current practice, an anomaly detector should not be evaluated by using a single dataset of the same regularity because it behaves differently on datasets of different regularities.

3.2.2. Evaluation by analysis

Ideally, this technique consists in analyzing some kind of model that describes the behavior of the evaluated system (ToE). It does not require the ToE to be really implemented and can be applied during the early stages of IDS development. The advantage of this approach is that we can avoid the difficulties that inherently exist in IDS testing (e.g., construction and generation of test datasets). The only work that can be cited here is the one performed by Alessandri {Alessandri04}.

In his PhD dissertation, Alessandri states the main goal of the evaluation as being to "*provide guidance to IDS designers by predicting the detection capabilities of intrusion detection systems*", {Alessandri04}. Thus, instead of testing or analyzing a behavioral model of the IDS, a descriptive-model was depicted and analyzed.

Indeed, this evaluation is carried out through: 1) Classifying attacks according to their characteristics that could be observed by an IDS; 2) Describing the IDS in terms of its characteristics and particularly those related to the way by which the IDS gathers and analyzes the information; 3) Describing attack classes in terms of the IDS characteristics that are necessary for detecting a given type of attack; 4) Once attack classes and IDS are described, simple inspection of both can determine whether a given type of attack will be detected by the evaluated IDS or not and decide accordingly whether "*generalized alarms*" will be generated or not. *Generalized alarms* are described separately by determining the necessary and sufficient conditions that are required for an alarm to be generated {Alessandri04}.

Attack variations were also considered by applying some predefined variation rules on attack classes that may result in new attack classes. Attack classification and attack selection was based on *VulDA*, the vulnerability and attack database maintained internally by IBM {Dacier99}. Noting that *VulDA* is not publicly available for IBM outsiders, classification and description of attacks, which is a principal step in this approach, is a nontrivial task.

This technique is well adapted for "white-box" analysis but it is not suitable for black box evaluations where IDS internals are unknown for the evaluators. Moreover, as one can expect, an IDS is

examined only against attacks without taking into account background activities and environment-specific effects. Therefore, the performance of an actual implementation of the same IDS may significantly deviate from the predicted one. Deviations may also arise from implementation flaws or bad configurations. However, analytic evaluations give deeper and clearer insight of the expected behavior of IDSes and hence we can obtain a better comprehension of IDS behavior.

Table III.1 summarizes the main features of the evaluations just presented above. In the next section, we will discuss the common mistakes related to these evaluations.

Table III.1: A comparison of IDS evaluations.

	Evaluation Technique	Test bed & environment	Metrics	Workload	
				Attacks	Background
University of California-Davis	Real-time Testing	A software platform to generate attack by scripts written in "expect" and TCL-DP	No clearly defined metrics	Single and multi session attacks and concurrent attacks	Simulated non intrusive user sessions
DARPA 1998	Off-line Testing	<ul style="list-style-type: none"> A test bed with an emulation of thousands of workstations & websites (external) and hundreds of emulated PCs and workstations (internal) Attacks against UNIX victim hosts (SunOS, Solaris, Linux) Attack generator based on "expect" 	<ul style="list-style-type: none"> Detection Rate False alarm rate Detection + False alarm + ROCS 	<ul style="list-style-type: none"> Seven weeks of training data that contain background data & labeled attacks. Two weeks of unlabeled test data 	
				<ul style="list-style-type: none"> 300 instances of 38 different attacks from outside Novel Unix attacks Stealthy attacks Attack classification: <ul style="list-style-type: none"> A.1 Probe A.2 Remote-2-local A.3 User-2-Root A.4 DoS B.1 abuse of legal B.2 bug B.3 masquerading Attack selection: Available attacks & attack scripts from the Internet. 	<ul style="list-style-type: none"> Solaris audit data
DARPA 1999	Off-line and Real-time testing	<ul style="list-style-type: none"> Same as DARPA 98 + Windows NT Victim 	Same as DARPA 98 + <ul style="list-style-type: none"> Identification Error Analysis 	<ul style="list-style-type: none"> Three weeks of training data. The first and the third weeks do not contain attacks. The second week contains background data & labeled attack. two weeks of unlabeled test data Stealthy attacks 	
				<ul style="list-style-type: none"> 201 instance of about 56 types of attacks inside & outside Win NT attacks, Unix attacks 	<ul style="list-style-type: none"> Solaris & NT audit data False alarm analysis on actual AF traffic
IBM Zurich	Real-time Testing	<ul style="list-style-type: none"> Several machines as clients and servers + workbench controller Linux, Sun AIX 	Not available	<ul style="list-style-type: none"> Scripts from internal Vulnerability Dbase FTP attacks 	Test suites developed by operating system developers and pre-recorded "live data"
France Telecom	Off line testing	<ul style="list-style-type: none"> Snort as baseline product and other four commercial NIDS 	False positive and false negative rates	<ul style="list-style-type: none"> IP denial-of-service attacks, Trojan horse, and various HTTP-based attacks 	Profiled network traffic
Alessandri	IDS & Attack- class Analysis	<ul style="list-style-type: none"> No testbed Environment independent 	Generalized alarm	Attack-class description	No Background

3.3. Common Critiques of Previous Evaluations

About 20 mistakes can be found in most computer systems performance evaluations {Jain91}. When we review these mistakes, we discover, surprisingly, that several of them are also present in most IDS evaluations and testing {Mell03}, {McHugh00a}, {Maxion98}. To be objective, some of the mistakes are difficult to eliminate because of the complexities inherent in IDS evaluations but we claim that most of them can be avoided or at least their effects can be attenuated. We cite, hereafter, the most relevant among these mistakes.

The first mistake is *using unsystematic approaches*. One can notice that most IDS testing approaches are ad-hoc and the selection of system parameters, factors, metrics as well as evaluation datasets is often arbitrary.

The second mistake is the use of *non-representative workloads or attack test cases*. Regarding intrusion detection systems, the workload consists of two components: *background dataset* (that is normal background network traffic in case of NIDS and normal system activity or system events for HIDS) and *attack dataset* (malicious and intrusive dataset). This mistake means that neither background data nor attack data correspond to those of the real world. Therefore, the evaluated IDS behaves differently when implemented in a real operational environment. For example, the packet rates found in the background traffic of DARPA datasets are much lower than what would be expected: a few Kbits/s whereas we would expect it to be in the range of hundreds of Kbits/s or even several Mbits/s according to the announced number of workstations and servers that the simulated network comprises.

The problem of generating network traffic has been addressed for a long time by network researchers, who defined and used traffic models to generate synthetic traffic with various characteristics and rates. Such early traffic generators have helped to evaluate and to improve the performance of network equipments and protocols. However, network researchers have been later confronted to a serious problem: simple distributions cannot model Internet traffic, due to its irregularity, self-similarities and burst phenomena {Paxson97-a}. Even when these models can describe successfully traditional communication traffic, they are unfortunately unsuitable for describing the Internet traffic.

Furthermore, poorly implemented network protocol stacks may generate many packets, which are legitimate but have strange characteristics, which thus could be interpreted by the IDS as intrusive traffic. On the other hand, some recent studies have proved that most IDSes are sensitive to packet payload contents {Antonatos04}, which means that simple traffic generators used for testing network devices like switches and router are less useful for evaluating Network-based IDSes.

Similar problems are raised by generating system audit records, system calls or logs as a background dataset for HIDS evaluation. For example, audit record and log formats differ from one operating system to another and this implies providing an audit dataset for each OS. In addition to that, audit records should be validated to ensure that they represent or closely approximate the reality. Although this is also true for NIDS evaluation, it is more significant in evaluating HIDS.

Regarding the attack component of the dataset, it is difficult to construct test cases for all known attacks and impossible to test IDS for all potential unknown attacks. For this reason, the sample of attacks selected for test or evaluation should cover a wide range of known attacks and should take into consideration the unknown attacks that may appear in the future. Otherwise, the results of test or evaluation would be biased towards those attacks that are included in the dataset. A potential solution can reside in a good classification of attacks that can aid in selecting relevant and representative attack test cases. However, generating attacks is still difficult and time consuming.

Testing anomaly-based IDS, in its turn, is troublesome. This type of IDS needs often to be trained before being in use. Therefore, the training dataset is as important as the test dataset. Unfortunately, it was found that performance of anomaly detection algorithms (probabilistic, statistical, neural networks, decision trees, etc.) is highly sensitive to environment variations. Consequently, the nature and the degree of regularity of either training or test datasets will have an important impact on the performance of the tested IDS {Maxion00}.

Another mistake is the *use of incorrect metrics*. Some people tend to select metrics that are easy to measure without regarding to what extent they will honestly reflect the actual IDS behavior. Worse, it may be intentionally tailored in a certain way to show that one system appears better than another. There are several metrics defined for IDSes such as *detection rate*, *detection ratio*, *false alarm rate*, *false alarm ratio*, *expected cost metrics*, etc. We only give an example of inappropriate or incorrect metrics that was

mentioned in {Mchugh00b}. False alarm rate has several definitions that differ in denominators. It can be defined as the number of false alarms divided by the number of sessions, or divided by the number of packets. We agree that the quantity of the analyzed or processed data (in the form of sessions or packets) may be relevant to illustrate the false alarm characteristics of an IDS. However, in practice, the credibility of alarms, i.e., the proportion of false alarms among all raised alarms, is more important. In that case, even if the rate of false alarms depends uniquely on the “normal” traffic (and its quantity); the frequency of true positives (i.e., real attacks that are detected) should also be taken into account.

The identification of these main mistakes in previous IDS evaluations has motivated us to propose a new framework for IDS evaluation, which will be described in the rest of this chapter.

3.4. Evaluation Framework

By creating such a framework, we aim to provide methods and tools that can aid in avoiding the mistakes and the limitations previously discussed in this chapter. The overall goal is to render IDS evaluations systematic, with representative datasets, a set of comprehensive metrics and effective methods for selecting attack test cases.

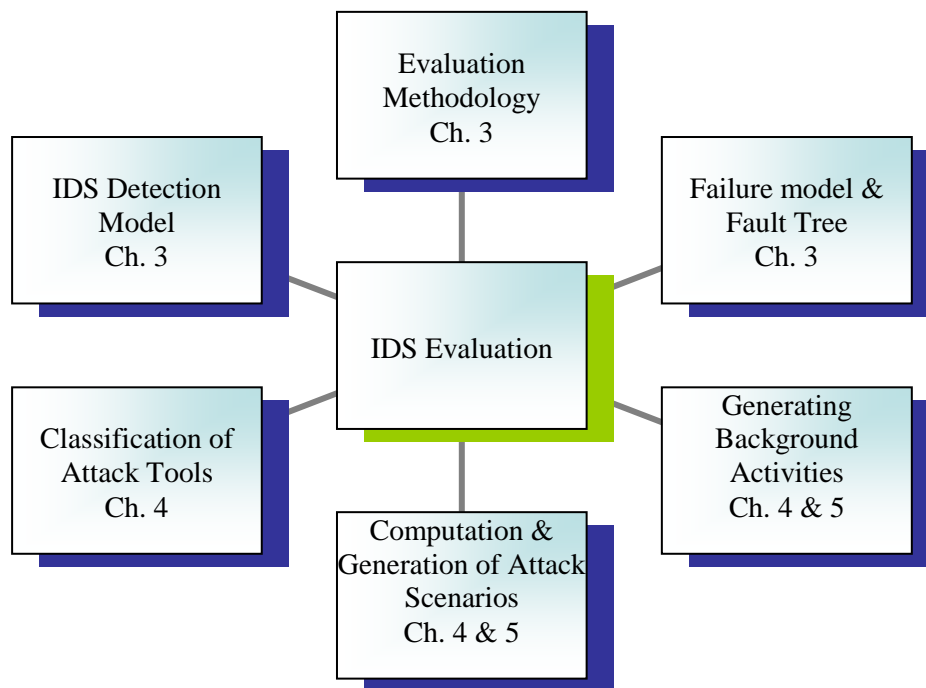


Figure III.2: The foundations of the evaluation framework.

In fact, to achieve that and to improve the diagnostic capability of the evaluations, we have based our approach on six theoretical foundations, as shown in Figure III.2:

- 1) An evaluation methodology that provides guidelines for engineered IDS evaluations (see 3.5).
- 2) A detection model: we have established a generic detection model for IDSes and defined the related parameters in terms of events that can be observed, processed and analyzed by the IDS (see 3.6).
- 3) A failure model and a fault tree model: to determine situations in which an IDS fails to provide its expected service (i.e., the detection). We established an IDS failure model, which is combined with the fault tree model to provide diagnostic capability to the evaluation (3.7).
- 4) A classification of attack tools: we have thoroughly analyzed existing attack tools and consequently we have developed a classification scheme on those attack features that seem to be significant from the IDS viewpoint (see 4.4). This classification provides a solid theoretical basis for the selection of attack test cases. It also allows the presentation of evaluation results in an organized manner with respect to attack classes or features.
- 5) Scenarios computing and generation methods: by analyzing human-centric attacks and attack procedures followed by automated malware attacks, we managed to extract a model for attack

processes (see 4.5). We also created a model for attacker behaviors (see 5.2). Thanks to the attack process model and by using constraint programming, we can calculate abstract attack scenarios. These abstract scenarios can be transformed into executable scenarios in function of the attacker competence model to generate representative malicious activities (see 5.3).

- 6) Background generation method: first, we have determined the security-relevant characteristics of the background activities (see 4.8), then, we have implemented a tool for constructing background dataset that edits and manipulates network traffic traces while maintaining these characteristics.

These bases comprise our contribution to the evaluation of intrusion detection systems. We will describe each basis in the same order. The evaluation methodology and the IDS model are explained in the rest of this chapter. Because classifying attack tools and computing attack scenarios — which are closely related — are critical to our approach, we will describe them separately in Chapters 4 and 5 respectively.

3.5. An Engineered Evaluation Methodology

To follow a systematic approach and to manage the complexity of the evaluation process, we have proposed a simple methodology that should be ideally followed by evaluators of intrusion detection systems {Gadelrab06}. The main objective is to obtain a systematic evaluation performed through well-structured steps.

As shown in Figure III.3, this can be achieved in two stages: a preparation stage (stage 1) and an experimentation stage (stage 2). The second stage is straight forward as it encompasses the empirical part in the same way as traditional evaluations. Thus, we concentrate on the first stage, which begins by stating the goal of the evaluation and by identifying the expected needs of users. Second, the main characteristics of the IDS Target of Evaluation (ToE) itself should be determined. Third, the evaluator should recognize the main characteristics of the environment where the tested IDS will be deployed and operated.

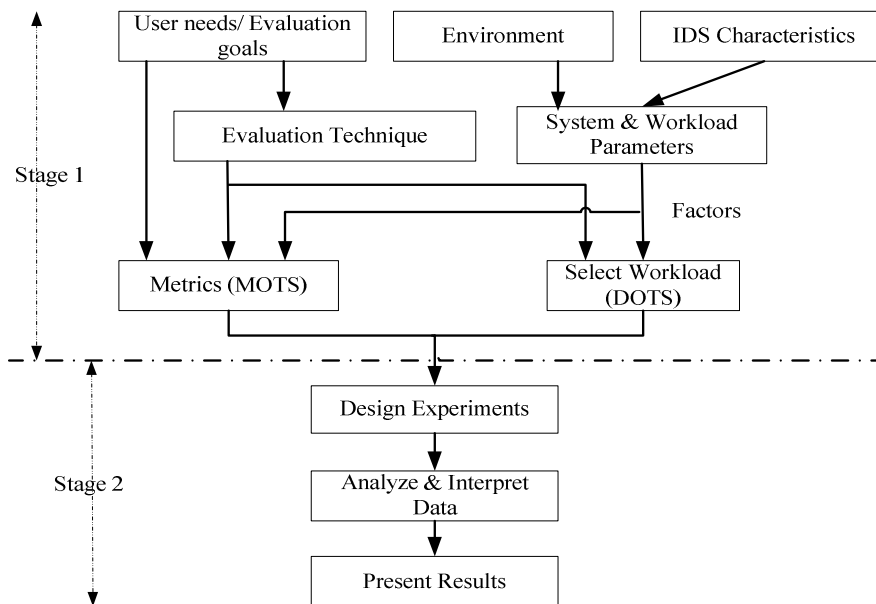


Figure III.3: IDS evaluation methodology.

When completing these steps and according to the goal of the evaluation, the evaluators can select an appropriate evaluation technique. They can also determine the parameters of both the IDS and the workload. However, since these parameters are not all controllable and modifiable by evaluators, we consider only factors (i.e., parameters that can be changed). The identified factors and the selected evaluation technique affect the selection of metrics and the construction of workloads (i.e., evaluation datasets). In the next subsections, we will explain in more details the different steps of our methodology.

3.5.1. User Needs and Evaluation Goals

There are many features desired by users in their IDS. For any IDS, we distinguish three types of characteristics: functionality, performance and usability. While functional features include those related to detection efficiency, performance characteristics refer to the traditional meaning of “performance”: they cover the aspects related to speed, memory utilization and CPU load. Usability features concern issues such as the ease of use, the ease of configuration, the ease of maintenance and the availability of documentation. Users’ wish list or needs are usually translated into IDS requirements. For instance, an IDS should ideally:

- have perfect detection (detect all known or still unknown attacks),
- generate no false alarms,
- be resilient against attacks targeting IDS itself as well as DoS attacks affecting the network or hosts where the IDS is run,
- cause low overhead (low resource consumption),
- be reliable.

Evaluation goals may be different too. For example, some IDS users may simply want to make a comparative evaluation to select an IDS that best fits their needs, or may want to assess the efficiency of an IDS actually installed on their systems. By contrast, developers usually may want to calibrate their parameters or to improve their performance and efficiency. Therefore, they need detailed information to interpret and understand the behavior of their IDS. Clearly, if a black-box evaluation could be useful for IDS users, it would be inadequate for IDS developers.

3.5.2. The Environment

Regarding differences between environments, let us consider typical networks in academic, military and commercial domains. They exhibit different policies, types of traffic, services, etc. Moreover, security objectives are not of the same importance. Confidentiality is crucial for a military network with a stringent security policy, while integrity and availability may be primordial in a commercial network. As another example, gigabit networks may require specific issues to be examined, such as packet dropping. Moreover, important assets like servers, files, databases, etc., and platform characteristics such as operating systems, applications and software versions, all these should be taken into account for the evaluation of intrusion detection systems. More precisely, attack test cases must be selected as relevant as possible, according to the targeted environment. Background datasets should be made of flows of events that match the normal activities in the environment: e.g., on a particular commercial network, web traffic may be dominant, whereas burst data transfers could be more important on certain military networks.

3.5.3. The Characteristics of the IDS under Evaluation

On the architectural level, IDSes are not all identical. They are no longer those monolithic programs installed on a single machine that handle all the tasks: capturing, normalizing and analyzing event data, and finally generating alarms. Sensors, preprocessors and detectors could be distributed and deployed on several hosts on the network. A wide variety of techniques and algorithms are integrated within IDSes. Detection algorithms have been implemented in several ways such as expert systems, neural networks, genetic algorithms, etc. Agent-based IDSes are still different, with agents that have more or less intelligence, autonomy and mobility, thus requiring different issues to be evaluated. Regarding all these differences, an evaluation procedure suitable for some type of IDS may be inappropriate for another.

3.5.4. System and Workload Parameters

A complete control over the computational and networking environment is impossible. Evaluators and testers can only adjust and change the controllable parameters, i.e., the factors {Jain91}. However, not all parameters or factors affect significantly the behavior of the System Target of Evaluation (ToE). The identification of the interesting parameters and factors for both the system and the environment help to observe and understand their influence on the tested system. For example, considering the workload required as a dataset for testing a network-based IDS, we can identify the following potential factors: traffic composition with respect to protocol types, packet lengths, payload contents, bandwidth

utilization, etc. Similarly, workload factors for a host-based IDS might be: operating system (platform, version, etc.), applications and services running on the machine, etc.

Amongst the factors related to the IDS itself one can find signature rules, detection algorithm, architecture, etc. On the other hand, learning algorithms and profile thresholds are two examples of potential factors for anomaly-based IDSes.

3.5.5. Evaluation Technique

The choice of an appropriate evaluation technique (i.e., evaluation by test or analytic evaluation) is highly dependent on the goal of the evaluation and on the stage at which it is performed. Whereas an evaluation by analysis could be carried out from early stages of design and development, evaluation by test is not possible before the implementation phase. Moreover, evaluations by analysis require a good knowledge of the internal structure of the IDS and of how its components work. Consequently, it is not suitable for black box evaluation for which evaluation by test is better fitted. Table III.2: summarizes the main features of analytic evaluations and evaluations by test.

Table III.2: Main features of IDS evaluation techniques.

	Test	Analytic evaluation
Place in IDS life cycle	Lately, not before the implementation of at least a prototype of the system.	Possible from the early phases: specification and design
Target	A prototype or an implemented IDS	A model of IDS
Input	A real or synthesized dataset	A model of attacks
Background normal activities	Can be considered	Usually not considered
Evaluated features	Detection capabilities and performance issues	Detection capabilities only
Effects of Environment	Can be considered	Not considered
Required Knowledge level	Knowledge about IDS internals is not necessary; suitable for black-box evaluation	A good knowledge of the structure and the design of the evaluated IDS.

3.5.6. Selecting the Evaluation Workload

Careful selection and construction of the workload (i.e. the dataset consisting of intrusive and normal activities) governs the quality of the evaluation. The major considerations in selecting workload datasets are:

1. which IDS functions or services will be exercised by the workload. For example, the capability of a NIDS to detect fragmented attacks,
2. which level of details should be assigned to the evaluation dataset (e.g., it was discovered that some detection algorithms are highly sensitive to payload contents {Antonatos04}),
3. representativeness: the characteristics of the test workload should match those of the real workload,
4. temporal characteristics: the distribution of the test workload over time should reflect the real distribution,
5. repeatability: a workload should be such that the evaluation can be easily reproduced without too much variance in the results.

It is expected that the previously discussed steps of the evaluation methodology will affect and help to refine the selection of evaluation datasets and the design of experimentations. For example, some users search for an IDS that is efficient in detecting a particular type of attacks (a specific user's need), then datasets should cover more specially these attacks. Similarly, a developer who comes to add a new type of sensors (IDS characteristics) would probably be more interested in testing the new features of these sensors. Thus, the number of experimentations and test cases should be more focused than for a general IDS evaluation.

In the next chapter, we will discuss the characterization of IDS workloads in the real world and explain how a representative evaluation dataset can be constructed.

3.5.7. Selecting Metrics

Defining appropriate metrics is a corner stone of any evaluation processes. Without well-defined metrics, conclusions based on the evaluation results would be biased or might be completely wrong.

Actually, several metrics have been defined for IDS, including detection rate, detection ratio, false alarm rate, false alarm ratio, expected cost metrics, etc. Rather than adopting particular metrics a priori, we believe that metrics should be defined under the following assumptions:

1. There is no absolute metrics but relative metrics with respect to selected test cases (normal activities, attack classes) within the dataset.
2. Expressing the evaluation results by a single number or metrics (e.g., a ROC curve) is not sufficient to present results.
3. We should search only meaningful and measurable metrics and we should avoid meaningless, measureless or too generic and ambiguous metrics (e.g., the false alarm rate, as discussed in Section 3.3).
4. A defined metrics may be more or less important to the evaluators depending on their goal and on the ToE.

In the metric set that we suggest in Table III.3, we make a distinction between detection-related metrics and resource utilization metrics or performance-related metrics. Furthermore, detection-related metrics are further divided into macroscopic metrics (system-level metrics) and microscopic metrics (component-level metrics).

The steps of the second stage of the methodology, i.e., experiments and result analysis, correspond to the empirical part of the evaluation. Because they are self-explanatory, they will not be described here, and we will describe in the next few sections the other elements of the evaluation framework, beginning by the detection model of the IDS.

Table III.3: Example of suggested metrics.

Detection Related Metrics	Definition
Macroscopic:	
Detection Ratio	DR= (Number of detected attacks/ Total number of attacks included in the dataset)
False Alarm ratio	FAR= (Number of generated false alarms/ total number of generated alarms)
Microscopic:	
Detection Ratio per attack Type	Number of Detected attacks of a particular type/ total number of attacks of this type
False Alarm Ratio per Attack Type	Number of generated false alarm for a particular attack type/ total number of generated false alarms
Captured Events/Non Detected Attacks	Number of undetected attacks whose events are captured / Total number of undetected attacks (pro-sensor failure)
Non Captured Events/Detected Attacks	Number of attacks whose events were not captured / total number of undetected attacks
Intrusive Events Drop Ratio	Number of non captured intrusive events /Total number of intrusive events
Resource-Utilization Metrics:	
CPU Utilization	Percentage of CPU used by IDS
Memory Utilization	Percentage of memory used by IDS

3.6. A Detection Model

We need to define the boundaries of the evaluated IDS to determine explicitly what belongs to the ToE and what lies outside. A detection model of IDS can help in this regard. Moreover, such a model helps us to characterize the IDS (since this is a primordial step in our evaluation methodology).

As we have adopted the structural model shown in Figure III.4, we can accordingly define the properties that an IDS must have to provide a good functioning. We do that in terms of attack events with

respect to four main issues: *Visibility*, *Analyzability*, *Detectability* and *Alertability*, which we called the VADA Model. It represents the least necessary functions to detect an attack.

Definition 1 (Attack event): E_a is the set of atomic events e_a produced by the attack A , that occur in sequence or in parallel, where each attack event e_a can be a malicious event e_{am} ($e_{am} \in E_{am}$) or a benign (i.e., apparently normal) event e_{an} ($e_{an} \in E_{an}$).

$$E_a = \cup e_a \text{ with } E_a = E_{am} + E_{an} \text{ and } E_{am} \cap E_{an} = \emptyset$$

Definition 2.1 (Visibility): Given a set E of atomic events captured by an intrusion detection system (*ids*), an attack A is *visible* with respect to this particular *ids* if and only if all of its own atomic events e_a (either malicious or benign) are captured by at least one sensor of the *ids*. Then:

$$A \text{ is visible IFF } \forall e_a \in E_a, e_a \in E$$

where E is the set of all events captured by all the sensors: $E = \cup E_{sensor}$

Definition 2.2 (Partial Visibility): An attack A is *partially visible* if at least one of its own malicious events were captured by at least one sensor of the *ids*. Then

$$A \text{ is partially visible IFF } \exists e_{am} \in (E_{am} \cap E),$$

where e_{am} is a malicious event that has been produced by the attack A .

Definition 3.1 (Analyzability): Given a set E of atomic events captured by an intrusion detection system (*ids*), an attack A is *analyzable* by this particular *ids* if: 1) it is visible (see definition 2.1), 2) its events can be handled by the detector.

The second condition implies that all attack events (either malicious or benign) have been captured and have not been omitted by the preprocessing:

$$E_a \text{ (after preprocessing)} = E_a \text{ (before preprocessing)}$$

Definition 3.2 (Partial Analyzability): An attack is *partially analyzable* if at least one malicious event is delivered to the detector, even if some of attack events have been omitted. This means that A is partially analyzable IFF:

$$\exists e_{am} \in E_a \text{ (after preprocessing) and } E_a \text{ (after processing)} \subseteq E_a \text{ (before preprocessing)}$$

Definition 4 (Detectability): Given a set of atomic events E captured by an intrusion detection system (*ids*), an attack A is *detectable* by this particular *ids*: 1) if it is *visible* or *partially visible*, 2) if it is *analyzable* or *partially analyzable*, and 3) if the detection unit of the *ids* can recognize at least one of its own malicious events or a malicious sequence of benign events.

Definition 5 (Alertability): Given a set of atomic events E captured by an intrusion detection system (*ids*), an attack A is *alertable* by this particular *ids* if it is *detectable* by the *ids*, and if the reporting unit can generate at least one alert corresponding to the attack malicious events or malicious sequences.

3.7. IDS Failure Model

Generally, according to {Avizienis04}, a *failure* is an event that occurs when the delivered service deviates from correct service. An *error* is the part of the system state that may lead to its subsequent service failure and a *fault* is the adjudged or hypothesized cause of an error. Regarding the context of IDS evaluation, we consider in particular the following *faults*:

- implementation or configuration flaws: for example, the lack of signatures or anomaly model for known attacks, a bug in the detection algorithm, a bad configuration parameter, etc.,
- intentionally malicious activities (e.g., DoS, illusion and evasion techniques) mainly intended to bypass the IDS or disable it,
- benign, non malicious activities that affect the efficacy of the IDS, e.g., system overload.

Considering *errors*, the above faults may produce errors within an IDS component and hence a component failure. An error may propagate to further produce failures/errors in other components until it causes a failure at system level.

An *IDS failure* occurs when the IDS fails to deliver its expected service (i.e., to detect attacks). In other words, if it generates a false alarm, does not detect an attack (i.e., a false negative) or if it badly identifies the detected attack. An IDS failure may be also a *security failure*, i.e. the violation of a security property of the intended *security policy*. This includes any violation of the *confidentiality*, the *integrity* or the *availability* [Maftia03].

A *False negative* is an IDS failure corresponding to the occurrence of an error that may lead to security failure and that is not detected as such. This means that no alarm is raised – also called a *miss*.

A *False positive* is an event corresponding to an alarm generated in the absence of any error that may lead to a security failure – also called *false alarm*.

A *True positive* is an event corresponding to the correct decision to rate an activity as being malicious – also called a *hit*.

A *True negative* is an event corresponding to the correct decision to not rate an activity as being malicious.

Figure III.4 shows the failure model of an IDS. According to this model, we will analyze the generic structure of IDSes, searching for the root causes of IDS failures. We begin with a qualitative analysis of IDS failures by applying a Failure Modes and Effects Analysis (FMEA), which allows identifying potential failures of each component and their causes. Further, we run a complementary Fault Tree Analysis (FTA), which facilitates the identification of component-failure combinations that may lead to an overall system failure.

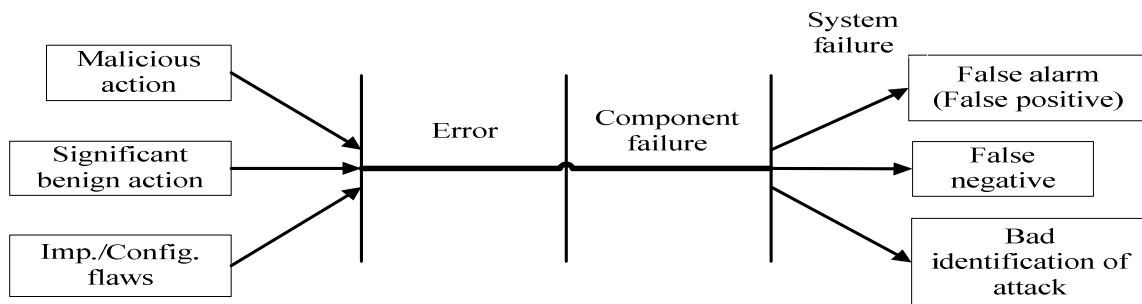


Figure III.4: IDS failure model.

3.7.1. Failure Modes and Effects Analysis

FMEA is an inductive method that analyzes the consequences of potential errors for each component in order to identify systematically the set of failure modes of this component and the consequences of these failures at the system level. FMEA is generally applied, after identifying components failure modes, by constructing a table that contains, for each failure mode, the following attributes: the probable causes, its effects, the methods used to detect this failure and corrective actions [Avizienis04], [Bouti94].

Table III.4: Generic FMEA for IDS.

Component	Failure Mode	Causes	Effects
Sensor	Not capturing intrusive events	1- Out of scope malicious events 2- Intentionally masked events	Intrusions become completely invisible false negative
		3- Dropping events	Intrusions become completely or partially invisible false negative
Preprocessor	Suppression of useful information	1- Inappropriate format 2- Insufficient information supplied by the sensor	Detector failure
Detector	Unable to detect captured intrusive events	1- Preprocessor failure 2- Detection algorithm failure	False negative
	Considering non intrusive events as intrusive	Detection algorithm failure	False positive
	Bad identification	Detection algorithm failure	Incorrectly reported attacks
Reporter	No generated Alarm or wrong alarms	Bad configuration	False negative

In Table III.4 we have constructed a generic structure-based FMEA for intrusion detection systems, where failures, probable causes, and effects for each component are identified.

3.7.2. Fault Tree Analysis

Fault Tree Analysis is a deductive method that allows searching for combinations of events that may lead to an undesirable event (i.e., a system failure). It is often used with FMEA in a complementary manner, where it considers failure combinations that escape from FMEA. A Fault Tree Analysis (FTA) is conducted by constructing a tree of consecutive levels of events (nodes) connected by logic gates (AND, OR, etc.). The undesirable event is situated at the root level. The tree is constructed by composing each event, starting from the root until we obtain elementary events that cannot be further decomposed.

Fault tree analysis is based on the calculation of the minimal cut-set. A cut-set is the set of elementary events that, when they occur, will lead to the occurrence of the undesirable event at the root. A cut-set is minimal if it is independent and does not contain any other cut-set. The study of these minimal cut-sets illustrates the critical events that cause the occurrence of the universal undesirable events {Avizienis04}. By contrast, while fault tree analysis was suggested for analyzing IDS requirements {Helmer01}, we use it in the following section to analyze the failures of the IDS itself.

A) Constructing Generic Fault Tree for ID Systems

Ideally, an IDS is supposed to detect and identify all attacks and generates no false alarm. It fails if when it does not detect attacks or when it generates false alarms. Another failure – however less severe – occurs if it detects the occurrence of an attack/intrusion but cannot identify it correctly. While the Fault Tree that we constructed is a generic one, it could be instantiated to describe a particular IDS. Moreover, it can be extended by adding implementation specific issues in order to be more comprehensive.

Figure III.5 shows the fault tree for a signature-based IDS. The undesirable root event, an IDS failure (i.e., a false positive or a false negative) may be due to a sensor failure, a preprocessor failure, a detector failure or a reporter failure.

According to Table III.4, a sensor fails if it cannot capture events. Usually, events pass completely unseen if they are out of the scope of this type of sensor. It could be also unseen by the IDS, if attackers manage to hide events or change their appearance to mask their intrusiveness. However, the invisibility of events is not the only way sensors fail. They also fail if they are overloaded to the degree at which they can no longer handle events and begin dropping them. The overload can be either a consequence of some malicious activity such as DoS attacks or due to normal activities at a rate too high for the sensors.

Regarding the preprocessor, it fails if relevant information about events is suppressed — assuming that the sensor had captured the events. The detector in its turn fails in three cases: 1) if it does not detect an intrusion whose events were captured, 2) if it identifies a normal event as an intrusive event or 3) if it does not identify correctly a detected attack.

It is valuable to note that the analysis made here is a qualitative analysis to find out and visualize combinations of events that may lead to IDS failures. Unfortunately, there is no sufficient available information on attack incidents to make quantitative analysis at this moment. Although a precise assessment of these probabilities might not be possible for now, future advances in the domain may lead to reasonable estimations of such probabilities (e.g., by using statistics extracted from data gathered by large scale Honeypot networks). A quantitative analysis might be performed in the future if, for example, we manage to obtain the probabilities of occurrences of basic events. We believe that this is feasible by deploying customized IDS on a large honeynet, with monitoring components able to account for IDS component failures.

B) Identification of Cut-set and Minimal Cut-set Nodes (Single Points of Failure)

Leaf nodes shown in Figure III.5 could be further developed until we reach basic events. For instance, the DoS in the sensor branch could be extended to include DDoS and to include attack types or attack instances that are used for DoS. However, we prefer keeping the generality of our analysis in order to be applicable for different IDSes and against different attacks.

To reduce the fault tree of Figure III.5 to a mathematical statement, we use boolean algebra (The “+” sign means OR and the “.” dot sign means AND):

$$\text{IDS failure} = A + B + C + D \quad \text{Where}$$

A = Event-capture failure

B = Preprocessing Failure

C = Detector Failure

D = Reporter Failure

By substituting the values of the gates in upper levels by the equivalent gates from subsequent lower level we obtain the following boolean statement:

$$\text{IDS Failure} = (A3.A6) + (A3.E1) + (A3.E2) + (E3.E4) + (E5.E6) + A5 + (B1.B2) + (B1.B3) + E7 + E8 + E9 + E10 + E11 + E12 + D1 + D2$$

Thus the minimal cut sets are:

$$(A3.A6), (A3.E1), (A3.E2), (E3.E4), (E5.E6), A5, (B1.B2), (B1.B3), E7, E8, E9, E10, E11, E12, D1, D2$$

A simple inspection of this result indicates that these cut sets are single points of failure (i.e., one-event failures or two-event failures at maximum). It is highly probable that attacks pass undetected upon the occurrence of one event or two events at maximum with little effort from attackers. For example, it is sufficient that an attack manifestation appear out of the sensor's scope ($A5$) or out of the analyzer's scope ($E8$) to be undetectable. An example of the two-event failure ($E5.E6$) where fragmented attacks cannot be detected if the IDS does not support fragments reassembly. This result confirms the necessity of reinforcing IDSes by a diversity of capturing and detection techniques. It also confirms the usefulness of using different IDSes to complement each other and overcome limitations found in individual IDS.

In Chapter 6, when we perform an explanatory evaluation of an IDS, we will show how all these pieces of our framework (IDS detection model, failure model and FTA model) can be assembled together to perform robust evaluations.

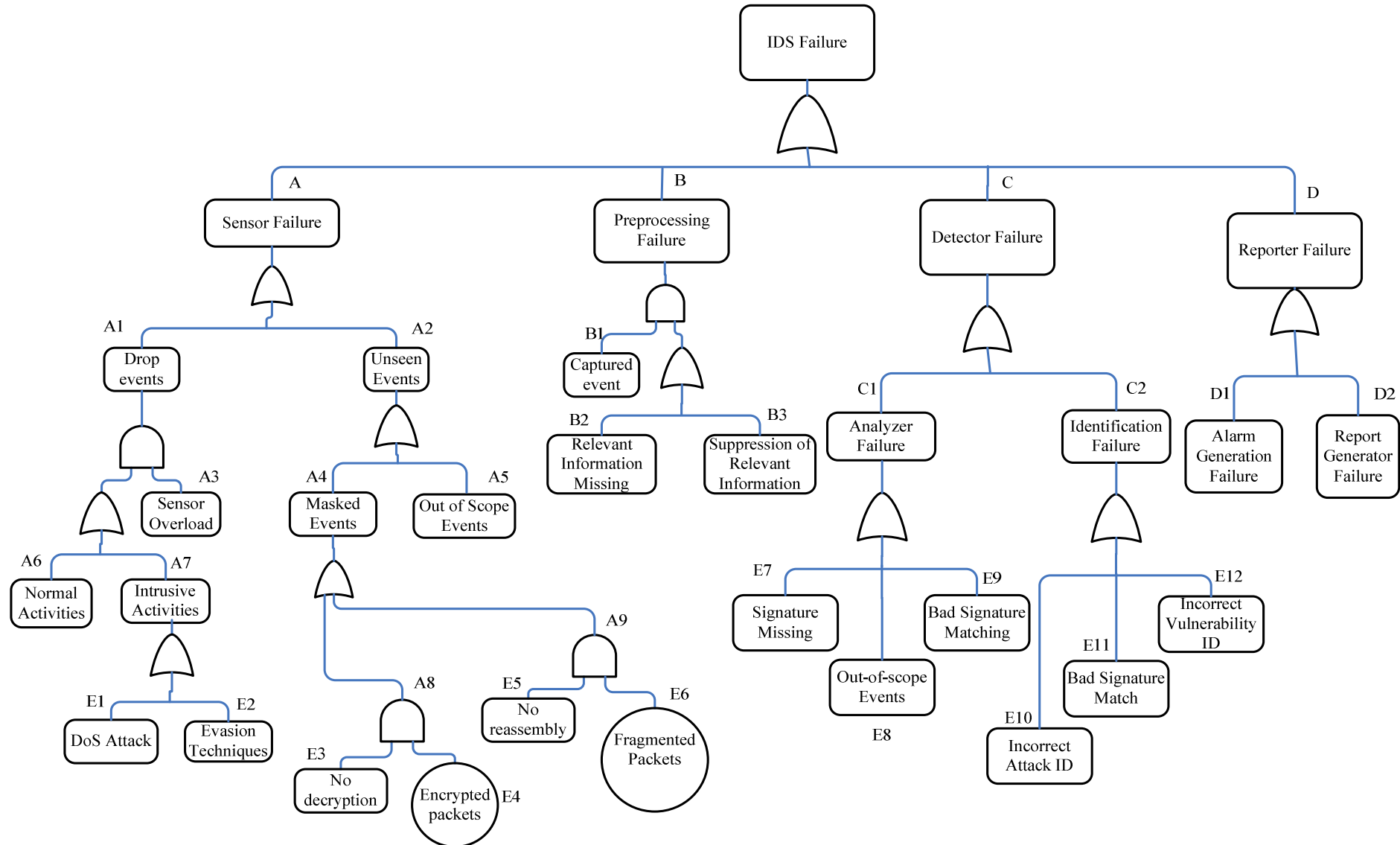


Figure III.5: Generic fault tree analysis for intrusion detection systems.

3.8. Benefits of the Evaluation Framework

Evaluating intrusion detection systems has proved to be an effort intensive and error prone task. As shown in Figure III.6, a traditional evaluation process usually consists of selecting and constructing the evaluation datasets and defining metrics, in addition to designing and carrying the experimentation and analyzing results.

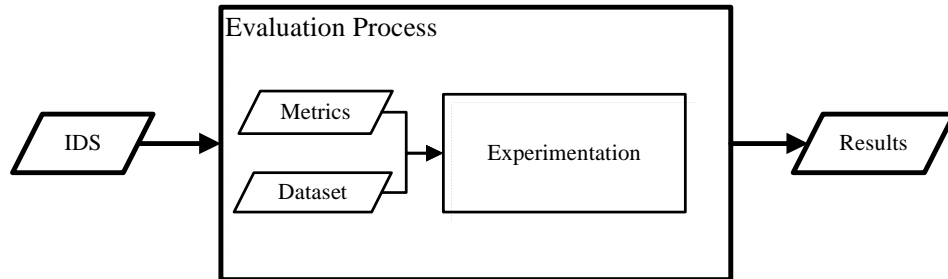


Figure III.6: Process of IDS evaluation.

The provision of a shared dataset has been highly recommended {Mell03}, {McHugh00a}. Using Datasets-Off-The-Shelf (DOTS) and Metrics-Off-The-Shelf (MOTS) where metrics and datasets could be selected – in function of previous steps – from a set of predefined metrics and pre-constructed datasets. DOTS need more effort to provide many datasets with different characteristics. This can be achieved by implementing dataset generators that are able to generate varieties of realistic workloads. DARPA datasets is an example of a widely used DOTS, though it is quite obsolete.

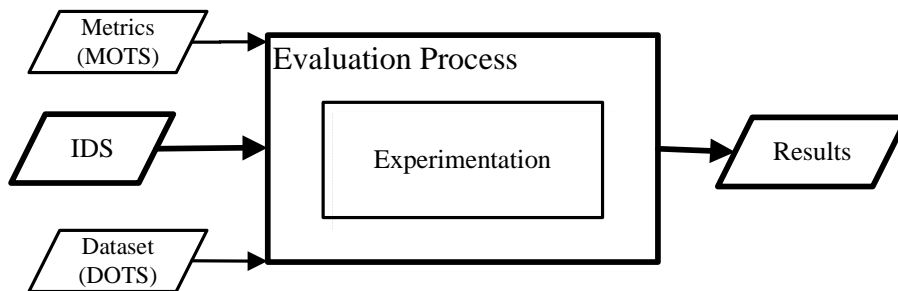


Figure III.7: Enhanced process for IDS evaluation.

Following our approach and given a set of DOTS and MOTS the evaluation process could be carried out with less time and effort. The first stage will be reduced to a simple selection of existing DOTS and MOTS according to the evaluation goal and user needs, as shown by Figure III.7. Therefore, most of the effort will be dedicated to experimentation design to be well fitted and unbiased. We further treat the problem of dataset generation in Chapters 4, and 5.

3.9. Challenging Issues in IDS Evaluation

To illustrate how evaluating IDS is a nontrivial task, we summarize here some of the challenging issues involved in IDS evaluation:

1. **Collecting exploit scripts and attack tools:** One of the main issues in setting up a testing experiment consists in collecting exploits. While it may be relatively easy to find attack scripts on the Internet, these scripts are often proof-of-concept implementations for particular vulnerabilities. They usually require more effort to be tested and to be made reusable. It may also take some time for the user to figure out how the attack script works and how to fix occasional problems (e.g., the correct return address for buffer overflow attacks).
2. **Generating the evaluation dataset** is an important problem related to the generation of both background and malicious datasets. We will discuss this problem and its consequences on the design of testing experiments in more details in the next chapter.

-
3. **Constructing evaluation platforms:** In addition to constructing hardware and network facilities, we should collect and install software systems that correspond to vulnerable services in order to verify the efficiency of attacks. Procuring the corresponding target applications can be very difficult since, most of the time, only the current (and already corrected) version is publicly available. Moreover, most of the vulnerable software is only available under commercial licenses and may require a large budget to be acquired.
 4. **Configuring the evaluated IDS properly:** evaluators should decide whether to evaluate the IDS out-of-the-box without any previous tuning or, alternatively, to modify the default configuration before the evaluation. This approach of testing “default” installations can lead to unfair results. On the other hand, tuning many different IDSes properly can be very difficult and the results can still be biased because some systems can be configured better than others.

IV. Chapter 4: Characterization of IDS Workload

One of the most important issues in evaluating intrusion detection systems is related to generating test datasets. Actually, this process is an art and depends heavily on the experience of evaluators who spend most of their effort and time for this purpose. In the absence of a solid theoretical basis, the produced datasets are likely to be ad-hoc and of low quality. One reason for the failure of previous IDS evaluations is the use of poor quality or badly constructed datasets. Moreover, using such datasets without caution is risky because ignoring their characteristics leads to invalid assumptions. Consequently, people often make incorrect interpretations, improper analysis and erroneous conclusions. For this reason, we need to identify the basic components of IDS workload and identify their main features.

This chapter is divided into three main parts: the first one describes related work, the second part is dedicated to the characterization of attack activities (Sections from 4.3, to 4.7) while the third part characterizes background activities (Section 4.8).

4.1. Introduction

In IDS evaluation, traditional ad-hoc approaches jump directly to the experimentation phase, taking measures and presenting results. Although this is possible for performance evaluation in well-established domains, it is unacceptable for IDS evaluation because we actually lack enough knowledge about IDS workloads and attack processes. Therefore, we believe that any robust evaluation of intrusion detection systems should be preceded by the following steps:

Characterizing real workload → Identifying relevant test-cases → Designing test dataset → Generating test dataset

As we mentioned in Chapter 2, any IDS observes and analyzes some kind of activities searching for suspicious actions. An IDS is very similar to a human security guard or a police officer who watches over a building and its surrounding areas. He is responsible for the security of the building, its doors, parking area, etc. He should observe activities occurring within the area under his supervision. Thousands of events occur every day by hundreds or thousands of people who are entering, exiting, walking, talking, driving their cars in and out, etc. All these actions are supposed to be normal and innocent unless they are followed or preceded by a suspicious action or if it is itself an obviously illegal or criminal act.

In the previous context, although it seems easy for a security guard to distinguish between the two types of actions: the innocent actions (e.g., walk, talk) and the criminal acts (e.g., force open a closed door, break a car window), it is not that easy in practice. Let us consider the following scenario: some person enters the monitored parking area, walks normally, looks like someone searching for his car, and suddenly stops beside a particular car and tries to open the driver door but does not manage to open it. Then he makes a call from his mobile telephone. Finally, he breaks a car window, opens the car and drives the car out.

This scenario can be simply a car theft, while the first part, up to breaking the car window, comprises only innocent actions. Imagine a scenario where the owner of the car has lost his car keys. He calls his office mate in the building to look for the keys on his desk but he does not find them. Suddenly, he sees the keys on the driver seat inside the car and decides to break the car window. For an observer, nothing distinguishes this scenario from a car theft. The security agent would have no suspicion on this person's activities, until he breaks the window, which the guard would probably consider as a criminal offense.

Returning to intrusion detection systems, they observe events that may be a result of actions carried out by legitimate users as well as by attackers. The dividing line between the actions carried by the two parties is often blur rather than clear-cut, and some confusing intersections between the actions of the two parties can be identified. Moreover, an action carried out by a legitimate user does not mean that the action itself is legitimate since a registered user can be an insider attacker. Moreover, an innocent user can unintentionally carry out illegitimate actions (e.g., a user enters a wrong password by mistake). On

the other hand, a significant portion of attacker actions is identical or similar to the actions of normal users (e.g., after getting access to a victim machine, an attacker executes commands to browse and explore the file system). The Venn diagram of Figure IV.1 illustrates the intersection between the two event types.

The ambiguity lies in the intersection between attack and normal use activities. This is where false alarms and false negatives often take place. The problem arises from the fact that some actions (correct or incorrect) can be carried by both parties. Contrarily, purely malicious actions (e.g., executing a buffer overflow exploit) should have no ambiguity and should be always identified as malicious by any good IDS.

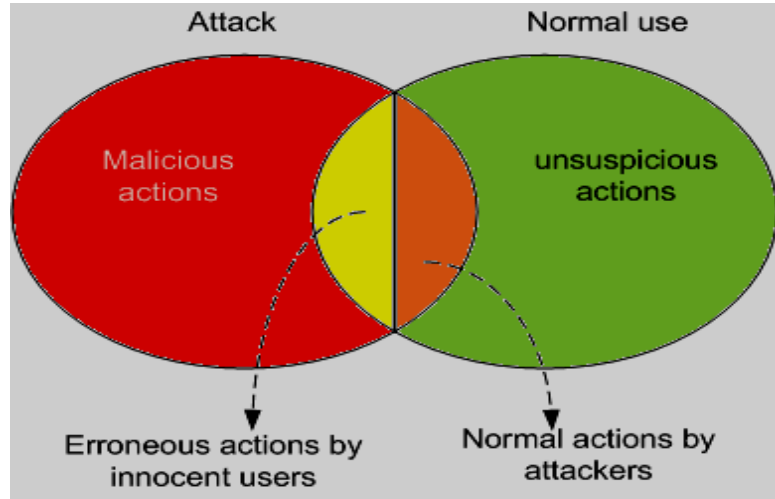


Figure IV.1: Intersection between attack and normal use activities.

For simplicity, we divide IDS workloads into two components: the *background component* (i.e., events that occur due to normal use or normal operation of computers and networks) and the *attack component* (events due to attacks). Correspondingly, an evaluation dataset consists of events that are generated by background activities and attack or intrusive activities.

In order to characterize both components, we identify, in this chapter, the main features of each component while giving more focus to the attack component. The remainder of the chapter is organized as follows: First, we present an overview of related work. Starting from Section 4.3, we characterize attack activities found in the real world. Real-world attacks are analyzed and classified with respect to elementary attack features (Section 4.4) as well as scenario-level (Section 4.5). Examples on attack characterization are detailed in Sections 4.6 and 4.7. Then, we characterize background activities in Section 4.8 and draw out conclusions in Section 4.9.

4.2. Related Work

Before presenting our work on the characterization of attack activities, it seems necessary to discuss related work on attack models, malicious dataset generation techniques as well as attack description languages. Discussing the research work in these areas will be helpful to understand and situate our work. In the following we present briefly the most relevant ones.

4.2.1. Attack Models

Several models of attacks have been proposed in the last few years [LiChen03], [Cheung03], [Garg06], [Tidwell01], [Schneier99], [Sheyner02], [Templeton00], [McDermott00], [Dahl06], [Dacier94], [Kaaniche06]. Hereafter, we briefly present some of the models that seem to be relevant for our objective.

A) Attack Trees

Attack trees are intended to model security threats [Schneier99]. Actually, an enterprise security breach (the final goal of an attack) is first identified as the root of the tree. Then, the steps by which this

breach could be accomplished by an attacker are iteratively added as branches and leaves of the tree. In this respect, each path through an attack tree represents a unique attack scenario. Attack trees are similar to fault trees where they are frequently used in system dependability analysis (see 0).

An enhanced approach for the attack trees was introduced in {Tidwell01}, where a dual specification language to express exploits and network characteristics was proposed. The model developed in {Moore01} is also based on the attack tree model. Its main purpose is to provide means for documenting intrusions. Therefore, it tends to be a pure descriptive approach.

B) Attack graphs

Attack graphs are mainly used to analyze and assess vulnerabilities {Sheyner02}. A typical vulnerability analysis of a network proceeds as follows: first, gather vulnerabilities of individual hosts, e.g., by using tools such as *nessus* {Nessus08}; then, obtain information about the network topology and host interconnections; finally construct attack graphs. Each path in an attack graph represents a sequence of atomic attacks or exploits that may lead to a final insecure state of the network. Upon the constructed attack graph, a set of analysis can be carried out, such as: shortest path analysis, reliability analysis, and risk assessment analysis. Attack graph analysis is frequently used in system dependability analysis.

C) Requires/Provides

The "requires/provides" model was derived in order to describe complex scenarios and to generalize them to unknown attacks rather than considering single exploit vulnerabilities {Templeton00}. Instead of describing attacks by the specific sequence of actions that an attacker uses to reach a specific goal, this approach describes attacks by using abstract components. Each concept is defined locally without determining which other concept it will work with. Only, the capabilities required and the capabilities provided by a concept are defined. Capabilities are the information required or the situation that must exist for a particular aspect of an attack. Consequently a linkage can be done through a require/provide model. Therefore, a complete attack can be composed of these components that serve as building blocks for single-point attacks. The advantage of this model is that it needs no prior knowledge of a particular scenario and thus numerous unknown attacks can be described implicitly.

D) Attack Petri Nets

Attack Petri nets or simply attack nets were introduced firstly in {McDermott00} and mainly used for modeling the process of penetration testing. It uses disjunctive Petri nets to describe attack processes. Attack Petri nets can be directly derived from attack trees by mapping the nodes and arcs of attack tree into places and transitions of a Petri net. Places represent interesting states of the system or other security relevant entities. This state could be the acquisition of knowledge by the attacker or the gain of control. Transitions represent input events, data or commands that cause a change of system or entity state. The move of tokens between places indicates the attack progress. Using Petri nets for modeling attack processes is beneficial as it enables the modeling of concurrency with tokens as well as commands and inputs, which are modeled by the transitions.

An extended version of general attack nets was presented in {Dahl06}. It is mainly based on Interval Timed Colored Petri Nets (ITCPN). This allows the modeling of multi-agent interactions and time-dependent attacks. It provides also a sufficient level of details for the simulation and the execution of attacks since the resulting attack net could be mapped directly onto an execution element of an attack.

E) Privilege graphs

The privilege graph approach is amongst the first attempts to model security weaknesses {Dacier94}, {Ortalo99}. It models vulnerabilities existing in computer systems that, when exploited, raise the current level of privileges and through which attackers can compromise the system. A node in the privilege graph represents a set of privileges owned by a user or a set of users. The arc between nodes X and Y represents a possibility for a user owning the privilege set X to gain the privileges corresponding to node Y; thus, each arc represents a specific attack opportunity (an elementary attack action). A weight can be assigned to each elementary action, corresponding to the effort an attacker must spend to exploit the vulnerability. A security breach corresponding to an attack scenario can occur when a path exists between one node representing a possible attacker's privileges to another node representing an attack target. Privilege

graphs were invented principally as a basis for quantitative security assessment. Once the privilege graph constructed and according to a predefined security policy (which helps to define the attacker and the target nodes), different metrics can be computed, in particular the METF (Mean Effort To Failure), with various attacker behaviors (ranging from memory-less to total memory). As indicated by its name, this model concentrates on the access gain and privilege escalation attacks.

F) Statistical attack models

The purpose of statistical models is quite different from those presented above. They aim to study distributions of attacks over time. For example, to determine which systems/applications or IP address zones are the most targeted, the frequency of attack occurrences, etc. The information for such type of models is usually extracted from data captured and recorded by honeypots or honey networks such as those deployed on the *Leurrecom* data collection platform {Leurrecom08}.

An example of statistical models can be found in {Kaaniche06} where two statistical models of attack behavior were formulated from data collected by many low-interaction honeypot platforms during 320 days. The first describes the evolution of attack numbers over time by means of a linear regression model. The second describes the distribution of time between attacks, using a mixture of Pareto and exponential distributions, which proved to produce the best fit.

G) Discussion

Unfortunately, the majority of attack-related models were developed to enable security analysts and red teams understanding and analyzing their networks, and they are not necessarily adequate for IDS evaluation. Other models are focusing only on simple attack patterns such as signatures of elementary attack instances, which might not be sufficient to evaluate all the capabilities of IDSes. For example, to evaluate state-full IDSes or to test the correlation function of multi-sensor IDSes, it is necessary to run sequences of events that correspond to complex scenarios rather than isolated elementary attacks.

Furthermore, almost all the models, cited above have poor scalability and suffer from combinatory explosion when the size of the analyzed network exceeds tens of hosts. Another hindering limitation is that the creation of these models is strongly dependent on the specifics of the analyzed network. Therefore, they are not suitable for analyzing other networks. Even worse, any slight modification of the analyzed network may induce significant modifications in the model or may even require recreating it.

A few models, such as those presented in {Templeton00}, {Futoransky03}, {Coreimpact08} and {Jonsson97}, adopt the attacker viewpoint and can serve to mimic the attack process. The Rapid Penetration Test (RPT) model described in {Futoransky03} and {Coreimpact08} focuses only on attacks for penetration testing. It splits the penetration process into six steps: (1) information gathering, (2) attack and penetration, (3) local information gathering, (4) privilege escalation, (5) clean up and (6) report generation. The model of {Jonsson97} divides the intrusion process, according to attackers' behavior, into three phases: learning phase, standard attack phase and innovative phase. Although such models describe well some important features of the attack process, they fail to express or capture other important ones. For example, the RPT model ignores DoS attacks that represent a significant proportion of security incidents. Additionally, this method pays less attention to post-access activities that attackers execute after they gain an appropriate access. This can be advantageous for penetration testing where we intend to test how easy it can be to break in a targeted system. However, it may be a drawback for IDS evaluation because it ignores a significant part of attack attempts.

For all these reasons and because a non-biased evaluation requires generating various attack scenarios that correspond to different attacker profiles and covers a wide range of the attack space, we ought to create our own models. Therefore, we propose in Section 4.5.3 a model for the attack process, which will be completed in Chapter 5 by a model characterizing attacker skills and a statistical parameterization based on honeypot data. In the following, we expose the related work on attack generation techniques.

4.2.2. Attack Generation Techniques

Many tools have been used for IDS testing {Athanasiaades03}. A non-exhaustive list includes: network scanning tools such as *nmap* {Nmap08}, vulnerability scanning tools (e.g., *nessus* {Nessus08}), tools for testing web sites such as *Nikto* {Nikto08} and *httperf* {Httpperf08}, network traffic generators such as *D-ITG* {Itg08}, etc. Quite often, such tools have been developed to test a very specific security function, or

even for purposes other than security. Besides that, we can cite *snot* {Snot07}, a tool for testing IDS signatures, which uses *snort* rules to generate corresponding attack packets and does not generate attack scenarios.

Expect (a *TCL*-based tool) has been used to generate both attack and user sessions for IDS testing {Puketza97}. It provides some level of interactivity with the victim machines during attack sessions, where a priori written scripts are used to automate command execution. An interactive execution of commands according to the reactions of the normal user or the attacker is advantageous but writing automation scripts manually is tedious, and limited to the expected responses of victim machines only.

Recently, penetration testing frameworks such as *metasploit* {Metasploit08}, *CoreImpact* {Coreimpact08} and *CANVAS* {Canvas08} have been increasingly used by security practitioners and developers for security assessment and security product testing. The first is an open source package whereas the two later are commercial products. We have little information on the commercial tools about the included exploits, attacking procedures, etc. Therefore, we restrict our discussion here to the publicly available tools. Generally, penetration-testing tools are focusing on attacks that provide access to victim machines, and have limited or no interest in post-access actions. In other words, they provide elementary access-gain attacks rather than attack scenarios. In fact, this category of tools provides a good starting point for testing IDS with various levels of automation. However, complementary tools are essential to provide a better coverage of both the attack space and attacking procedures.

Finally, the last kind of tools or datasets relies on the replay of pre-registered attack traces that are collected from the wild or synthesized from previous test sessions. The famous datasets from DARPA {Lippmann00a}, {Lippmann00b} are clear examples of this kind. However, they are now considered obsolete and have been heavily criticized {McHugh00a}, {Mell03}. A more recent dataset is the one created by the *Canadian Communications Research Center (CRC)* {Massicotte06} that contains attack sessions against virtual victims (i.e., VMware machines). Attack sessions in the *CRC* dataset are mostly generated by using *metasploit*; therefore it inherits the limitations related to the *metasploit* dataset.

In the commercial tool category, we can cite *Traffic-IQ* {Trafficiq08}. It consists of a front-end by which users control the evaluation process: selecting attacks, launching the corresponding attack traces, viewing results, etc. Despite the advantages provided by *Traffic-IQ*, it is not easy to change the attack traces/scenarios or to produce more scenario variations and integrate them in the attack traces. Moreover, the growth in the number of attack traces makes the approach of replaying ready-made attack traces quite expensive in storage size.

For completeness, we should also mention the attack transformation tools used to test the resistance of IDSes against the evasion techniques used by attackers to escape detection, such as *Fragroute* {Fragroute08}, *Mucus* {Mutz03} and *THOR* {Marty02}. Based on our experience, we summarize in Table IV.1 our observations regarding the main features of methods and tools commonly used by IDS evaluators.

From these observations, it is clear that the last decade has witnessed an explosion in the number and diversity of tools available for security evaluations. As the list of alternatives becomes longer, it might be confusing for evaluators as each tool has its own strengths and weaknesses. The fact that evaluators (or the audiences targeted by the evaluation) might not be aware of the characteristics of such tools and particularly their coverage of attack types and how attack processes take place makes the results seriously biased. Moreover, since some of these tools were developed for very specific purposes other than IDS evaluation, it would be inappropriate and probably dangerous to establish IDS evaluations solely on one of these tools, a penetration-testing tool, for instance.

Table IV.1: A comparison between different evaluation tools/techniques.

	Flexibility	Effort	Extensibility	Reproducibility	Coverage of attack space
Script TCL/TK	High	High	Yes	Yes	Variable
Network scanning tools	Low	Low	No	Yes	Limited
Vulnerability scanning tools	Low	Low	No	Yes	Limited
Penetration Testing Tools	High	Medium	Yes	Yes	Limited
Replay of attack traces	Low	Medium	No	Yes	Limited

Another approach is based on attack description languages, which sometimes are developed either as a part of tools, e.g., *NASL* (Nessus Attack Scripting Language) {NASL08}, or as a standalone language such as *ADeLe* {Michel01}. In the next subsection, we discuss some of these attack description languages.

4.2.3. Attack description Languages

Vigna *et.al.* have classified attack languages in six categories {Vigna00}: exploit, event, detection, correlation, reporting and response. The aim of exploit languages such as *NASL* (Nessus Attack Scripting Language) {NASL08} is to describe the steps in which an attack can be performed. The purpose of event languages is to describe the format of events captured during attack sessions. Accordingly, *tcpdump* {Tcpdump08} and audit trails {Bishop95} were considered as event languages. Detection languages such as *STATL* {Eckmann02} describe the manifestation or the signature of attack events. Correlation languages represent the probable correlation of alerts produced by different IDSes. IDMEF {Debar07} is an example of reporting language that aims to normalize IDS alert formats to facilitate the exchange and the analysis of alert messages between different IDS or correlation engines. Finally, the purpose of response languages is to describe the reaction that should be taken by security countermeasures to mitigate attacks. Languages such as *ADeLe* {Michel01} and *LAMBDA* {Cuppens00} are multipurpose languages and can be used, for example, as exploit, detection or correlation language.

Generally, attack description languages require defining attack-scenarios a priori, which implies the definition of attackers' actions as well as pre and post conditions. Exploit languages share the same objective with the work presented in this chapter (i.e., to generate attack scenarios). However, it differs from our work in that they provide language statements that can be used, by an expert, in a generic way to "program" or to "script" attacks. They can be viewed exactly as programming languages that offer developers a rich set of statements and libraries.

We do not aim by our work to create another attack description language, but rather to produce attack-scenario "programs" and to automate the process of creating such programs. From this perspective, attack languages may be added to our attack-scenarios development kit.

4.3. Characterizing Attack Activities

To clarify the meaning of some basic concepts, we have adopted, and sometimes adapted, the definitions proposed by the MAFTIA project {Maftia03}.

- *Exploit (noun)*: a script, a program, a mechanism or other technique by which some vulnerability is used to realize an attack or a part of an attack.
- *Activity*: an action or a set of actions that generate *events* in the system.
- *Malicious activity*: an activity carried out by an attacker that aims to violate the security policy.
- *Normal activity*: an activity carried out within the context of normal operations without the intent to compromise the security policy.
- *Attack scenario*: a set of organized activities, including malicious activities and apparently normal activities, which are executed in sequence or in parallel to achieve the attacker's goal.

Therefore, we mean by attack activities all activities that attackers can exercise on a host or network victim. This includes activities that are malicious by nature (e.g., buffer overflow, DoS attacks) as well as normal activities that are not necessarily suspicious (e.g., using system utilities to browse the victim host). We prefer to talk about attack activities rather than attacks for three reasons. First, there is no consensus between security experts on the definition of attacks. Second, attack tools have become so complicated and highly customizable that a single tool can provide a wide variety of actions in many ways. For example, a tool such as *nmap* comprises several techniques to scan networks, and it can be parameterized to select a particular technique by modifying command options. Third, we aim to characterize not only simple attacks but also attack scenarios that cover various attack patterns. Simple attacks can have a one to one mapping to elementary attack actions, while attack scenarios usually consist of series of both malicious and non-suspicious actions.

In fact, we characterize attacks at two different levels: a low level where an attack is defined as a single action and a high level that corresponds to attack scenarios. At the lower level, we are more interested in the occurrence and the manifestations of attacks, while at the scenario level, we concentrate on the functional characteristics.

In the forthcoming sections, we characterize firstly simple attack actions (i.e., activities that cannot be further decomposed into elementary attack actions). For this purpose, we have created a classification scheme for attack actions according to their manifestations observable by IDSes. Then, the potential characteristics of attack scenarios are described by an attack process model, which focuses on the functional features of attacks (the attack classification and the attack process models are presented in the next two sections respectively).

4.4. Low-level Attack Characterization (Elementary Attack Action Classification)

Given a particular IDS, how can we determine if it behaves as expected? Certainly, we can test it with various inputs, but we can never test it with every possible input or every possible interleaving (combination) of all possible inputs. This problem has already been addressed in software testing, which uses the concept of equivalence classes {Myers79} to reduce the number of inputs for which the software has to be tested. The idea is that input test cases belonging to the same class are assumed to stimulate (activate) the same parts of the software in the same conditions, and thus should produce equivalent results. We try here to extend this principle to IDS testing, by defining a classification for elementary attack actions.

Before discussing further the classification that we propose, let us clarify what it means. *Classification* is defined as a “systematic arrangement, in groups or categories, according to established criteria” {Webster}, while *taxonomy* is defined as “the study of the general principles of scientific classification” {Webster}. It is worth noting that both words are often used interchangeably as synonyms in the security literature, and we follow the same convention in this chapter.

In the past few years, there have been several attempts to classify vulnerabilities in a common way. This has enabled the construction of vulnerability databases such as MITRE’s CVE (Common Vulnerability and Exposure) {Cve08} and OSVDB (Open Source Vulnerability Database) {Osvdb08}. There have been also many attempts to classify attacks, e.g. {Alessandri04}, {Bishop99}, {Lindqvist97}, {Neumann89}, {Kumar95}, {Weber98}, {Kendall99}, {Howard98}, {Hansmann03}, {Lough01} and {Killourhy04}. However, no comprehensive and widely accepted classification has been established yet, probably because those who defined these classifications did not share the same objectives. Before elaborating our own classification scheme, we analyzed several previous attack classifications to check whether one can match our objective, i.e., IDS evaluation and testing. In the next section, we present a brief description of the classifications we analyzed. A detailed description of all classifications is beyond the scope of this dissertation.

4.4.1. Analysis of Existing Attack Classifications

Let us begin with Bishop’s vulnerability taxonomy {Bishop99}. Although this taxonomy is intended for classifying vulnerabilities rather than attacks, it might be useful to present it briefly because it proposes interesting attributes (or axes): the *nature* of the flaw; the *time of introduction* of the vulnerability; the *exploitation domain* (i.e., the consequences of exploitation); the *effect domain* (what is affected); the *minimum number of components* essential to exploit the vulnerability; and the *source of identification* of the vulnerability.

Another interesting work is the two-dimension taxonomy that was introduced by Lindqvist and Jonsson in {Lindqvist97}. By adding the result dimension (exposure, denial of service, erroneous output), it extends the Neumann and Parker’s taxonomy {Neumann89} that had only the technique dimension (e.g., bypassing intended controls, active misuse of resources). Lindqvist and Jonsson defined their taxonomy from attack experiments realized by internal users (undergraduate students of a computer science class). This is the weak point of this classification, since it ignores a significant part of more sophisticated attacks that would be carried out by more experienced attackers.

Kumar had classified attacks according to four attributes related to signature patterns: *existence*, *sequence*, *interval* and *duration* {Kumar95}.

Weber’s taxonomy is based on three dimensions: the *required level of privilege* to conduct the attack, the *means* by which the attack proceeds (e.g., exploiting a software bug) and the *intended effect* (e.g., a denial of service) {Weber98}. DARPA evaluations were based on Weber’s taxonomy, but distinguished

between effect (e.g., probe or scan and denial of service) and transition to upper privilege levels (e.g., remote to local, (R2L), user to root (U2R)) {Kendall99}.

Howard's taxonomy {Howard98} is more interested in the attack process rather than in the attack itself. It proposed to divide the attack process into stages:

1. *attacker* (who is she/he: a simple hacker or a terrorist group),
2. *tool* (what does the attacker use: a kiddy script or a specialized tool),
3. *vulnerability* (in implementation, configuration or design),
4. *access* (what kind of unauthorized access is obtained: to files, objects or processes),
5. *results* of attack (exposure or corruption of data), and
6. *attack objectives* (e.g., destroy data, collect information).

Simon Hansmann's taxonomy {Hansmann03} has four dimensions: the *attack vector* (i.e., the means by which the attack reaches its target: virus, worm, DoS, etc.), the *attack target* (e.g., OS, application, network protocol), the *exploited vulnerability* and the *effects of the attack*.

After reviewing many attack and vulnerability taxonomies, Lough has established the VERDICT taxonomy {Lough01} that focuses on causes and effects of improper conditions that enable attack occurrences. It has four main dimensions: *Validation* (e.g., absence of input validation), *Exposure* (e.g., information provided by *finger* service), *Randomness* (e.g., poor random algorithms for cryptographic key generation) and *De-allocation* (e.g., ineffective memory management). An attack can be described or classified by one or more of these four improper conditions.

The so called defense-centric taxonomy was introduced in {Killourhy04} to serve network administrators in defending their own systems. It classifies attacks according to attack manifestations in system calls as seen by anomaly host-based intrusion detection systems. The four features or dimensions of interest are:

- 1) *Foreign symbol*: a system call that appears when attacks occur and never appears in normal operation,
- 2) *Minimal-formal-sequence*: a sequence of manifestations that appears when attacks occur and never appears in the normal operation (although all its subsequences appear in the normal operation),
- 3) *Dormant sequence*: a sequence of manifestations that partially matches a subsequence in the normal operation, and
- 4) *Non-anomalous sequence*: a sequence of manifestations that fully matches sequences in the normal operation.

Unlike the classifications presented above that takes the attacker/administrator viewpoint, the taxonomy presented in {Alessandri04} was created for the purpose of analyzing IDSes. It classifies activities that could be relevant to IDS instead of classifying attacks directly. An analytic evaluation of IDSes was later carried out, based on this taxonomy, to compare IDS detection capabilities with respect to attack classes. The underlying model of the observable manifestations distinguishes dynamic characteristics from static characteristics of activities. Static characteristics are further split to separate the characteristics related to interface objects and those related to affected objects. Similarly, dynamic characteristics are developed into three sub-characteristics: communication features, method invocation characteristics and other additional attributes. Thus, an attack can be described by five parameters: *interface object*, *affected object*, *communication*, *invocation method* and *other minor attributes*. In total, it distinguishes 24 interface objects, 10 affected objects, 2 communication characteristics, 5 method invocations and 4 minor attributes.

In fact, even if this classification has been developed to evaluate IDSes by analysis, it is still inappropriate for our purpose, i.e., the evaluation of IDSes by test. For example, it is focusing on the manifestations of attack activities that can be observed by the IDS while ignoring other attributes that may be important in practical operation, such as the consequences, the privileges acquired, and the source of attacks. Moreover, this classification contains very fine-grained dimensions that massively increase the number of test-case classes while the attained level of detail has minor significance for the tested IDS. For instance, the dimension *interface object* – that contains 24 types – considers five distinct types related to the application layer:

- Application layer-connectionless;
- Application layer-single connection-single transaction;
- Application layer-single connection-multiple transaction;

- Application layer-multiple connection-single transaction;
- Application layer-multiple connection-multiple transaction.

In practice, this aspect has very little interest to classify elementary attack actions, because this would lead to some classes to be under populated, while others would gather most of the real cases.

Globally, existing classifications are inadequate for our purpose, due to several reasons. First, they often take either the attacker or the IDS (the defense) viewpoint but neglect the evaluator's viewpoint. Second, they have attributes beyond the scope of the IDS while often ignoring or masking significant attack features. Third, they sometimes have ambiguous, inconsistently defined attributes. Fourth, they usually have a huge number of classes. Fifth, there is no accompanying scheme for test case selection. Despite the limitations of these classifications, it is worth analyzing their attributes to select those that can be relevant for our objective. We noticed that the analyzed classifications take different viewpoints and use inconsistent attribute names. However, they are generally based on attributes of attack and/or vulnerability such as the following ones:

1. *Type of attack*: virus, worm, Trojan horse, denial of service, etc.;
2. *Detection technique*: pattern matching, statistical approach, etc.;
3. *Signature*: observed attack pattern, attack sequence pattern;
4. *Tool*: physical, user command, script, tool kit, etc.;
5. *Target* : OS, network protocol, application, service, process;
6. *Results*: data corruption, exposure of information, DoS, etc.;
7. *Gained Access*: root access, user access;
8. *Preconditions*: e.g., presence of particular software versions;
9. *Vulnerability*: buffer overflow, validation error, weak password, inappropriate configuration,
10. *Objective*: terrorism, political/financial gain, self proving;
11. *Attacker location*: external, internal;
12. *The compromised security property*: confidentiality, integrity, availability.

These attributes are the basis of the new classification of attacks that we propose in the following.

4.4.2. Classification Requirements

Developing a classification of attacks that takes the evaluator's viewpoint yields many benefits. First, it will reduce drastically the number of necessary test cases. Second, it provides a more comprehensive evaluation with better coverage of the attack space than traditional IDS evaluations where the evaluators use a few attack scripts available in their hands or on security mailing lists, but such available attack scripts do not necessarily reflect real attack distributions or even do not cover some critical attack types. Third, expressing the results of evaluations in terms of attack types will provide a more precise image of the results. For example, a misunderstanding could arise from the generalization of conclusions when expressing results for all attacks included in the test cases whereas the tested IDS is weak in detecting certain types of attacks and strong in detecting others.

In order to serve in IDS evaluations, a good classification should satisfy the following requirements {Alessandri04}, {Lindqvist97}:

1. *Completeness/exhaustiveness*: it means that a categorization scheme should take into account all possible attacks (e.g., known and unknown).
2. *Clear and unambiguous criteria*: if each dimension has a number of *distinct classes*, any attack belongs to one and only one class in each dimension.
3. *Mutually exclusiveness*: to ensure that an attack is placed at most in one category.
4. *Repeatability*: the classification procedure ensures that an attack is always placed in the same category.
5. *Compliance with existing standards*: for example, vulnerability databases have become de-facto standards in security and any comprehensive attack classification should be linked to at least one of them.

In addition to these general requirements, we can identify two more requirements that are important from the evaluation perspective:

1. The combinations of all the classes of all the dimensions could lead to an excessive number of distinct classes. Consequently, there should be a method in classifying attacks to help in selecting only the categories of interest.

2. The classification should also take into account special aspects of attack generation, which are essential for the evaluation process (e.g., information necessary for the configuration of the test platform).

4.4.3. New Classification Proposal

We have analyzed carefully the attributes of previous classifications to determine which attributes and dimensions are significant from the IDS evaluation perspective. We will discard issues that are invisible for IDSes as well as those that are meaningless for the evaluation. For example, the *attacker's objective* (or *intention*) will not be considered as a relevant dimension within this classification since it is both hard and useless to identify it.

Similarly, both the *type* and the *detection technique* dimensions do not provide precise, clear-cut categories. The *type* dimension, as it was defined, includes attacks such as virus, worm, Trojan, etc. Letting aside the propagation features, these types of attacks have too many diverse manifestations and functionalities to be categorized at this level. Furthermore, they represent attack scenarios rather than simple attack actions (we will treat attack scenarios in the next section). On the other side, the *detection technique* is not an intrinsic feature of attacks, and a given attack can be detected by several ways and techniques.

While the *result* and *security property* dimensions give an indication about the expected damage, it is out of the IDS scope. The three basic security properties (i.e., Confidentiality, Integrity and Availability) are neither visible nor measurable by the IDS.

The *preconditions* and the *vulnerability* dimensions are closely related and can be reduced to one dimension. The *signature* dimension tends to be very specific to attack instances rather than characterizing attack classes. Thus, it should be discarded.

Thanks to the analysis that we made and according to the requirements stated above, we propose a new classification {Gadelrab07}. As illustrated in Figure IV.2 our classification has five dimensions, which reflect attack manifestations as well as evaluation-relevant aspects:

1. *Firing source* that indicates the place from which the attack is launched. It has two distinct classes: remote and local. This will determine the location from which an attack test case has to be launched. It can help in selecting the location and the type of IDS (e.g., on which network segment, host-based or network-based). It is also important to assess the capacity of the evaluated IDS to detect remote as well as local attacks.
2. *Privilege escalation*: indicates whether the attack results in raising the privilege level. It has five distinct classes that correspond to the gained privilege level: *root*, *user*, *system*, *variable* and *none*. The last one covers attacks that do not need or do not result in any access to system resources. (e.g., remote DoS attack). The *variable* class denotes attacks that gain various privilege levels depending on the execution privileges of the exploited application. For example, exploiting an *apache* server may result in user or root privilege escalation, depending on whether it is run as a user service or a root service respectively.
3. *Vulnerability*: expresses the relationship between attacks and vulnerability databases. It can point to a specific vulnerability exploited by the attack. Up to now, we keep it as abstract as possible: the classification indicates whether the vulnerability is due to *configuration* or *design/implementation* flaws.
4. *Carrier*: describes the means by which the attack reaches its victim: either via network traffic or through an action performed locally on the machine. An attack can be carried via the network and needs no direct intervention on the local machine. Otherwise, an attack activity can be invoked and executed entirely on a local machine without appearing on the network interface. In other cases, attacks have observable symptoms on both the network and the local machine because they trigger events on both.
5. The last dimension is the *targeted object*: attackers may target a service, the operating system, the network stack, an application or a file system object.

Our classification does not focus solely on the observable characteristics of attacks like did the IDS-centric {Alessandri04} and the defense-centric {Killourhy04} taxonomies. Instead, it considers also operational issues that are important for administrators. For example, the severity of attacks is reflected implicitly by privilege escalation dimension. In addition, the source of danger (i.e., the firing source and the vulnerability types) could suggest how the risk could be alleviated by which counter measure (e.g.,

modify firewall rules to block a remote source or apply a missing patch). Moreover, it does not ignore the evaluators' needs, providing essential information for generating attacks and analyzing test data. For example, the firing source dimension gives an idea about the location from which an attack should be generated, and the vulnerability dimension tells whether a particular configuration should be set/unset.

We will give some classification examples of elementary attacks in Section 4.6 that explain how our classification can help in characterizing attack actions, but before we will explain in the next subsection how it can help in selecting attack test cases.

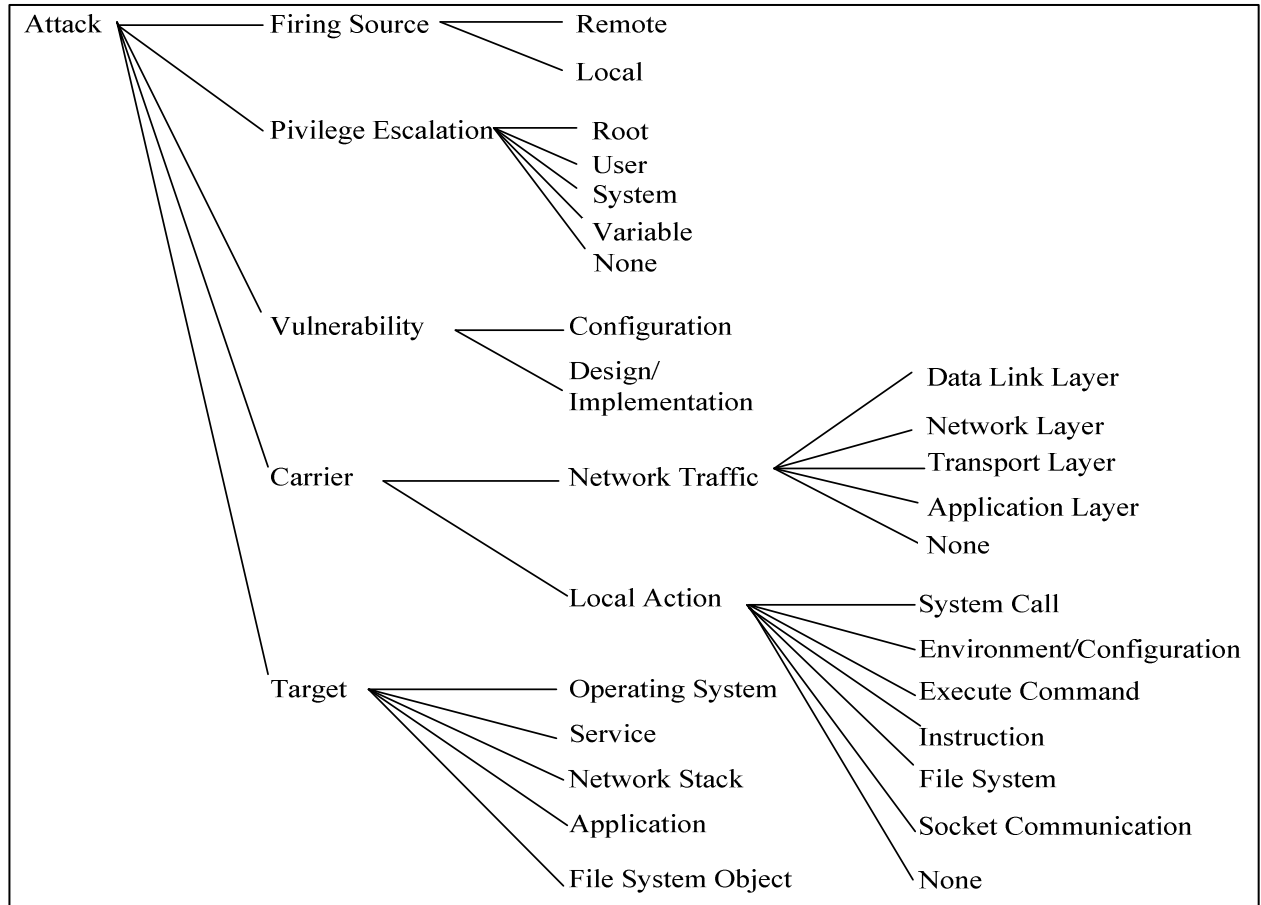


Figure IV.2: A classification scheme of elementary attack.

4.4.4. Classification-based Selection of Attack Test-cases

As always, testing can reveal errors, but it cannot demonstrate or guarantee the absence of errors. The equivalence-class approach can be helpful to perform better testing, but in our case, it would not be judicious to consider each class of each dimension as an equivalence class (see the following subsection). First, there is no certainty that two attacks with the same classification would necessarily lead to the same detection properties. Moreover, taking samples in each class of each dimension usually produces a so large number of attack cases that it is practically impossible to include them all in an evaluation dataset. A selection of test cases based on the classification can reduce the number of test cases while improving the predictive power of the test and giving more confidence in its results.

In this section, we propose evaluators to select relevant attack test cases by using the Classification Tree Method (*CTM*), which was developed by Grotchamann and Grimm {Grochtmann95}. It was applied in testing systems in various domains and we apply it to the security-testing domain. But first, let us describe the method itself.

By means of the *CTM*, the input domain of a test object is regarded under various aspects or dimensions that are assessed according to their relevance for the test. For each aspect, disjoint and complete classifications are formed. The stepwise partition of the input domain by means of classifications is represented graphically in the form of a tree.

To construct test cases, a grid is drawn below the tree. The columns of the grid result from vertical lines that correspond to the leaves of the classification tree. A tester can construct a test case by selecting a single leaf class of each higher-level branch of the classification. Each row of the grid indicates a distinct category of test cases. Because not all test cases are legal or valid, the tester should eliminate the invalid ones. This can be done by the definition of constraints or generation rules in the Classification Tree Editor (*CTE*) tool.

Example: Given the attack classification tree, the *CTE* {Systematic08} can produce all the possible combinations of the distinct subclasses in all the dimensions. The number of valid combinations produced from our classification is 3400 test cases compared to 9600 in {Alessandri04}. The test cases can still be reduced, grouped and reordered to a smaller number of relevant ones, by applying constraints or generation rules in *CTE*. For example, the following rule: [*Remote* * (*root* + *system*) * *configuration vul* * *Network traffic* * (*FS object* + *OS*)] will result in 20 test case categories, which represent remote attacks that provide root or system access by exploiting configuration vulnerabilities and that could be observed in network traffic, targeting the system file or the operating system (Figure IV.3).

As we will explain in the next chapter, the classification scheme, combined with the CTM method, is integrated in our evaluation framework to provide a versatile selection of attack test cases. However, there are still some limitations in selecting attack test cases based on this classification. We will discuss this issue in the next subsection.

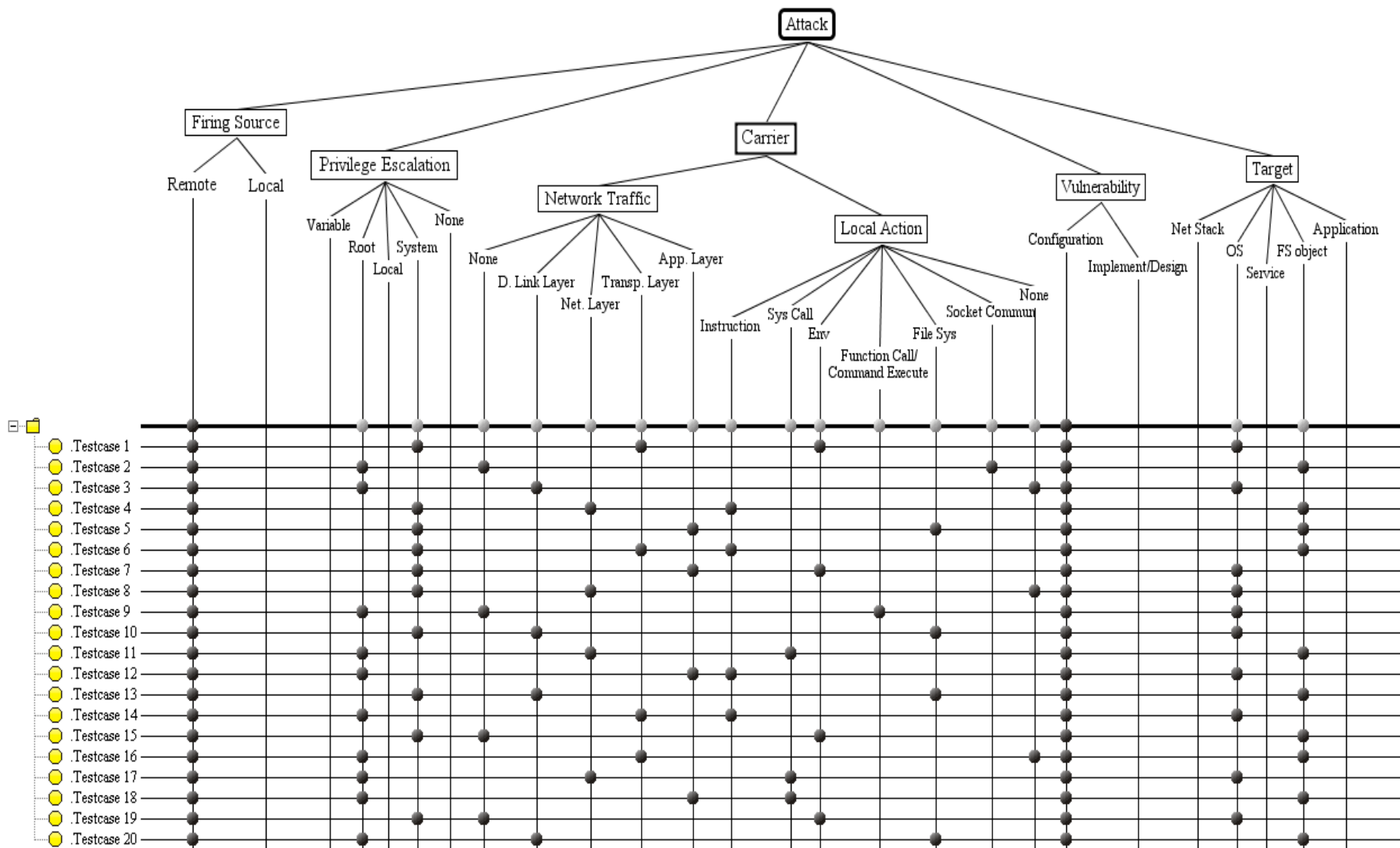


Figure IV.3: Test cases produced by the CTE tool.

4.4.5. Discussion and Limitations

Based on the classification, we argue that we can better select attack test cases than previous IDS evaluations do. It may come to mind that our classification can serve merely as a basis for equivalence-class testing. This is also known as “Equivalence partitioning” {Myers79}, which is a software testing related technique with the goal: (1) to reduce the number of test cases to a necessary minimum; and (2) to select the right test cases to cover all possible scenarios.

This technique assumes that all inputs within a partition or a class are equivalent. Therefore, only one test case of each partition is needed to evaluate the behavior of the program for the related partition. In other words, it is sufficient to select one test case out of each partition to check the behavior of the program. Thus, the number of test cases can be reduced considerably.

In the context of IDS evaluation, an IDS should behave (i.e., detect or not detect) in the same way against attacks that belong to the same class. However, we should distinguish between two different situations. On the one hand, we can assume that we have a perfect classification, and then according to the concept of class equivalence, the IDS will ideally behave similarly for all attack instances belonging to the same class. Otherwise, i.e. if the IDS behaves differently for some instances of the class, we can conclude that the IDS has a problem of implementation and/or configuration (e.g., lack of corresponding signatures).

On the other hand, we can assume that neither the IDS nor the classification is perfect and this is likely the case because the classification scheme is still in a preliminary state. At this stage, it may have a weak equivalence relationship within its classes. Therefore, we suggest using it as a guideline for selecting test cases. Further, to ensure the representativeness of attacks in the test suite, evaluators should use several instances of attacks of the same class then elaborate statistics on the detection/non-detection results. This will enhance the quality of results with the price of increasing the number of test cases. In Chapter 6, we will explain how the classification scheme proposed in this chapter can be used to select relevant attack test cases and how this can facilitate and improve the analysis of the evaluation results.

4.5. Scenario-Level Attack Characterization (Attack Process Model)

In addition to the characteristics covered by the classification discussed above, there are functional features that need to be considered. To characterize attacks at a high level, our approach consists of analyzing known attack scenarios, and then constructing a model that can abstractly describe the series of actions executed by real scenarios, even those scenarios that are still unknown.

In order to create a precise model, it is necessary to analyze a sufficient amount of information on the phenomena that we want to model, i.e., attack incidents. In fact, relying on insufficient data or limited-scope data on these phenomena could result in biased or unrealistic models.

Unfortunately, as opposed to the antivirus community that has a large corpus of data to analyze (i.e., captured malware), the intrusion detection community suffers from an acute lack of data about security incidents (attacks, intrusions, etc.). Nevertheless, despite insufficient comprehensive information about such security incidents, a preliminary analysis can be carried out based on the revealed ones as well as known viruses and worms. Since worms are self-propagating and self-contained attacks, the whole processes of attack can be deduced by analyzing their codes. This is relatively easy, because security analysts are usually able to capture worm binary code and disassemble it. Besides, if we admit that worms, viruses and Trojans are some kind of automated attacks that are developed by skillful attackers, we can anticipate how manual, human-interactive attacks could occur.

We present in this section an attack process model that has been constructed by analyzing many worms, viruses, Trojans as well as other attack incidents. This analysis takes a high-level viewpoint to extract the type and the sequence of actions that each attack executes. In the following sections, we present samples of the analyzed attacks.

4.5.1. Malware Analysis

Most of the attacks that we have analyzed are described in the MITRE’s Common Malware Enumeration list (*CME*) {Cme08}. At the time of writing this work, the CME list contains 39 malware samples, representing the most famous and most dangerous ones. Furthermore, we relied on other information available at specialized web sites like <http://research.eeye.com>, and <http://www.viruslist.com>.

First, we have enumerated the execution patterns of the analyzed attacks. We found that attack steps can be categorized in eight patterns. Each attack step is given a unique symbol as follows (the meaning of attack steps is further explained in Section 4.5.3):

- R: Reconnaissance
- VB: Victim Browsing
- EP: Execute Program
- GA: Gain Access
- IMC: Implant Malicious Code
- CDI: Compromise Data Integrity
- DoS: Denial of Service
- HT: Hide Traces

It is worth noting that we are more interested in the attack process itself rather than in the fine-grain details such as the particular command or instruction used to overwrite a buffer or the exact code sequences. Moreover, to avoid the biasing effect of focusing only on worms, we give little attention to the self-propagation feature and sum it up as an implicit sequence of “gain access” and “implant malicious code” steps. To be consistent throughout the analysis of the whole attack list, we followed a set of predefined directives. First, it is clear that almost all attacks are a kind of program execution (EP). Therefore, to guide the assignment of abstract steps, we defined several partial order relations (denoted by $>$) to determine which category of execution pattern should be assigned to an attack step:

IMC $>$ CDI $>$ EP

HT $>$ CDI $>$ EP

[R|VB] $>$ EP

HT $>$ DoS

This means that:

- If the executed attack step contributes to the installation of the malicious code, it is classified as IMC. Else, if it modifies the file system, the configuration files, the registry keys or the environment variables, it should be considered as a CDI. Otherwise, it is considered as EP.
- If the executed attack step hides or blocks access to information about the malicious code, it is classified as HT. Else, if it modifies the file system, the configuration files, the registry keys or the environment variables, without hiding information related to the attack, it will be considered as a CDI. Otherwise, we consider it as EP.
- Searching information remotely from the victim or a potential victim is a reconnaissance step (R), while searching information locally on the victim system is a Victim Browsing step (VB).
- If the attack step blocks/stops/compromises access to data or services that provide information about the malicious activity, it is a trace-hiding step (HT). If the blocked/stopped/compromised service does not hide such information, we consider it as a DoS step.

With this step classification, let us analyze some typical malware scenarios.

A) CodeRed-I

As illustrated in Figure IV.4, the Code Red-I worm begins by hitting a vulnerable IIS web server with an HTTP *Get* request that contains the necessary code to exploit the *ida vulnerability* {Ida07}, {Eeye08}. This vulnerability allows the worm to install itself by writing the malicious code into the memory (Gain Access and Implant Malicious Code, denoted below GA and IMC respectively). The code initializes the worm variables and locates the addresses of the functions within the *.dll* files loaded in the memory (Execute Program, EP). Then, it creates 99 threads (EP) to spread further the worm by infecting other web servers. The 100th thread checks the language of the running windows (Victim Browsing, VB). If it is an English Windows NT/2000, it will proceed to deface the infected system's website (Compromise Data Integrity, CDI). If it is a non-English Windows, the 100th thread continues spreading the worm (EP). Each worm thread checks for the file *C:\notworm* (VB). If it exists, the worm goes dormant. Otherwise, it continues infecting more systems (EP). It checks the current date (VB). If the date is not between 1st and the 19th of the month, this worm thread tries to propagate to *www.whitehouse.gov* (denial of service attack, DoS).

Accordingly, we can thus conclude that the attack process of CodeRed-I is represented as follows: GA, IMC, EP, EP, VB, [CDI, EP], VB, EP, VB, DoS.

B) CodeRed-II

Code Red-II exploits the same vulnerability as CodeRed-I. However, it uses different mechanisms to propagate the infection and to implant a Trojan [Ida07], [Eeye08]. The first two steps (i.e., GA and IMC) are similar to those of CodeRed-I, see Figure IV.5. The worm then gets the local IP address (VB), which will be used later to deal with subnet mask propagation. It gets also the local system language (VB). Moreover, it checks whether it has been executed before (VB) and if so, it proceeds to the propagation section (EP). The worm then creates 300 threads for non-Chinese systems and 600 for Chinese systems. Each thread is a new propagation thread that repeats the previous steps. After that, the worm searches the native systems directory (e.g., *C:\winnt\system32*) (VB). It copies the file "*cmd.exe*" to "*inetpub/script/root.exe*" and "*program~1/common~1/msadc/root.exe*" (CDI). An embedded binary within the worm is written out to *explorer.exe* (IMC). Finally, it reboots the infected system (EP). After reboot, the Trojan will be activated by modifying *Windows* registry keys to disable the file system protection and to create virtual web paths (CDI). As long as the Trojan *explorer.exe* is running, the attacker can access the infected server remotely.

Thus, the first part of the attack process of CodeRed-II can be represented as follows: *GA, IMC, VB, VB, VB, EP, VB, CDI, IMC, EP, CDI*.

The next time the attacker comes back and wants to connect to the infected system, it will be sufficient to send a request such as *http://IpAddress/c/winnt/system32/cmd.exe/?c+dir*, where *dir* could be any command the attacker would want to execute. The second part of the attack process is thus: *GA, EP*.

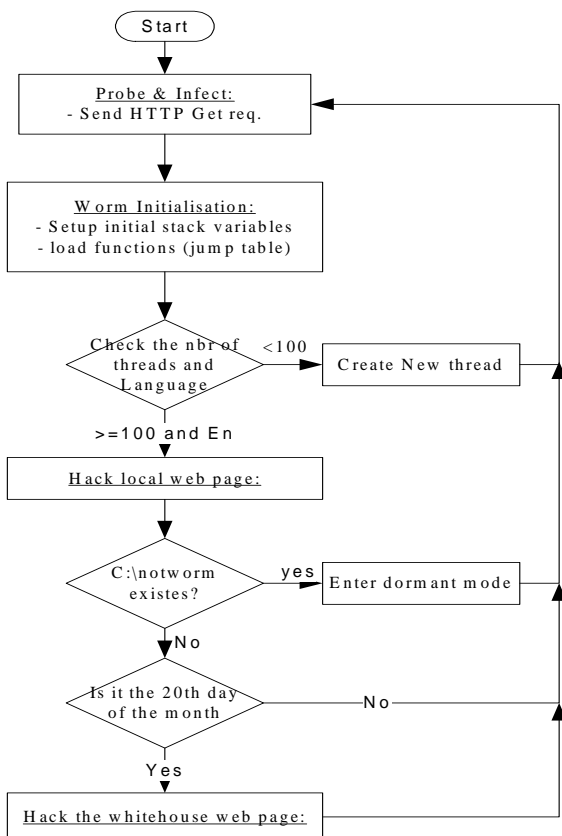


Figure IV.4: A simplified flowchart for CodeRed I.

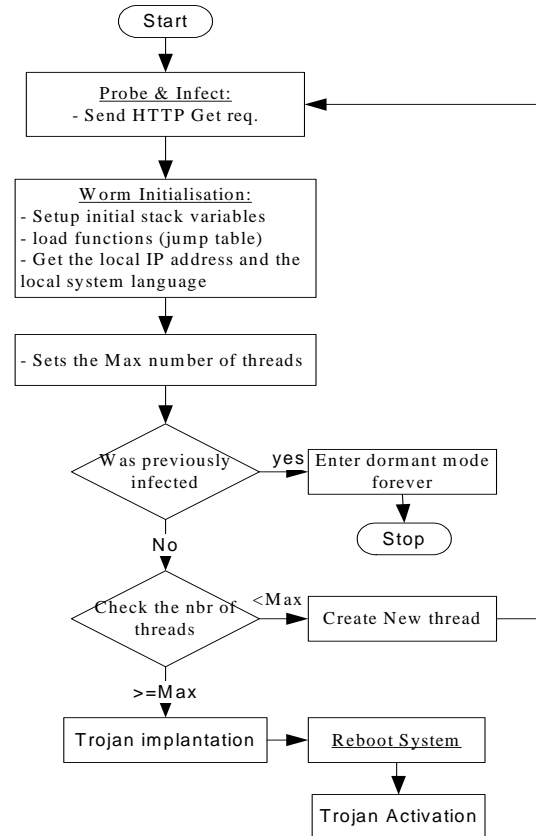


Figure IV.5: A simplified flowchart for CodeRed II.

C) SQL-Slammer (Sapphire)

The *SQL-Slammer* worm exploits a multiple buffer overflow (CVE_2002-0649) in *Microsoft SQL* server 2000 [Eeye08]. After overwriting the buffer, the worm takes control (GA and IMC). It begins to send itself to randomly generated IP addresses on the UDP port 1434 (EP) in an infinite loop (DoS). Therefore, the scenario executed by *SQL-Slammer* is: *GA, IMC, EP, DoS*

D) Sasser

Sasser uses a public exploit for the LSA vulnerability in order to obtain a SYSTEM-level command shell on its victims [Eeye08]. First, it searches for IP addresses listening to TCP port 445 (Reconnaissance, denoted R). Once found, it tries to figure out the Windows version through SMB banner (R). It then sends the LSA exploit that establishes a system shell connection on TCP port 9996 (GA). It creates a FTP script (CDI). When executed, the script connects back to the attacking host (EP) and downloads the worm EXE file (IMC). Finally, the worm is executed to start a FTP server thread and 128 propagation threads (EP) before deleting the FTP script (Hide Trace, HT). As shown in Figure IV.6, Sasser's scenario can be represented as follows: *R, R, GA, CDI, EP, IMC, EP, HT*.

E) Trinoo

Trinoo is a distributed Denial of Service (DDoS) attack [Trinoo07]. It searches first for IP addresses that have vulnerable RPC services (R). Then, it infects these hosts by exploiting the discovered vulnerabilities to create root command shells (GA). The infected host listens to TCP port 1524 (EP). It creates an installation script to automate the installation of the worm (CDI), and then uses *netcat* to pipe the script to the listening shells (EP); the script executes the command *rcp* to download the file *rcp.listen* and stores it at */usr/sbin/* directory (IMC); finally, it schedules the installed binary code for running with *crontab* (CDI).

The infected hosts are divided into masters and daemons. The attackers control one or more "master" servers, each of which can control many "daemons". The daemons are all instructed to coordinate a packet-based attack against one or more victim systems (DoS). The scenario corresponding to Figure IV.7 of Trinoo's flowchart is: *R, GA, EP, CDI, EP, IMC, CDI, DoS*.

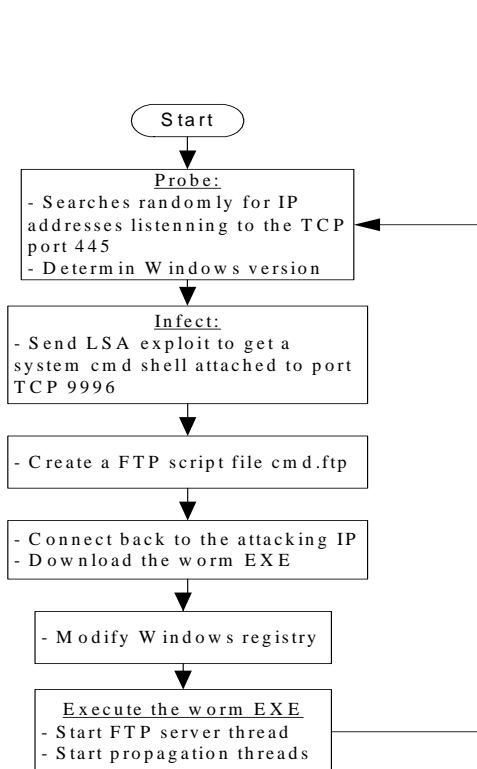


Figure IV.6: A simplified flowchart for Sasser.

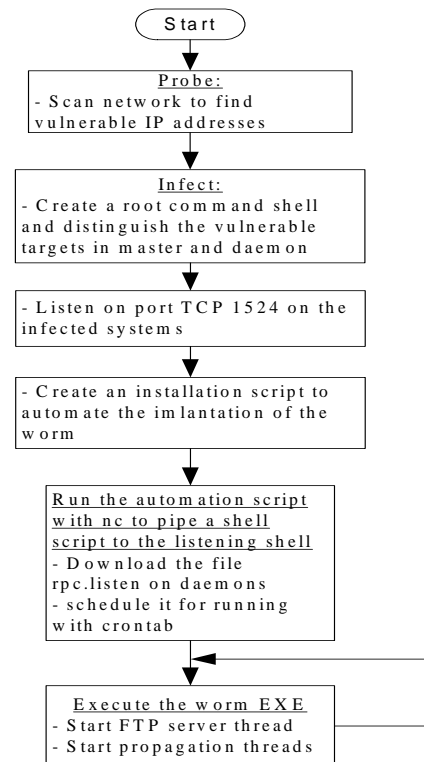


Figure IV.7: A simplified flowchart for Trinoo.

4.5.2. Other Attacks

From these observations, it is clear that we have identified attack step patterns that can represent various instances of worm attacks. An important question can be raised here: is there a generic model that would be also valid for attack scenarios other than those corresponding to worms? To answer this question, we analyze here a sample of interactive attacks, to see if they could fit well with the same model.

A) SYN flooding

A SYN flood is a form of denial-of-service attack in which an attacker sends a succession of TCP SYN requests to a target system. The attacker ignores the SYN/ACK sent back by the server and does not acknowledge it. These half-open connections bind so many resources on the server that connections from legitimate users are no longer possible. The scenario of SYN flood is thus: *DoS*.

B) Mitnick

In fact, the Mitnick attack, which is very similar to session hijacking, consists of two parallel attacks against two different hosts, the two endpoints of a TCP session. Assume that there are two hosts called *A* and *B* that trust each other. The attacker initiates a SYN flood attack against host *A* to prevent it from responding to host *B* (Denial of Service, *DoS*). Then, the attacker sends multiple TCP SYN packets to host *B* (the target) and tries predicting the TCP sequence numbers generated by host *B* (Reconnaissance, *R*). The attacker then pretends to be host *A* by spoofing its IP address and sends a SYN packet to host *B* in order to establish a TCP session between Host *A* and Host *B* (Gain Access, *GA*). Host *B* responds with a SYN/ACK to host *A* that will not respond to this packet since its input queue is full due to the SYN Flood attack. The attacker, impersonating *A*, then responds to the SYN/ACK of host *B* with the right expected sequence number and establishes a TCP connection. Consequently, he can send data and execute commands on the established connection (Execute Program, *EP*). Therefore the attack scenarios against host *A* is: *DoS*, whereas the attack scenario against host *B* is: *R*, *GA*, *EP*.

C) Cross-Site Scripting (XSS)

Cross Site Scripting attacks are typically found in web applications that allow code injection by malicious web users into web pages viewed by other users. Let us consider an example of this attack.

Example: While a user is logged into his bank account, an attacker can send an email with some potentially interesting content and convince the user to click on a link in the email (Gain Access, *GA*). The link points to or contains a malicious script (Implant Malicious Code, *IMC*), probably within an *iFrame* that mimics an actual user form submission to perform a malicious activity, such as transferring funds from the victim's account. The attacker can have the script embedded in, or targeted by, the link to perform any arbitrary action with the authenticated user's privileges (Execute Program, *EP*). When this script is executed, the targeted application authenticates and accepts the actions based on the victims existing session cookie.

The scenario can be abstracted as *GA*, *IMC*, *EP*. Even if a reconnaissance step should be performed before conducting this attack, it is not mentioned in the abstract scenario because it is often not observed by the IDS. For example, the attacker needs to find information about the form contents that should be sent back to the bank server.

D) Phishing

Phishing is a form of information gathering or "fishing" for information by using social engineering techniques. An attacker masquerades as a legitimate entity in order to prompt the user to reveal some confidential information such as authentication credentials. An attacker can use the gathered information later. After the social engineering part of the attack, the attack stages could be to gather credentials (Reconnaissance, *R*), use the gathered credentials (Gain Access, *GA*) and play with the account to do whatever subsequent action (i.e., Victim Browsing, Compromise Data Integrity, Execute Program, etc.)

The automated-malware attack scenarios distinguish themselves from interactive scenarios by their propagation and spreading mechanism. The level at which the scenarios are realized is also different: worm steps are generally run at the level of system calls or APIs, whereas the scenarios steps of non-automated attacks occur more frequently at the application or program-execution level. Despite these differences, both automated and interactive attacks follow similar steps to achieve their goals, and thus should fit well a common model of attack process.

4.5.3. A Generic Attack Process Model

In addition to malware and attacks described above, we have analyzed about 40 malware samples. Because of space limitation, we cannot expose them all in this dissertation but some are presented in Appendix A.

Globally, the observations we obtained from these samples as well as worms and interactive attacks studied previously allowed us to construct the model shown in Figure IV.8. Surprisingly, we observed that, in the studied samples, attackers often follow nearly the same (kind of) steps whatever their experience is! But of course, attacks may vary in sophistication, code quality and damaging effects, which indirectly reflect the level of the attacker's knowledge and competence. For example, experienced attackers may use more advanced techniques or customized tools, whereas newbies (e.g., script kiddies) are using simple exploits developed by others.

In any case, all categories of attackers proceed with more or less steps in different orders, but all of them are complying with our generic attack process model (Figure IV.8):

- *Reconnaissance*: it is reasonable for an attacker to search for necessary information about potential victims and their characteristics before targeting them. This enables attackers to select appropriate attack tools, exploits, etc. that can defeat or paralyze the victim.
- *Gain access*: generally, attacking a victim's system implies that it is reachable by the adversary: to accomplish her/his goal, the attacker often needs to gain some access to some victim resources. The level of the required access differs according to the chosen attack. Note, however, that some types of attacks, such as DoS attacks, need little or no access at all.
- *Privilege escalation*: Sometimes, the access acquired initially by the attacker is not sufficient. Therefore, she/he performs privilege escalation activities that provide a more powerful access. For instance, some vulnerability may be exploited to pass from local user to root privileges.
- *Victim browsing*: after having acquired sufficient access, the attacker tries to explore the victim internals (e.g., browse folders and files, search through user accounts, identify hardware components, identify installed programs, search for trusted hosts, etc.).
- *Principal actions*: as shown in Figure IV.8, this stage can take different forms. For example, attackers can launch a DoS attack (if this is their main objective), implant/upload a malicious code, compromise data integrity or execute a program.
- *Hiding traces*: more experienced attackers usually carry out this step to erase traces of what they did on the victim and to make forensics more difficult.

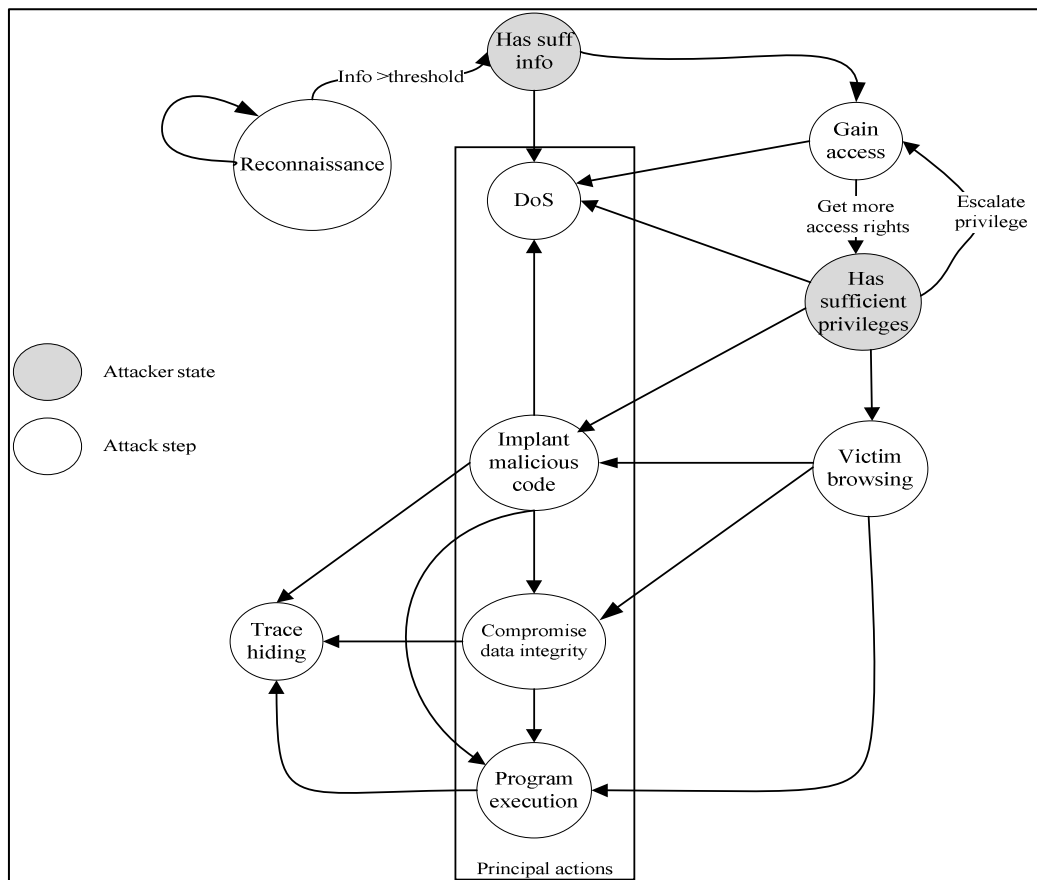


Figure IV.8: Attack process model.

It is worth noting that multi-hop attacks (that involve attacking several victims in sequence) are not represented directly by this model. However, they could be represented by concatenating individual scenarios (i.e., one scenario for each victim).

Actually, we developed this attack process model to help in generating high-level plans of attack scenarios. Of course, this “high-level” view is not sufficient to generate a real attack trace (e.g., at network or host level). Instead, it allows us generating abstract attack scenarios with the advantage of avoiding the combinatory explosion at this stage. To be able to generate executable attack scenarios, this generic attack model has to be complemented by the classification of attack tools (*cf.* Sections 4.6 for examples) and a model for attacker’s competence (*cf.* Sections 5.2.1). The complete approach will be explained in Chapter 5.

We expect that a model-based generation of attack scenarios will help in generating rich variations of more realistic attack scenarios than traditional IDS evaluations. Therefore, we are confident that it will have positive implications on evaluating IDSes, thus contributing to the improvement of IDSes.

4.6. Examples of Attack Activity Characterization

In order to show the applicability and the versatility of our approach, let us apply it on a set of attack tools. We consider not only purely malicious attack tools such as buffer overflow exploitation tools and password cracking but also other general purpose tools (i.e., system utilities, administrative tools, file transfer tools, etc.), which are non-malicious but can be used within the context of an attack session.

It is worth noting that we characterize elementary attack actions rather than composite attack scenarios. For example, there may be several entries corresponding to each function provided by the same tool for a multi-purpose attack tool that provides several functions. Similarly, when we characterize an attack tool that executes an attack scenario or a sequence of attack steps (according to the attack process model) corresponding to a partial scenario, we consider each step separately as an attack action even if it is carried out by the same tool.

Characterizing elementary attacks from both the classification perspective and the scenario perspective is a critical task on which we base the attack test-case selection and the generation of executable scenarios. Achieving this task allows also the selection of elementary attacks according to their classification-features as well as generating executable scenarios by instantiating abstract scenarios by attack tools according to their attack step. The “*Attack Step*” in each example is the key link that establishes the relation between the attack process model and the classified attacks.

In the following, we present, in more details, five examples for characterizing elementary attack actions. Each example represents an elementary attack activity, which is characterized by both its appearance (i.e., a classification perspective) as well as its attack step (i.e., a scenario perspective). Afterwards, we present a detailed characterization of *metasploit* attacks, in the next section.

Example 1: *Ping is a simple tool that is originally used for diagnosing network problems by network administrators. One of its common uses is to discover whether a certain IP address is in use, or a particular machine at this address is up. This ideally serves attackers to get valuable information about the potential victims.*

<u>Name:</u> <i>ping</i>			
Source:	Remote		<i>Ping</i> can be used locally by the attacker from the victim against other machines, either within or outside the local network. We characterize it here in the case when being used remotely to check a victim IP address.
Privileges acquired:	None		It does not provide any privilege (if used in the common way).
Vulnerability:	Configuration		Through this activity, attackers exploit networks that accept ICMP request packets coming from outside and allow ICMP reply.
Network carrier:	Transport Layer		The network traffic that results from ping is composed of ICMP packets, which is a transport layer protocol.
Local carrier:	None		No action is explicitly executed by the attacker on the targeted machine.
Target:	Network Stack		The entity activated on the machine victim is the network stack.
Attack step:	R		This particular use of ping represents a reconnaissance step as it aims to get information about remote machines.

Example 2: Connecting to a FTP server to transfer files is an example of normal activities that can be conducted by attackers. It can be used in two ways: (1) to connect to a victim machine if attackers manage to acquire valid user credentials (i.e., a user name and a password). (2) Attackers can also connect back from the victim machine to another machine to download a piece of malware.

We classify the source with respect to the victim. Therefore, in the first case, which is explained by this example, executing an ftp command is remote while in the second case it is local.

Name: establish ftp connection

Source: Remote	A remote attacker can connect to a victim FTP server if he has previously managed to obtain a valid user name and a password.
Privileges acquired: Variable	This activity may lead to different levels of privileges, depending on the used account (normal user or privileged user).
Vulnerability: Configuration	Generally, this attack succeeds by exploiting weak password policies or by exploiting a vulnerability in the configuration of an anonymous ftp server.
Network carrier: Application Layer	The main and effective elements of the activity are encapsulated in the packet workload reserved for the application layer.
Native carrier: None	No explicit invocation method executed on the local machine.
Target: Service	The activity targets the ftp server on the victim machine.
Attack step: GA	The purpose of executing the command <i>ftp</i> is to gain an ftp access to the victim machine. After connecting to the remote FTP server, attackers can execute further FTP commands, which in turn can be also characterized.

Example 3: Exploiting vulnerabilities in software applications on the victim machines is widely used by attackers to obtain access to these machines. Attackers can launch exploits against a victim machine from a remote or local access. Remote attackers usually target the applications exposed to network as services via known access points such as web servers, login servers, ftp servers, etc. Exploiting service vulnerabilities is usually used by attackers when they fail to obtain valid user accounts. Attackers who have already some local access on the victim machine can exploit other applications to promote their privilege levels. In this example, we pick two exploits from metasploit

A) metasploit/windows/dcerpc/ms03_026_dcom :

OSVDB-2100 Description: Windows platforms contain a flaw that may allow a remote attacker to execute arbitrary code. The issue is due to a flaw in the Remote Procedure Call (RPC) Distributed Component Object Model (DCOM) interface that does not properly sanitize remote requests. If an attacker sends a specially crafted message to the server, they may be able to crash the service or execute arbitrary code with SYSTEM privileges.

Name: metasploit/windows/dcerpc/ms03_026_dcom

Source: Remote	This exploit can be launched from a remote location.
Privileges acquired: System	Because the targeted service is executed with system privileges, it provides system privileges if the exploitation succeeds.
Vulnerability: Implement/Design	The exploit benefits from an error in validating the size or the boundaries of the input data.
Network carrier: Application Layer	The effective part of the attack is included in the application layer part of the sent packet.
Native carrier: Instruction	The exploits injects some instructions in memory to take control of the program execution.
Target: Operating System	Although the targeted software provides a service (RPC service), the target is the operating system because the RPC software module is implemented as a part of <i>Microsoft windows</i> (NT, 2K, XP, server 2003).
Attack step: GA	It provides an access gain to the machine and injects malicious code in the memory.

B) *metasploit/windows/ftp/3cdaemon_ftp_user*:

CVE-2005-0277 Description: Buffer overflow in the FTP service in 3Com 3C Daemon 2.0 revision 10 allows remote attackers to cause a denial of service (application crash) and execute arbitrary code via (1) a long username in the USER command or (2) an FTP command that contains a long argument, such as *cd*, *send*, or *ls*.

Name: *metasploit/windows/ftp/3cdaemon_ftp_user*

Source: Remote	This exploit can be launched from a remote location.
Privileges acquired: Variable	The provided privileges depend on the privileges of the execution privileges of the exploited service.
Vulnerability: Implement/Design	The exploit benefits from an error in validating the size or the boundaries of the input data.
Network carrier: Application Layer	As the effective part of the attack is included in the application layer part of the sent packet.
Native carrier: Execute Command	Exploit occurs when executing FTP commands with long arguments.
Target: Service	The exploited software is 3Com 3C Daemon which is an FTP service.
Attack step: GA	It provides an access gain to the machine and allows injecting malicious code in the memory.

Example 4: The SYN Flood attack is a type of denial of service attack in which a large number of TCP SYN packets (the first packet in a TCP/IP connection), usually with spoofed source IP addresses, are sent to a target. The target system replies with the corresponding ACK packets and waits for the final packet of the TCP/IP three-way handshake. Because the source IP address of the initial packet was spoofed, the target never will receive the final packet, leaving it to hold TCP/IP sessions open until they time out. A SYN flood causes so many TCP/IP open sessions that the system becomes overwhelmed and cannot handle any more network traffic. A simple tool that implements a SYN flood attack can be characterized as follows:

Name: *SYN Flood Tool*

Source: Remote	SYN flood attacks can only be launched from outside the victim system.
Privileges acquired: None	No privileges neither required nor provided upon the execution of this attack only.
Vulnerability: implementation/design	Machines are vulnerable for this type of attack because the early design of the TCP/IP protocols was for a friendly Internet. Later implementations of the TCP protocol have reduced the queue length or the connection timeouts to prevent such attacks from succeeding.
Network carrier: Transport Layer	The effective part of the attack can be found at the transport layer (the SYN field of the TCP header) of the sent packet.
Native carrier: None	No native action is explicitly invoked on the local machine, except opening a TCP connection.
Target: Network stack	It targets directly a feature of the network stack.
Attack step: DoS	The objective of SYN flood is to cause a denial of service.

Example 5: Attacks that result from automatic malware such as those we have analyzed in Section 4.5 are different from the attack tools we have analyzed above because they are not simple attack activities. Rather, they are attack scenarios comprised of several elementary activities. Therefore, we cannot characterize the entire attack at once but we should characterize each elementary activity separately. For example, a Slammer attack that corresponds to the scenario (GA, IMC, EP, DoS) can be characterized as follows:

Name: Slammer_GA, IMC

Source: Remote	The worm can propagate from infected machines to other remote ones.
Privileges acquired: System	Slammer targets <i>Microsoft SQL</i> service that is executed with system privileges, it provides system privileges.
Vulnerability: Implement/Design	The vulnerability is due an improper handling of data by SQL.
Network carrier: Application Layer	The worm code is transmitted through the application layer part of the sent packet.
Native carrier: Instruction	The exploits injects some instructions into the memory to take control of the program execution.
Target: Service	It targets server software (i.e., <i>MS SQL</i> server).
Attack step: GA,IMC	It provides an access gain to the machine and injects malicious code in the memory.

Name: Slammer_EP

Source: Local	After implanting malicious code of the worm in the previous step, the worm can be executed locally.
Privileges acquired: None	The worm execution keeps the system privilege without any new escalation.
Vulnerability: None	It exploits no specific vulnerability at this stage.
Network carrier: None	The worm generates no network traffic while it calculates pseudo-random IP addresses that will be attacked in the next step.
Native carrier: Instruction	The worm executes instructions to retrieves the address of <i>GetProcAddress</i> and <i>Loadlibrary</i> from the IAT in <i>sqlsort.dll</i> . It then can obtain necessary library base addresses and functions entry-points.
Target: Service	It continues targeting <i>dll</i> libraries loaded in the memory.
Attack step: EP	The core actions of the worm are executed in this step.

Name: Slammer_DoS

Source: Local	We take the point of view of the infected machine, which executes Slammer and sends a flood of requests to another remote victim.
Privileges acquired: None	The worm gains no new escalation.
Vulnerability: Configuration	The exploit benefits from an error in validating the size or the boundaries of the input data.
Network carrier: Application Layer	The worm payload is sent in a <i>SQL Server Resolution Service</i> request to the pseudo-random target address.
Native carrier: Socket Communication	The worm creates a UDP socket on the port 1434.
Target: Network stack	It targets software that is a part of the operating systems.
Attack step: DoS	The excessive load due to the request flood gets down the infected machine and consumes the network bandwidth.

4.7. Metasploit Characterization

After identifying the main characteristics of attack incidents and malware that exist in the wild, an important question may arise: to what extent evaluation datasets currently used in IDS evaluations match the characteristic of real-world attacks. To answer this question, we have characterized the attacks present in *Metasploit* {Metasploit08}.

We focus on *Metasploit* because of its popularity as a penetration-testing tool that is increasingly employed by IDS evaluators and we need to know if its attacks comply with real-world attacks. Moreover, we have implemented a proof-of-concept of our approach based on *metasploit*. Thus, describing it in more details is important to understand the solution that we propose in the next chapter.

4.7.1. An Overview of Metasploit Framework

The main purpose of *Metasploit* framework is to facilitate developing and executing exploit code against a remote target machine. It consists of several modules that together constitute a powerful exploitation framework. The main modules are the *Exploit module* and the *payload module*. The first is a particular code module responsible for exploiting a specific vulnerability. An *exploit* simply triggers the vulnerable condition and does not provide any shellcode¹⁶ (shell for short) or advanced encodings. Shellcode, encodings and NOP sleds¹⁷ are provided by other pluggable modules. The second is the *Payload module*, which is sent along with the exploit. It encapsulates the desired functionality of the attack.

Once the vulnerability has been triggered by the exploit, the payload performs some actions on the victim computer, such as yielding access to a command shell or downloading a backdoor installer.

The attacks carried out by *Metasploit* are highly configurable. Both exploits and payloads may have options available to configure their behavior. For example, an exploit might provide different variants based on the RPORT (Remote Port) or TARGET options. Similarly, a payload might allow choosing the port to open a shell using the LPORT (Local Port) option. Another special type of options is the “Evasion option”, which tweaks various settings to evade detection by an IDS or IPS. Moreover, *Encoders* can be used to bypass filters imposed by security countermeasures. For example, common limitations are the need to avoid null (0X00) characters, or the need to use only alphanumeric characters. *Metasploit*’s provides several encoders that circumvent these limitations while preserving the exploit and its payload.

Decoupling vulnerabilities and payloads is one of the core purposes of *Metasploit*. In general, *Metasploit* tries to make all payloads available to all exploits. However, depending on the nature of the particular vulnerability and target host, certain payloads may not be available for certain exploits.

The number of available exploits and payloads increases within the new versions. At this moment, the *Metasploit* 3.1 contains 258 exploits and 117 different payloads. This may seem a lot, but there are really only seven types of payloads. The large number of payloads is due to the small changes required in the actual shellcode in order to handle various use cases or target platforms. The seven payload types that *Metasploit* provides for each platform are shown in Table IV.2.

Table IV.2: Available payloads for each platform (metasploit version 3.1).

	Windows	Linux	MacOS X	BSD	Solaris	Unix
VNC injection	X					
File execution	X					
Interactive shell	X	X	X	X	X	X
Meterpreter	X					
Command execution	X	X	X	X	X	X
DLL injection	X					
Add user	X	X				

¹⁶ Low-level machine instructions specific to a particular operating system and CPU

¹⁷ No-Operation instructions inserted in shellcodes

Some payload types may have several variants. For example, windows shell payload has the variants shown in Table IV.3. The difference between the payload variants shown in this table is the type of network connection used to relay the shell commands. Depending on the network topology in place around the target computer, some payload variants may succeed where others may fail.

Table IV.3: Variants of shell payloads for Windows (metasploit version 3.1).

Payload name	Description
windows/shell/bind_tcp	Windows Command Shell, Bind TCP Stager
windows/shell/find_tag	Windows Command Shell, Find Tag Ordinal Stager
windows/shell/reverse_http	Windows Command Shell, PassiveX Reverse HTTP
windows/shell/reverse_ord_tcp	Windows Command Shell, Reverse Ordinal TCP Stager
windows/shell/reverse_tcp	Windows Command Shell, Reverse TCP Stager
windows/shell_bind_tcp	Windows Command Shell, Bind TCP Inline
windows/shell_reverse_tcp	Windows Command Shell, Reverse TCP Inline

4.7.2. Exploit Characterization

In order to be able to select attack test cases and to generate attack scenarios, we need to characterize tools that can be used by attackers during their attack sessions. *Metasploit*, is one of such tools that we have characterized. To achieve that, we have classified all exploits according to our classification scheme and the step they realize in the attack process model.

From the scenario viewpoint, it is clear that all exploits aim to benefit from some vulnerability to enable attackers to access vulnerable hosts and run a payload. An exploit cannot be executed without a payload, which complements its functionality, and the payloads available in the *metasploit* framework provide different kinds of access gain (i.e., bind_tcp, reverse http, etc). Therefore, we can consider all attacks issued from *metasploit* as “GA” attacks.

On the other hand, we have classified about 258 exploits that are available in the current version of *metasploit* framework (at the time of this work). According to our classification scheme, we found that the exploits belong to 16 categories. Figure IV.9 illustrates the categories and their percentages. The largest four categories contain 103 exploits (39.92%), 81 exploits (31.4%), 17 exploits (6.59%) and 15 exploits (5.81%) respectively. There are two categories with 9 exploits (3.49%) each. Three categories have 4 exploits (1.55 %) and four categories with only one exploit (0.39%). We give, hereafter, the description of the largest category and one of the smallest categories:

Name: Cat 1127 <ul style="list-style-type: none"> Firing source: Remote Vulnerability: Implementation/Design Privilege escalation: Variable Carrier: <ul style="list-style-type: none"> Network: Application Layer Native action: None Target: Service 	Name: Cat 1429 <ul style="list-style-type: none"> Firing source: Remote Vulnerability: Implementation/Design Privilege escalation: System Carrier: <ul style="list-style-type: none"> Network: Application Layer Native action: None Target: Operating System
---	--

A limited number of categories can be found in *metasploit*. According to our classification scheme, only 16 categories out from 3400 (the number of all valid combinations, see Section 4.4.4), have been observed. This is perfectly consistent with the nature of the exploits included in the *metasploit* framework, which is mainly a tool for penetrating into vulnerable machines from a remote location. Thus, it is not strange to find almost all categories have a remote *firing source* and implementation/design *vulnerability*. Moreover, this confirms the risk of using *metasploit* as a single source of attack test cases in IDS evaluations.

Furthermore, we have, surprisingly observed a limited diversity in the *network carrier* attribute. It often lies in the application layer with few instances in the data link layer. The diversity comes from the *privileges* and the *target* attributes where exploits distributes well on almost all types within these two attributes (i.e., user, system, root, etc. and OS, service, application, etc.).

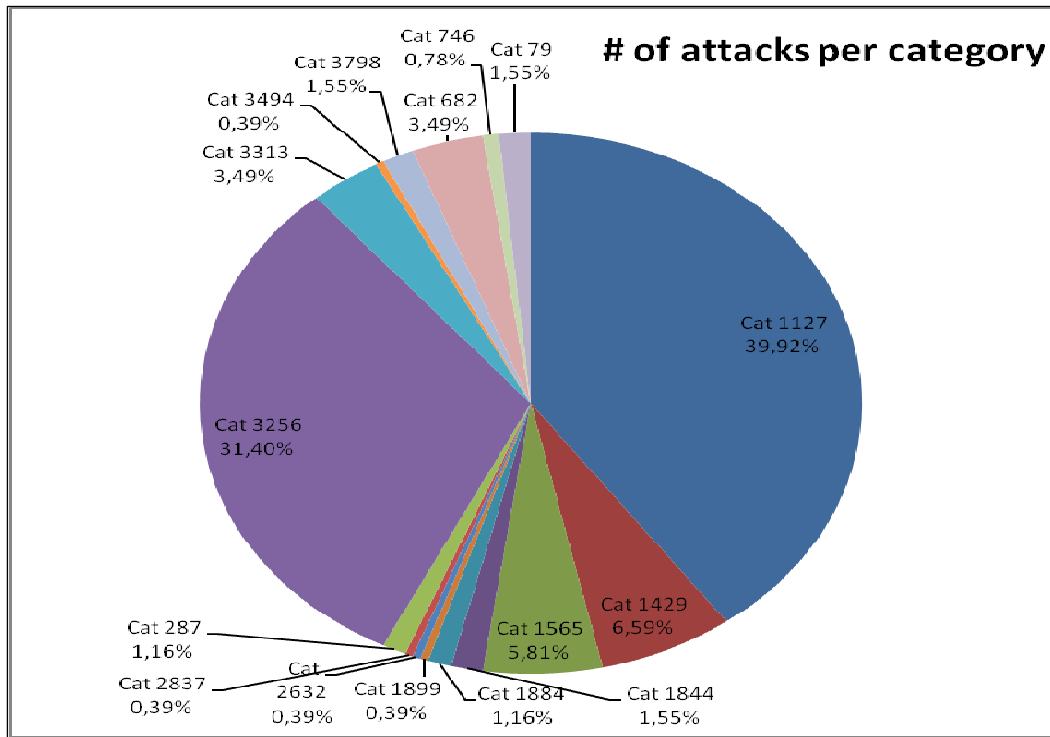


Figure IV.9: Classification-based characterization of *metasploit*'s exploits.

Having described how elementary attacks that comprise attack scenarios (that in turn, comprises the attack dataset) can be characterized, we proceed to the characterization of the background dataset.

4.8. Characterizing Background Activities

Background activities are composed of normal, benign activities that present no security risk. Such activities may generate several forms of event data: the most common forms are either network packet flows in the case of a network-based-IDS (NIDS) or log entries in the case of a host-based IDS (HIDS).

In this section, we identify the security-relevant characteristics of background activities in both network traffic traces as well as log file entries.

4.8.1. Network Activity Characterization

The Internet exhibits a wide range of different behaviors that are continuously changing. Consequently, it is difficult to characterize packet dynamics. This is the reason why the notion of a “typical” Internet traffic does not exist {Paxson97-a}.

However, when we examine traffic traces of TCP/IP networks, we can observe many features at different levels. Some of these features are related to the design or the implementation of the network layers, protocols, routing algorithms, congestion and retransmission mechanisms, etc. Other features can be environment dependent and closely related to the nature of the network use (i.e., academic, commercial, military, etc.) or to the network state (e.g., servers or routers breakdowns or malfunctioning).

All these features could be important when characterizing the generated traffic within a particular network. This has been the subject of hundreds of papers on networking, especially those issued from network metrology projects such as the CAIDA project {Caida08}. In the following, we are interested only in security-relevant characteristics amongst these features. More details about network traffic characteristics are out-of-the-scope of this chapter but can be found in {Paxson97} and {Thompson97} for example.

A) Overall Characteristics

A network-based IDS (NIDS) continuously observes packet flows that compose the network traffic. During NIDS evaluation experiments, we need to feed the NIDS in its experimental environment with

packet flows that are similar to those observed during the real IDS operation. These packet flows can be traces of traffic previously recorded on a real network, or synthetic traffic traces produced by a traffic generator, or some mix between both. These traces can be described according to different characteristics, such as

- *Traffic rate*: number of packets per second
- *Number of packets*: the number of packets recorded within a trace.
- *Number of sessions*: the number of sessions embedded within the trace.
- *Traffic composition*: what protocols and applications have generated the packets present in the traffic trace and in which proportions? What are the dominant protocols or applications (e.g., web, data transfer).
- *Packet retransmission ratio*: the number of retransmitted packets divided by the number of all packets in the trace.

A recent study of current trends in enterprise traffic {Pang05} showed that internal traffic has a higher volume (about 80 % of the overall traffic) with a richer set of applications when compared to the WAN traffic. Moreover, it was observed that the traffic corresponding to network file systems and backup applications dominates the internal traffic whereas email and web applications dominate the wide area traffic. Despite the fact that this study was limited to a single enterprise network, it shed some light on the traffic composition on medium-sized enterprise networks.

B) Temporal Characteristics

This set of characteristics regroups all features related to time. This may include:

- *Packet inter-arrival time*: distribution of time intervals between subsequent packets in the same flow (i.e., sent by the same IP address and destined for the same IP address in the same session).
- *Packet delays*: distribution of time delays to transmit packets from source to destination.
- *Session durations*: distribution of the time elapsed between the first packet in a session and the last packet in the same session.

C) Protocol Characteristics

Amongst the features related to protocols, we can enumerate the following:

- *Traffic types*: flows of packets produced by particular protocols present in the trace.
- *Sessions*: corresponds to TCP sessions, or sets of UDP packets exchanged between two IP addresses in a limited period of time.
- *Flags*: state of the flags in packet headers.
- *Packet length*: size of packet in bytes.
- *Payload*: type and contents of data sent in the packet payload.

D) Addressing Characteristics

Addressing features can be found at different layers of network protocols, such as the data link layer, the network layer and the transport layer. We can find different forms of addressing such as:

- *MAC address*: physical address of the network connection device.
- *IP address*: logical address used for routing in the network layer.
- *Netmask*: determines the sub-networks by using CIDR (Classless Inter-Domain Routing).
- *Internal/external address*: an address is internal if it belongs to the address space of the user's network (determined by the user) or if it is a reserved private address (non routable addresses: 10.0.0.0/8 CIDR block, i.e. 10.0.0.0 to 10.255.255.255, 172.16.0.0/9, i.e. 172.16.0.0 to 172.31.255.255, 192.168.0.0/16, i.e. 192.168.0.0 to 192.168.255.255)
- *Ports* can help identifying services provided by hosts connected to this segment as well as distinguishing servers and clients.

We do not intend to give an exhaustive list of all network traffic characteristics that can affect the operation of security tools, i.e., IDSes. Rather, we cited above the most relevant ones that have proved to be as such by empirical studies {Schaelicke03} or that are involved in the analysis and detection process carried out by NIDS. For example, modern NIDSes have protocol analyzers that search for anomalies in

packet headers at different layers. Therefore, we classify header fields as very significant data. Similarly, anomaly-based NIDSes that use learning techniques can easily detect inconsistencies in timing or sequence numbers.

4.8.2. Host Activity Characterization

Unlike NIDSes that have a unique source of event data (i.e., network traffic), host-based IDS (HIDS) can monitor and analyze a wide range of event-data sources such as operating system logs, application logs, web server logs, system calls, resource utilization, process invocations, etc. Moreover, HIDS are usually hybrid tools that monitor several activities. For example, *OSSEC* and *Samhain* monitor not only logs but they also perform integrity checking, *Windows* registry monitoring, rootkit detection, etc. {Samhain07}, {Ossec08}. This makes the characterization of HIDS workload more difficult.

Most of the event-data cited above can be reported to and recorded by a central *syslog* (or alike) server. Therefore, we concentrate here on the characterization of log files collected by *syslog* daemon as a typical workload of HIDS, because of its popularity and availability for several heterogeneous platforms, *UNIX* systems, switches and routers, firewalls, printers, *windows* NT/2K/XP (with tools like *Ntsyslog*), etc.

However, even if we limit our scope to logs recorded by *syslog*, their characterization is not trivial because logs come from many applications of different types and purposes. Consequently, as we will see in the next section, log contents and formats are not necessarily the same for all entries, even if they share some common characteristics.

For this reason, we look only at the common characteristics and the structure of *syslog* entries. Characterizing more precisely some logs of particular types (e.g., web server logs) is left to HIDS evaluators, if they are more interested in certain application types, as this is out of our scope in this chapter.

A) Common Structure of Syslog Entries

Syslog is a consolidated audit mechanism that is designed to simplify the task of system analysts and application developers by providing a single point of management for collecting and distributing audit data. It permits both local and remote log collection.

Most users create *syslog* messages through one of the standard *syslog* interfaces. For example, there is a C library contained within *libc* to create message strings. The *syslog* library behaves somewhat like the standard *printf* () interface. Alternatively, one can invoke the command *logger* to create *syslog* messages from the command line or within a shell script.

A *syslog* message is an ASCII string that consists of a header and a message string. The header consists of a message priority and a timestamp. The message priority is an ordered pair <Facility, Level>. The facility part identifies the originating subsystem of the message. There is up to 32 facilities. Some are reserved for the OS and the others are available for users and application developers as shown in Table IV.4. Each *syslog* message is assigned one of a set of possible levels corresponding to its priority as shown in Table IV.5.

Table IV.4: List of some available facilities.

FACILITY	DESCRIPTION
LOG_KERN	kernel messages
LOG_USER	random user-level messages
LOG_MAIL	mail system
LOG_DAEMON	system daemons
LOG_AUTH	security/authorization messages
LOG_SYSLOG	messages generated internally by <i>syslogd</i>
LOG_LPR	line printer subsystem
LOG_NEWS	messages generated by the news system
LOG_UUCP	messages generated by the UUCP system
LOG_CRON	messages generated by the <i>cron</i> daemon
LOG_LOCAL[0-7]	reserved for local use

The message part of *syslog* messages that are emitted by devices or applications are typically in a plain text form. The message format is specified in “*syslog.h*” where its content is arbitrarily defined by

the emitting device or application. There is no common standard for message contents. Each log emitter uses its own format for message contents. Thus, it is common that different versions of the same emitter may produce different message contents.

Table IV.5: List of possible logging levels.

LEVEL	CODE	DESCRIPTION
LOG_EMERG	0	kernel panic
LOG_ALERT	1	condition needing immediate attention
LOG_CRIT	2	critical conditions
LOG_ERR	3	errors
LOG_WARNING	4	warning messages
LOG_NOTICE	5	not an error, but may need attention
LOG_INFO	6	informational messages
LOG_DEBUG	7	when debugging a system

Within the same category, there are two orthogonal differences with respect to the message string. The first is related to the kind of data or information logged by applications or appliances (OS events, firewalls, etc.) where nearly the same kind of data is reported. The second is related to variations due to the level of reporting details, the order and the organization of the reported data, etc.

Syslog messages received by the *syslogd* daemon can be written directly to the console or to a log file after stripping out message priority and adding the name of the system (equivalent to *uname -n*) that originates the message. This can be the local system if the message was locally generated, or a remote system communicating over an Internet domain socket. As an example, Figures IV.10 and IV.11 illustrate sample *syslog* entries generated by an apache server.

In order to be analyzed by a HIDS, for each log category (e.g., kernel logs, web server logs, firewall logs, etc.) a normalization or preprocessing step is usually carried out on the logs. This allows identifying and extracting the identical types of log entries regardless of their position or their form in the original log.

```
10.0.0.153-[12/Mar/2004:12:23:41-0800] "GET /dccstats/stats-hashes.1month.png HTTP/1.1" 200 1636
10.0.0.153-[12/Mar/2004:12:23:41 -0800] "GET /dccstats/stats-spam.1year.png HTTP/1.1" 200 2262
10.0.0.153-[12/Mar/2004:12:23:41-0800]"GET /dccstats/stats-spam-ratio.1year.png HTTP/1.1" 200 1906
10.0.0.153-[12/Mar/2004:12:23:41 -0800] "GET /dccstats/stats-hashes.1year.png HTTP/1.1" 200 1582
216.139.185.45-[12/Mar/2004:13:04:01 -0800] "GET /mailman/listinfo/webber HTTP/1.1" 200 6051
pd95f99f2.dip.t-dialin.net -[12/Mar/2004:13:18:57 -0800] "GET /razor.html HTTP/1.1" 200 2869
d97082.upc-d.chello.nl-12/Mar/2004:13:25:45 -0800] "GET /SpamAssassin.html HTTP/1.1" 200 7368
```

Figure IV.10: An extract from apache access-log entries (source: www.monitorware.com).

```
[Mon Mar 8 14:54:56 2004] [info] [client 64.242.88.10] (104)Connection reset by peer: client stopped
connection before send body completed
[Tue Mar 9 13:49:05 2004] [info] [client 81.226.63.194] read request line timed out
[Wed Mar 10 11:45:51 2004] [info] [client 24.71.236.129] (104)Connection reset by peer: client stopped
connection before send body completed
[Thu Mar 11 02:27:34 2004] [error] [client 200.174.151.3] File does not exist:
/usr/local/mailman/archives/public/cipg/2003-november.txt
[Thu Mar 11 07:39:29 2004] [error] [client 140.113.179.131] File does not exist: /usr/local/apache/htdocs/M83A
```

Figure IV.11: An extract from apache error-log entries (source: www.monitorware.com).

At the end of the preprocessing stage, we have a set of well-defined log entries for each category of applications or devices. In addition to well-defined name/value pairs of properties that are specific to the kind of log, there are some common properties for each log entry such as the timestamp (date and time of generating the entry), the originating device, the process name and the process ID.

For example, web server logs, as defined in the W3C standard, consist of a single line for each request and that single line contains all information about the web request (The IP source, the request

method (e.g., POST or GET), etc. In addition to the common properties, it contains the URL being requested as well as the number of bytes received and sent while serving the request (see Figure IV.10).

B) Common Characteristics of Log Entries

From the previous section, we can distinguish three types of characteristics: time, amount and composition characteristics:

- **Time characteristics** are related to the time of event occurrence or to occurrence frequency. For example, during normal operation, there are often routine events that happen periodically or aperiodically. This may include events that happen every 5 minutes, but it may also cover events that happen only once a month or once a week. For example, a virus scan of the local disk can be scheduled for execution every Sunday afternoon.
- **Amount characteristics** describe how many events in general, or events of specific types, occur within a given period.
- **Composition characteristics** are related to the diversity of the events recorded in the log file, in particular sequences of events that usually occur together as a block corresponding to the same activity. For example, a normal operation of an FTP server may consist of a log entry for the connection initiation, other entries for the successful log in, file transfer and connection release.

The background component is very important for the evaluation of IDS and particularly for the evaluation of false positives. However, we will concentrate more on the attack activities, as the work in this area still needs more effort, whereas we can benefit from the work in other areas such as network metrology for the generation of background datasets.

4.9. Conclusion on attack workload characterization

We have identified the lack of representativeness of datasets as one of the serious limitations in existing evaluations of IDS. This problem is common for both the attack as well as the background components of the evaluation dataset. However, it is more serious for the attack component as it has a more direct and significant influence on the quality of detection provided by intrusion detection systems. To address this problem of the poor representativeness and to enable an easy selection of relevant attack test cases, we ought to characterize IDS workloads in the real world. First, we have characterized the main features of simple attack activities at a low level. As a result, we have proposed a new classification scheme that builds on previous classifications by keeping only evaluation-relevant attributes and discarding any ambiguous or useless attributes.

Moreover, when we began to characterize attack scenarios, we were confronted with the lack of sufficient information about attack incidents. To overcome this limitation, we have analyzed many automatic malware attacks as well as human-centric attack incidents. Analyzing attacks is not as easy as it may appear because of several obstacles that prevent a wide-scope analysis. This may not only affect the cross-vendor deployment of security tools such as IDS but also their evaluations. These obstacles are mainly due to the huge number of vulnerabilities and captured malware samples and the lack of a common nomenclature scheme that would be accepted by all experts.

Actually, there are no agreed-upon naming conventions for computer attack tools and malware instances. This complicates the analysis and the comparison of alarms generated by different IDSes because different IDSes often generate different alarms for the same type of attacks. This also makes the use of various types of IDSes and the correlation of their alarms more difficult.

Fortunately, there are fruitful efforts within the network security community to derive a common nomenclature for computer vulnerabilities. The most popular of these is the Common Vulnerabilities and Exposures List (CVE) {Cve08}, which is maintained by MITRE with inputs from a variety of security professionals worldwide. Another one is the OSVDB {Osvdb08}. Both are publicly available and searchable with different search options.

A still underway but less mature effort is also carried out by MITRE Corporation to create standardized repertories for attack patterns and malware instances. The Common Attack Pattern Enumeration and Classification (CAPEC) list {Capec08} considers attack patterns at a high level whereas the Common Malware Enumeration (CME) list {Cme08} considers malware instances with references to different names attributed by different vendors to the same piece of malware.

Despite these obstacles, we managed to define a model of attack process that describes the dynamics of attack scenario executions from a functional viewpoint. Both the classification and the attack process model contribute to the characterization of intrusion detection workload and both are intended to serve in generating representative attack scenarios.

To achieve that, we have analyzed and characterized many tools that may be used by attackers, with respect to the classification scheme and the attack process model, as explained in Section 4.6. Further, we have created a repository with a user-friendly front-end interface to store the information issued from the characterization in addition to other important information about the particular vulnerability exploited (i.e., *CVE* and *OSVDB* if any) and the particular product concerned by the attack (e.g., *apache* server version 2.2, *IE Explorer* version 6) about the classified attack tools. This repository will be integrated with other components for generating and recording datasets within our evaluation kit that will be presented in the next chapter.

The characterization of attack activities that we have described in this chapter is an angular brick in our approach. It is both critical for the selection of attack test cases, for the execution of attack scenarios as well as for the presentation of results. A detailed description of the complete approach to generate attack scenarios will be presented in the next chapter.

V. Chapter 5: Model-driven Dataset Generation

In the previous chapter, we have analyzed the input workloads for different IDS types. Since IDS workloads consist of attacks as well as normal activities, we divided the problem of generating test datasets into two subtasks: generating attack dataset and generating background dataset.

A typical attack dataset contains “pure” malicious activities that are supposed to be harmful by nature (i.e., using intrusive tools such as buffer-overflow exploits) or that abuse legitimate functions (e.g., launching a SYN flood attack). These are the activities that any good IDS should never miss and should properly identify and report. Beside the pure malicious activities, attacks often include activities that are apparently innocent such as using commands like *ls*, *pwd*, *who*, etc. to browse the victim machine. These activities may seem to be less dangerous, but they may be very significant for the detection process. Modern IDSes are becoming more state-full and are often equipped by correlation engines where their detection capabilities as well as the correlation accuracy depend heavily on the context. Therefore, to evaluate IDSes correctly, we should generate not only pure malicious attack actions but also other innocent-like actions that may be executed by attackers (in addition to the background traffic).

We tackle, in this chapter, the problem of generating evaluation datasets, and specially attack datasets. Based on the characterization of attack activities that we have presented in the previous chapter, we explain how the classification scheme and the attack process model can be combined with other models (attacker competence and statistical parameterization models) to generate workloads that are representative of real attack scenarios that contain both malicious and innocent-like activities. Moreover, we will discuss some important issues related to the generation of background datasets.

5.1. Introduction

Generating malicious activities that are representative of real attacks is very crucial to support evaluating and testing security tools. There are two ways to generate attacks for the purpose of IDS testing: real and virtual. Virtual attacks can be generated by tools that create sequences of crafted packets such that attack signatures will be identified. A possible use of this type of malicious traffic is in testing false alarm rates or the ability of NIDSes to distinguish real attacks from normal packets. IDS stimulators such as *Snot* {Snot07} belong to this category. The virtual attack method should be used with care in IDS evaluation because it generates only a small part of malicious attack activities, those that can correspond to specific signatures. For example, an attack dataset may be reduced to one packet that carries attack symptoms coded in an IDS signature while omitting the rest of attack traffic: it does not need to create complete TCP transactions, it can limit the traffic generation to one side of the communication, etc.

On the other hand, generating real attacks may be closer to the reality. However, it is a nontrivial task and unfortunately consumes a significant part of evaluator’s effort and time. To our knowledge, there is no systematic way to generate realistic attack scenarios automatically with a good coverage of real world attacks. Attacks are often generated manually in an ad-hoc manner.

The quality of the generated attacks, their complexity and their representativeness have a strong impact on the results of the evaluation, the correctness of its conclusions as well as the credibility of the evaluators. The best solution, in our opinion, is to automate the attack generation process while keeping it as flexible and as extensible as possible. The key method in this solution is to create a model of the attack process with robust theoretical basis that can express real attack scenarios.

Therefore, the first step in the solution is to find a suitable model for generating attack scenarios. Modeling attack scenarios can be difficult due to the large number of involved actors. They may have many diverse characteristics and opposite or even contradictory objectives (e.g., attacker *vs.* defender, or attacking tools *vs.* security countermeasures). They also have different relationships and dependencies with the surrounding systems and environment where they operate and interact with each other. This is why modeling such complex environments in a single model may be unsuitable to capture and to represent all the involved elements as well as their security-relevant behaviors.

In Chapter 4IV, we have analyzed several models of attacks found in the literature. This analysis shows that these models are unsuitable for our purpose (i.e., generating representative attack datasets) for three main reasons:

- 1- They have been often developed for analyzing a particular network, and thus they are too network-specific or environment-specific: such a model cannot be used for analyzing other networks or even other configurations of the same network, requiring to recreate the model for each evaluation.
- 2- They suffer from the combinatory explosion problem and from poor scalability. Almost all models suffer from this problem because they try to express all necessary features of the playground and players in a single model. This results in large-size models with poor scalability that are impractical for generating attacks.
- 3- They generally focus on particular parts of the attack process (i.e., gaining access, escalating privileges) while ignoring post-access activities.

This is why we propose a *multi-model approach* where various models represent specific issues from different views that are related to different entities involved in the attack process. The models are treated iteratively, starting from a high level of abstraction down to detailed, concrete attack actions. The iterative approach reduces the severity of the combinatorial explosion and the separation allows the resulting models to be of reasonable complexity. Furthermore, the modularity of this approach facilitates the modification and the refinement of models individually without affecting the other models. That way, we avoided the oversimplifying assumptions used previously to reduce the complexities inherent in a single large model.

In the following section, we will present an overview of our approach and the underlying models. Moreover, we present a brief survey of attack generation techniques. Then, in Section 5.3, we detail the main steps to transform these theoretical models into concrete attack scenarios. Section 5.4 describes the tool kit that we have implemented to generate evaluation datasets and Section 5.5 presents the conclusions on the generation of evaluation datasets.

5.2. Model-driven Attack-scenario Generation

As stated above, running real attack scenarios is more suitable to generate realistic attack datasets for IDS evaluations. Nevertheless, this requires a lot of effort in setting up a live network on which real exploits can be executed against real vulnerable servers. Moreover, only gathering attack tools and attack exploits is not sufficient to generate attack scenarios with characteristics close to real-world attacks.

To avoid these limitations, rather than developing a single large model we propose three models: 1) an attack process model, 2) an attacker competence model and 3) a statistical parameterization model. These models are intended to be sufficiently generic to allow generating attacks for different networks or systems; to be iterative, to avoid the combinatory explosion; and to represent attack scenarios that cover the whole process of attack.

In what remains of this chapter, we explain how the models that we propose for the attack process and attacker competence can be combined to generate realistic attack scenarios in a flexible and extensible manner, these scenarios can be instantiated with temporal and addressing characteristics issued from the statistical model.

5.2.1. Underlying Models

Our approach is based on three main models: the *attack process* model, the *attacker competence* model and the *statistical* model. The attack process model is derived by analyzing malware behavior and real attack incidents. The attacker competence model governs the instantiation of executable attack scenarios from the attack process model. Finally, the statistical model helps us to determine the frequency by which particular tools and commands may occur during attack sessions and accordingly generate evaluation datasets. In the following subsections, we describe each of these models.

A) Attack Process Model

This is the key model that aims to reproduce abstract attack scenarios similar to the scenarios that we encounter in the real world. The objective of considering firstly abstract scenarios is to alleviate the combinatory explosion inherent in scenario and graph computing. We have described the attack process model in Section 4.5.3 and we will explain in Section 5.3 how we use it to generate abstract scenarios.

B) Model of Attacker Competence

Security testing differs from other testing activities in that security testers are dealing with intelligent, creative attackers who often use systems in unexpected or unfamiliar ways. This has several implications: most importantly, an attacker might do things that an ordinary user would not do, such as entering thousands of characters to overflow buffers or embed commands or scripts in http request where simple textual fields are expected. Security testers should not only consider normal activities that can be carried out by attackers, but also actions that are far beyond the range of normal activities. Moreover, because IDS/IPS are increasingly going state-full, testers should also consider the context or the way by which these activities are carried out.

In addition, even if attackers can have many things in common, their behaviors may vary according to many factors such as their background, level of experience, skills, how much information they can obtain about the targeted system, etc. {Alata06}, {Alata07}.

Consequently, the model of attack process alone is not sufficient to capture all the features relevant to IDS evaluations; it should be coupled with a model of attacker competence. The attacker competence has a significant impact on the generated attack scenarios since it has a direct effect on:

- the nature of attack steps (e.g., trace hiding is generally made only by skillful attackers);
- the sequencing of attack events (e.g., an experienced attacker who knows what he does and what he searches, executes directly actions that achieve his goal and makes less victim browsing lately in the attack process);
- the tools used and implicitly the targeted vulnerabilities. Skillful attackers, by example, might use their own tools or customized versions of existing tools;
- the execution and typing of erroneous commands or bad command options; and
- the stealth execution of attacks to make detection more difficult where attackers apply several evasion techniques such as fragmentation, encoding, etc. Moreover, they can change intra-attack timing; concretely, it corresponds to the time of thinking or the stealthy execution over wide time slices.

Thus, we model attackers in terms of their skill-level (L), the set of tools in their possession $\{Tools\}$ and their IP address (IP): an attacker is represented by the tuple $ATR = (Level, \{Tools\}, IP)$. Attacker $Level$ can have one of three values: beginner (B), intermediate (I) and skillful (S).

C) Statistical Parameterization Model

Honeypots provide invaluable information about attacks such as attack arrival rate, most-attacked ports and services, etc. {Leurrecom08}, {Kaaniche06}. Such information can be used to generate attacks with similar statistical characteristics in the evaluation datasets. Furthermore, honeypots provide information about the most frequent sources of attacks, which may guide the assignment of attacking IP addresses in the evaluation dataset. Low-interaction honeypots are useful for collecting data about attack sources and most targeted services, whereas high-interaction honeypots provide more information on how attacks occur, which tools are used and how attackers behave {Honeynet04}.

High-interaction honeypots provide a wealth of information about post-compromise activities {Raynal04}. Other studies such as {Ermopoulos06} give indications about the usage frequency of normal commands. This allows the parameterization of frequencies in executing normal activities during attacks. For example, during the victim-browsing phase, the commands *ls* and *cd* are used more frequently than commands like *ps*.

To extract such statistical information, we relied on data recorded in the context of the study presented in {Alata07}. It is based on high-interaction honeypots developed at LAAS to experiment with SSH vulnerabilities such as easily guessable passwords. Mechanisms for monitoring attack activities were transparently implemented via a customized *linux* kernel. Other mechanisms were added to keep control over attack actions and to prevent multi-hop attacks against other systems.

Data were collected over approximately 27 months starting from 2006 where about one-million connection attempts and two thousand successful connections were observed. Even though the all attack sessions result from SSH connections, the collected data gives a clear insight of attackers' behaviors after obtaining access to victim machines. Moreover, the observations can be generalized for attack session other than SSH where the only difference is how the victim machines were initially accessed (GA step).

Figures V.1 and V.2 show the evolution in the number of commands executed during the observation period on two honeypot machines. Command numbers range from one command to 268 commands per session. Several interpretations may be proposed. For example, short sessions may be due to the use of

automatic tools in brute force dictionary attacks or a unintended connection that results from an error in typing the IP address and once discovering the error logging out {Ramsbrock07}, {Alata06}.

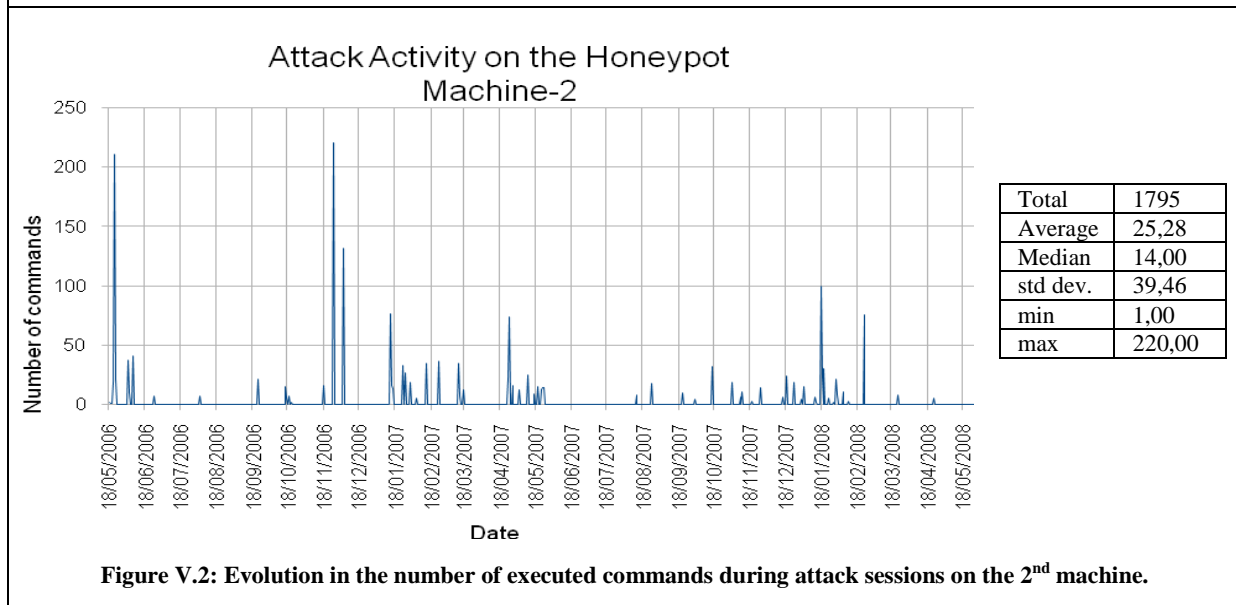
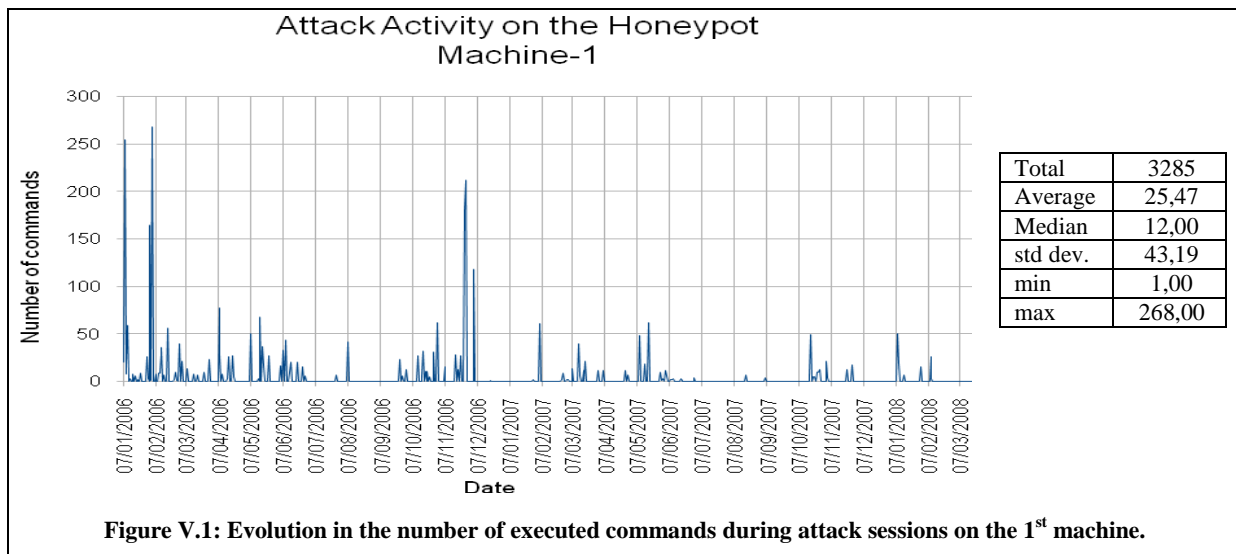


Table V.1: Most frequently used usernames and passwords.

Rank	User name	Password
1	test	test
2	admin	admin
3	root	root
4	guest	guest
5	root	123456
6	user	user
7	root	password
8	mysql	mysql
9	richard	richard
10	oracle	oracle
11	sales	sales
12	test	123456
13	web	web
14	ftp	ftp
15	michael	michael

Useful information about usernames and passwords that were used can be also extracted. Table V.1 shows the most frequently attempted username and passwords. Similarly, we have analyzed the commands executed during attack sessions.

As illustrated by Figure V.3, the most executed commands in this sample are *cd*, *ls*, *cat*, *rm* and *wget*. Furthermore, from the viewpoint of our attack process model, we noticed that almost all attackers browse intensively the victim machines (VB) to gain knowledge about their next steps. The second activity is the execute program (EP) and then comes the compromise of data integrity (CDI). When we closely examined the data, we discovered that this ranking was a bit biased because of the massive execution of *./john* command, which invokes a password cracking program (John the ripper) [John07]. Accordingly, the rank can be modified as follows: VB, CDI, IMC and then EP. Figure V.4 shows the percentages of each attack-step in the sample. We noticed that all the eight steps of the attack process are present in the sample. Besides that, there are 2.49 percent of the commands correspond to errors in typing commands (Typos). The *End* corresponds to session termination commands such as *exit* and *logout*.

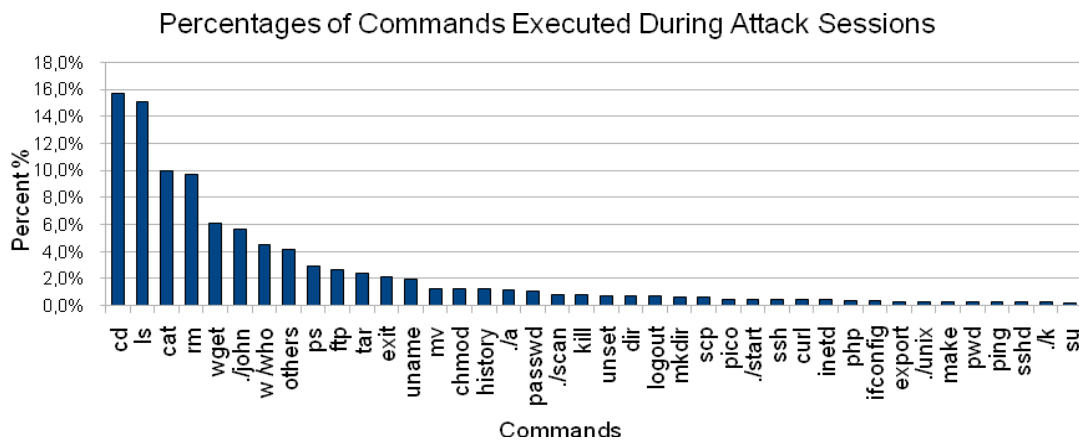


Figure V.3: Most frequently executed commands.

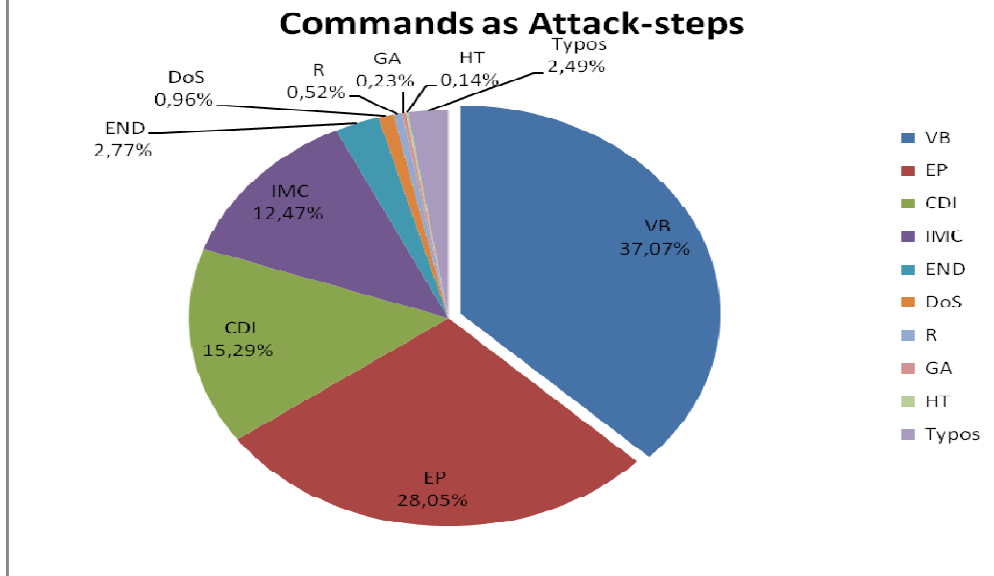


Figure V.4: Percentage of commands as attack steps.

Based on honeypot data, each attack tool or command can be assigned a weight value that reflects its usage frequency, its popularity as well as its obsolescence (e.g., in the case of vulnerability-exploiting attack tools, the weight decreases if the vulnerability is corrected by largely distributed patches or new software versions). The weight of a normal-activity command (W_c) corresponds to its utilization frequency in the real world ($W_c = f_c$) whereas the weight of malicious attack tools W_i can be computed as follows: $W_i = f_i/d_i$, where f_i is the frequency of utilization in the real-world and d_i is the number of months since the discovery date of the vulnerability exploited by this tool.

Now that the underlying models have been described, it is time to explain how we use them together to generate attack scenarios.

5.3. Novel Approach for Attack-scenario Generation

Let us recall from Chapter IV that IDS evaluation should inevitably pass through the following steps:

- characterizing real environment datasets, by analyzing the target environment where the IDS will be deployed as well as the characteristics of the evaluated IDS itself,
- identifying interesting test-cases,
- designing experimentations,
- generating test dataset and executing test-cases, and finally,
- analyzing results.

In the previous chapter, we have already characterized the real-world datasets and we have suggested a classification-based approach for selecting attacks and identifying interesting test cases. In the following subsections, we explain how we can generate evaluation datasets and specifically attack datasets, i.e., the one representing activities carried out by attackers.

5.3.1. An overview of Attack Scenarios

Let us imagine how an attacker carries out an attack. Suppose that (for some reason), an attacker has decided to target a certain computer system. Our interest is beyond his intentions and motivations (e.g., to gain money, fame, revenge, vandalism, terror, etc.). At this point, the attacker has more or less information about the victim machine. If he has no information or insufficient information, he inevitably searches for additional information, to increase his chance of success. Various information-gathering methods can be used starting from social engineering and search engine harvest, which cannot be observed by the IDS, to network and vulnerability scanning tools, which contrarily can be seen by the IDS.

From the perspective of IDSes, the number of attack steps that appear in one session of an attack process is relatively arbitrary because attackers may proceed slowly during several days or weeks to avoid detection. Therefore, when they resume their attack later for subsequent steps, it appears to the IDS as a new attack (e.g., if the attacker begins directly to perform intrusive actions, e.g., privilege escalation, without going through the previous steps). Moreover, the attacker could be an insider who has already sufficient information and/or has a valid account to begin; he thus does not need to pass through the reconnaissance step. Inversely, an attack process could be stopped deliberately because the attacker has decided to abandon it definitively, for example, because it is too difficult to succeed or he has found a more interesting target. In all these cases, the effect for the IDS is the same: only a part of the attack scenario is observable for the IDS.

Attackers are generally equipped with an arsenal of “weapons” of different types, purposes and with different munitions that can be used during the attack cycle. The richness of this arsenal depends on attackers’ experience and skills. A particular weapon is fired in each attack step according to the current situation. Attackers often cannot acquire a comprehensive and complete knowledge of all vulnerabilities and all possible ways to exploit them because either their competence is insufficient, or because such information is unavailable for the attacker, because security mechanisms prevent the information leakage and/or the attacker’s privileges are too low. Similarly, they generally have access to only a limited number of attack tools.

The notions of “partial knowledge” and “finite number” of attack tools are crucial for our approach. This is why we suggest using these notions in conjunction with the attacker competence model and the attack-process model to draw the details of attack scenarios while reducing the effect of combinatory explosion. Hence, we can determine what to execute, which sequence of execution, when to execute and against which victim machine. In fact, we can summarize the underlying assumptions for this work as follows:

1. An attack process does not always follow a complete cycle (i.e., reconnaissance, access gain, privilege escalation, etc.). Instead, it can be abandoned, suspended or even blocked.
2. An attacker has a partial knowledge about the targeted system. The amount of knowledge that can be acquired varies from one attacker to another and from one targeted machine or network to another.

3. Attackers have different skills and experience levels. They may use tools that are publicly available or tools distributed only within a closed community, or they may develop their own tools. However, the number of tools available to an attacker is generally limited.
4. The kind of tools as well as the sequence of using them depend on the main objective of the attack, attackers experience as well as the knowledge acquired about the targeted system.

5.3.2. Scenario Computing

Attack scenarios can be viewed as plans carried out by attackers. Moreover, because the resources available for the evaluation are limited, it is necessary to schedule the execution of different attacks on the available machines. This, in turn, is similar to the traditional scheduling problem. Fortunately, both the planning and the scheduling problems can be solved easily by constraint-programming languages. Once an appropriate model of the problem is created, and the constraints are defined, built-in search algorithms can be applied to find valid solutions.

Process planning is a typical problem in artificial intelligence and operational research. It aims to find suitable plans to perform a series of intermediate tasks to achieve a final goal while satisfying predefined constraints. It looks for valid solutions in a huge number of nodes in search tree or search space.

The art of constraint programming lies in finding an appropriate model for the problem and a distribution strategy that yield a computationally feasible search tree. In our context, we represent the problem of computing attack scenarios in finite domain as a path-finding problem. In order to do that, we transformed the attack process of Figure IV.8 into the graph shown in Figure V.5. It consists of eight nodes labeled as: R, GA, DoS, VB, CDI, EP, IMC, HT, which corresponds to: Reconnaissance, Gain Access, Denial of Service, Victim Browsing, Compromise of Data Integrity, Execute Program, Implant Malicious Code and Hide Trace respectively.

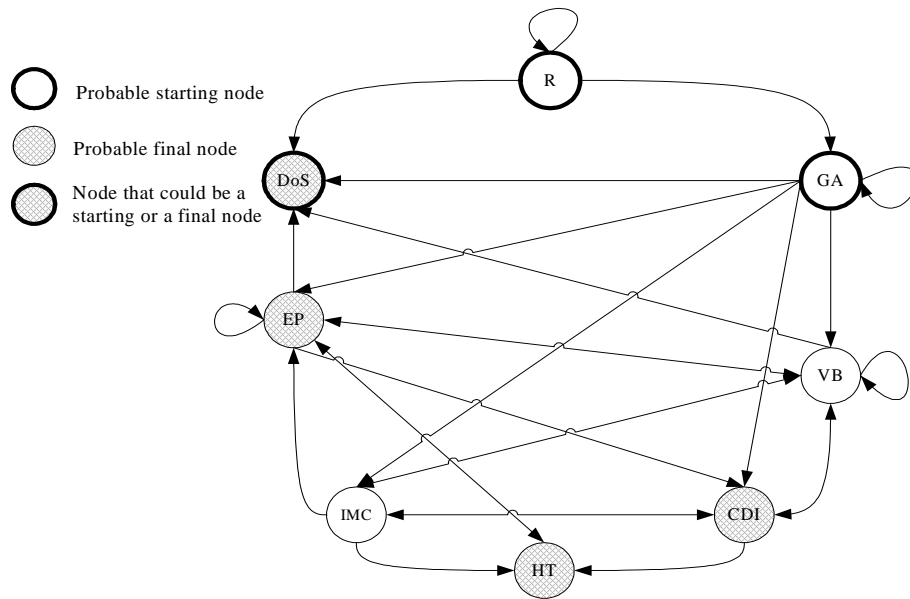


Figure V.5: Graph representing connections and possible paths between attack steps (i.e., attack cycle).

Marking nodes as start or final as well as the connections between different nodes have been deduced from our observations on attack scenarios embedded in malware (cf. Chapter 4) and honeypot data (cf. the previous section), which also coincide with common sense. For example, an attacker cannot begin browsing a victim host without gaining first access to that host. As another example, the node DoS can be a start node as well as a final node because some types of denial of service attacks do not require any access to the victim machine. The DoS node is also a sink node since it often results in crashing the system or making it unavailable, and thus further attack steps on this victim become impossible.

In fact, this representation is so flexible that the underlying attack process model can be radically changed: modifications in the attack process model can be applied easily by modifying the connection matrix and the constraints. For example, we can add a new abstract action by adding an entry for the new

node, defining its connection with other nodes and stating the assumed constraints (e.g. whether it can be a start or end step in the scenario, whether it can have self loops, etc.). The desirable solution corresponds to all possible paths (i.e., scenarios) that satisfy predefined constraints. For example, we assume that scenarios must satisfy the following constraints:

- Scenarios can begin only from nodes: R, GA, or DoS.
- Scenarios terminate only in one of the following nodes: DoS, CDI, HT or EP.
- Self-loops are allowed only at nodes R, GA, and VB.
- A node should not be traversed more than “MaxLoop” times (neither as self-loop nor as re-pass) in the same scenario. The MaxLoop parameter is chosen explicitly by the evaluator.
- Nodes are connected as shown by Figure V.5 or by the connection matrix of Table V.2.

Table V.2: Connection matrix.

\ From:	R	GA	DoS	VB	CDI	EP	IMC	HT
To:								
R	1	1	1	0	0	0	0	0
GA	0	1	1	1	1	1	1	0
DoS	0	0	0	0	0	0	0	0
VB	0	0	1	1	1	1	1	0
CDI	0	0	0	1	1	1	1	1
EP	0	0	1	1	1	0	1	1
IMC	0	0	0	1	1	1	0	1
HT	0	0	0	0	0	1	0	0

In essence, to obtain a detailed plan of attacks or executable scenarios, the following five questions should be answered:

- What sequence of execution?
- What to execute?
- Executed on which machine? And against which victim?
- When to execute?

Since the answers to these questions require searching in a huge (potentially infinite) space of possible solutions, we answer the above questions in an iterative manner. First, we generate a finite number of possible scenarios (what sequence of execution?) at a high level, according to the graph of Figure V.5. The produced abstract scenarios are in the form:

$$\text{Scenario} = [S_1, S_2, S_3, \dots, S_n], \text{ where: } S_1 \in \{R, GA, DoS\} \text{ and } S_n \in \{DoS, CDI, EP, HT\}$$

For example, a four-step scenario could be (R, GA, VB, EP).

In addition to that, constraints related to the attacker's profile can be applied at the same level to refine the generated scenarios. For example, if we are interested only in attacks performed by newbies, we add the constraint $HT \notin S_n$. Therefore, the possible scenario space will be further reduced by excluding all scenarios that contain a hide trace step.

The last task in generating attack scenario dataset is to translate abstract scenarios into concrete, executable scenarios, as explained in the next subsection.

5.3.3. Transformation into Concrete Scenarios

To answer the "what to execute?" question, abstract scenarios are taken and instantiated from a database of tools/commands according to their corresponding steps as well as their statistical occurrences extracted from honeypot data (statistical parameterization model). In the command database, an arsenal of tools/commands is categorized by our classification scheme and mapped to a particular attack step that it realizes. For example, tools such as *nmap*, *nessus* and *ping* are mapped to the reconnaissance step (R); commands like *ls*, *pwd* and *uname* are mapped to victim browsing (VB); *ssh* and *telnet* are mapped to Gain Access (GA), and so on.

Accordingly, we have categorized many commands related to Unix/Linux environment, as well as many attack tools. Table V.3 presents some examples of the categorized commands where each entry represents an elementary attack action, which is characterized by both its appearance (i.e., a classification perspective) as well as its attack step (i.e., a scenario perspective). Moreover, commands are also associated to level of competence. For instance, executing the command *nmap* with the option *-sF* can be associated to skillful profile.

Launching attack on which machine and against which victims machine of the evaluation platform (the answer to the “against which victim?” question) can be introduced explicitly by the evaluator either as a pool of single IP addresses or sub-networks and by determining which OS by example.

Finally, the answer to the “When to execute?” question has two factors: one is related to attacker profiles, e.g., whether they apply a stealthy technique, and the other one depends on the resources available for the evaluation. The first timing concern is estimated in rough time units according to the attacker profile, while the later is determined in time ordering for the scheduled execution on the evaluation platform. Both “*time ordering*” and “*machine attribution*” questions (i.e., executed on which machine) are answered by the scheduler.

We present, in the following section, the evaluation kit that we have developed to implement the ideas discussed in this dissertation.

Table V.3: Examples of attack tool characterization.

Attack Tool	Source	Privilege Acquired	Vulnerability	Network Carrier	Native-Carrier	Target	Step
ping	Remote	Configuration	None	Transport layer	None	Network stack	Reconnaissance
nmap	Remote	None	None	Transport/App. layer	None	Network stack	Reconnaissance
nessus	Remote	None	None	Application layer	None	Network stack	Reconnaissance
rlogin	Remote	variable	None	Application layer	None	Service	GA
telnet	Remote	variable	None	Application layer	None	Service	GA
ftp	Remote	variable	None	Application layer	None	Service	GA
ssh	Remote	variable	None	Application layer	None	Service	GA
get/put (FTP)	Local	None	None	None	Command Execute	File system	IMC
cron	Local	None	None	None	Command Execute	Application	EP
Su -u root	Local	Root	None	None	Command Execute	OS	GA
ls	Local	None	None	None	Command Execute	File system	VB
ps	Local	None	None	None	Command Execute	Process/Application	VB
grep	Local	None	None	None	Command Execute	File system	VB
Vi	Local	None	None	None	Command Execute	File system	CDI
rm	Local	None	None	None	Command Execute	File system	CDI/HT
touch	Local	None	None	None	Command Execute	File system	HT
kill syslogd	Local	None	None	None	Command Execute	Service	HT
kill httpd	Local	None	None	None	Command Execute	Service	DoS
Gcc malware src	Local	None	None	None	Command Execute	File system	IMC
Execute downloaded malware	Local	None	None	None	Command Execute	Application	EP
Ping-of death	Remote	None	Configuration	Transport layer	None	Network stack	DoS
Metasploit/dcerpc/ms03_026_dcom	Remote	System	Implement/Design	Application layer	instruction	OS	GA
Metasploit/ftp/3cdaemon_ftp_user	Remote	variable	Implement/Design	Application layer	Execute Command	Service	GA
Dictionary brute force attack	Remote	Variable	Configuration	Application layer	None	Service	GA
unshadow (John-the-ripper)	Local	None	None	None	Command Execute	File system	VB

5.4. Development of the Evaluation Kit

In order to produce attack-scenarios automatically and to generate background network traffic, we have implemented a prototype evaluation kit that integrates the aforementioned ideas. The key components of the kit are the *Attack Injection Tool (AIT)*, the *Background Traffic Injector (BTI)* in addition to the *evaluation manager*. In this section, we describe each one of these components. Figure V.6 shows the overall structure of the evaluation kit.

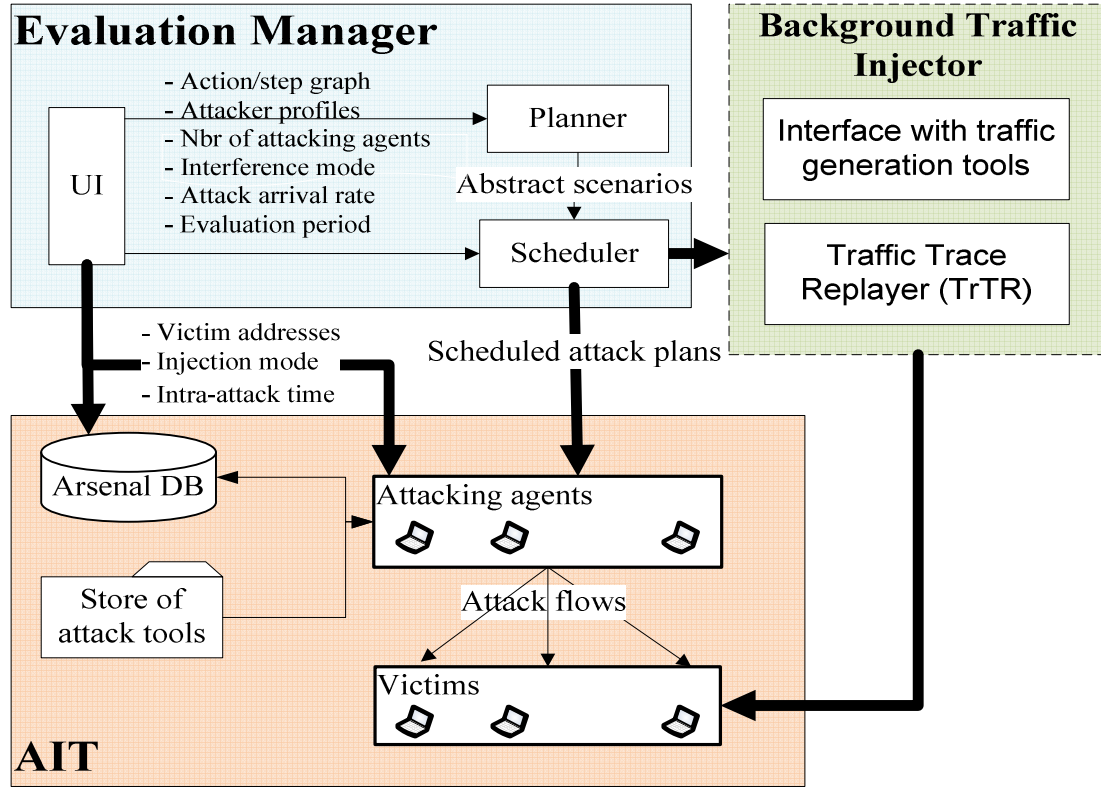


Figure V.6: An overview of the evaluation kit structure.

5.4.1. Evaluation Manager

The *evaluation manager*, as illustrated in Figure V.6, is composed of a *user interface (UI)*, the planner and the scheduler. The *UI* can be used by evaluators to control the experiment and to customize or configure the other components. The *planner* module produces abstract scenarios while the *scheduler* module generates scheduled executable scenarios and distributes them to *attack generators*. It also schedules replaying background traffic.

As we have discussed in Section 5.3.2 (Scenario Computing), attack scenarios can be viewed as a planning and scheduling problem, which can be easily solved by constraint programming languages. For this reason, we decided to develop the planner and the scheduler by using *Mozart* system, which is based on *Oz* language [Mozart08]. It provides the salient features of logic programming and constraint programming by providing logic variables, disjunctive constructs and programmable search strategies. Furthermore, *Oz* is a concurrent language where users can dynamically create a large number of sequential threads that can interact with each other.

The concurrent logic-programming paradigm of *Mozart* is an advantage as it facilitates building reactive programs where the generated scenarios can be interactive. Moreover, we can generate concurrent attacks or attack scenarios. In addition to that, it supports agent programming. The constraint programming and the logic programming beside the programmable search strategies allowed us to implement easily the planner and the scheduler of attacks. Thanks to all these features, the whole kit, including the planner, the scheduler as well as attacking agents can be implemented, in future versions, by the same language.

We preferred *Mozart* rather than other constraint solving toolkits such as *ILOG Solver* {Ilog08} or *JaCoP* {Jacop08} that integrates constraint programming into a host imperative programming language like *C++* or *Java*. The reason for this decision is the difficulty of controlling dynamically the addition of constraints, as in our context, the constraints may change quite often. Moreover, debugging and optimization in this type of toolkits is done at the level of the host language rather than at the level of the constraints programming, which leads to easier development.

A) Planner

Scenario planning is transformed into a constraint satisfaction problem (CSP). After the definition of variables and constraints over these variables, a constraint solver (e.g., *SearchAll*, or *ExploreAll*) is applied to find solutions. The constraint solver expects a CSP in the form of a script. A script is a procedure whose only argument is the solution of the solved CSP. The variable is often called the root variable of the script (here an anonymous procedure with the root variable called *Scenario*).

The *Data* structure that represents the step nodes and their connections is shown in Figure V.7. A new node called “end” is added to model the finishing nodes.

Data = [r	# [ga dos end]
	ga	# [dos ep imc cdi vb]
	dos	# [end]
	vb	# [cdi imc dos ep]
	cdi	# [ep imc ht vb end]
	ep	# [imc ht vb dos cdi end]
	imc	# [vb ht cdi ep]
	ht	# [ep end]
	end	# nil]

Figure V.7: Data structure that stores the information of the graph described in Figure V.5.

A portion of the code that implements the planner module is illustrated in Figure V.8. The anonymous procedure *proc {\$ Scenario}* is wrapped by the function *AttackScenario* that receives the above *data* structure. This script produces all the possible scenarios with different lengths that correspond to all possible paths of Figure V.5. It begins by mapping the input *Data* structure into two records one for the steps and one for their connections. It then maps the steps symbols into integers 1 to 8 and the END step to 9 because we use the finite domain solver (FD). The maximum length is calculated in terms of self-loops and the repetition of same attack step (if allowed).

Table V.4: Information on the size of search space.

	# of choice points	# of failed points	# of success points (Solutions)	Tree depths
MaxLoop = 1	2989	2703	287	32
MaxLoop = 2	196650	179775	16886	67

Note that in Oz scripts, we should define explicitly a search strategy (or, more specifically the distribution strategy). The procedure *FD.distribute* expects a specification of a distribution strategy and a record or list of the constrained variables. The distribution strategy specifies in which order variables are visited during the search process. The specification ‘ff’ stands for first-fail and means that always the variable with the smallest domain is visited next. The selection of a suitable distribution strategy is vital for the performance of the search process. Table V.4 demonstrates how explosive the number of possible combinations in the search tree is when we equate MaxLoop to 2. For this reason and to reduce the severity of the combinatorial explosion problem we decided to treat the repetition of steps at the time of scenario-execution according to the attacker’s profile.

```

declare
fun {AttackScenario Data MaxLoop}      %Beginning of the wrapper function that calculates the scenarios
  StepsSymb = {List.mapInd Data fun {$ I S#_} I#S end}      %Extract the steps symbols from Data
  StepsRec = {List.toRecord sr StepsSymb}
  Steps = {Record.arity StepsRec}                                %transform step symbols into integers
  Connections = {List.mapInd Data fun {$ I _#Cs} I#Cs end}      %Extract connectivity
relationships from Data
  ConRec = {List.toRecord cr Connections}
  NbSteps = {Length Steps}
  MaxLength = MaxLoop + NbSteps                                %Determine the MaxLength of scenarios
  %%This Procedure tests whether element X belongs to list Ys
  proc {BelongsTo X Ys ?B}
    if {List.member X Ys} then B = 1
    else B = 0
    end
  end
in
  %The main procedure that calculates scenarios
  proc {$ Scenario}
    ScenarioLength = {FD.int 2#MaxLength}      %ScenarioLength is declared to an integer from 2 to
MaxLength
    in
      {FD.distribute ff [ScenarioLength]}      %Find all possible values of ScenarioLength
    %%Scenario: Step ---> Step_number
    %Create the root variable Scenario in which solutions will be stored
    {FD.tuple scenario ScenarioLength 1#NbSteps Scenario}
    %*****
    %          %Scenario Constraints
    %*****
    %% Constraint: Starting Steps should be R, GA or DoS
    Scenario.1 =:<: 3
    {For 2 ScenarioLength 1
    proc {$ I}
      J K L
    in
      thread
        J = Scenario.I
        K = I-1
        L = Scenario.K
        %% Constraint: ensures that transitions occur only between adjacent steps
        {FD.impl I=<: ScenarioLength {BelongsTo StepsRec.J ConRec.L} 1}
      end
      %%Constraint: Last step should always be "end"
      {FD.impl I=:ScenarioLength Scenario.I=:NbSteps 1}
    end}
    %% Constraint: At most times a step could repeated
    {For 1 NbSteps 1
    proc {$ I}
      {FD.atMost MaxLoop Scenario I}
    end}
    {FD.distribute ff Scenario}      %Search strategy
  end
end

```

Figure V.8: An excerpt from the scenario calculation script written in Mozart language.

B) Scheduler

The scheduler takes the abstract scenarios produced by the planner module in addition to the following parameters: the number of attacking agents, the interference mode of attack sessions (i.e., several attack sessions in parallel or just one session per time slot), the attack arrival rate and the injection period. It assigns attack sessions to attacking agents and produces a schedule for attack execution. For simplicity, the scenario assigned to an agent is completely carried out by the agent, even in case of a coordinated or distributed attack scenario. Agents are able to launch attacks originating from different IP addresses. The scheduler is also coded in Mozart but its code size is quite large to be included here.

5.4.2. Attack Injection Tool (AIT)

AIT is the core component that, beside the tool database and repository, implements the models that we have presented earlier. An effective attack generation tool should eliminate the biasing effect of the limitations inherent in existing attack generation approaches (see Section 4.2.2). In other words, it should satisfy the following requirements:

- *Flexible*: It should enable evaluators to customize the evaluation dataset to match their own specific environment. This implies the possibility to vary sequences of attack scenarios, to generate different types of attacks as well as different variations of the same attack instances.
- *Automated*: to reduce the effort and time required for the evaluation. This allows evaluators to spend more of their time on improving the experiment design and the quality of the dataset, rather than loosing time on the dirty work of carrying out the experiments required for the evaluation.
- *Extensible*: Because of the rapid evolution of both software releasing, vulnerability discovery as well as attack development, if the tool or the dataset was neither extensible nor flexible (i.e., to include new attacks and add new environment features), it will be out-of-date very soon.
- *Reproducible*: It is important to repeat experiments under the same conditions to allow the confirmation of results and as well as the comparison between different products.

We have developed the AIT in a manner that satisfies the requirements cited above. It is flexible since evaluators can select particular attack test cases; adjust scenarios by selecting attacker's level of experience, the address pool of victims and the scope of targeted vulnerabilities (e.g. targeting only *Windows XP* victims). Moreover, it can be extended easily where new attack tools and scripts can be added easily to the tools arsenal and by updating the arsenal database. It has also a possibility to record attack sessions either as abstract scenarios or as network traffic. Thus, evaluation experiments can be reproduced.

In addition to that, attack scenarios can be generated automatically without any intervention from evaluators. It can be launched in two modes: (1) an exhaustive mode where all the attack tools available in the arsenal are considered or (2) a customized mode, if evaluators are interested only in attacks against particular types. Moreover, attack timing can be adjusted to determine attack arrival times, intra-attack delays and whether attack sessions interference is allowed.

A) Architecture

The main components of AIT that correspond to key entities involved in the attack process are shown in Figure V.9. These entities interact between them as well as they can interact with the tool *arsenal database* and the *tool store* where attack tools are installed or stored. Sometimes an entity by itself is directly implementing one of the underlying models and sometimes the model implementation is distributed over several classes. For example, the *attacker* class implements attacker competence model whereas the attack process model is implemented by the *AttackScenario* and the *AttackStep* classes.

Indeed, the most important classes correspond to attacker, victim machine, attack tool, attack scenario, attack session and vulnerability, which can respectively be represented as follows:

```
Attacker = (attacker_id, level, ip_address, {tools})
Victim = (victim_id, platform, ip_address, {vulnerabilities})
AttackTool = (level, commands, options, step, testcase_id, requirement, exploited_vulnerability)
AttackScenario = (scenario_id, {steps})
AttackSession = (session_id, attacker_id, victim_id, scenario_id, start_time, end_time, current_step)
Vulnerability = (vul_id, platform, software, date)
```

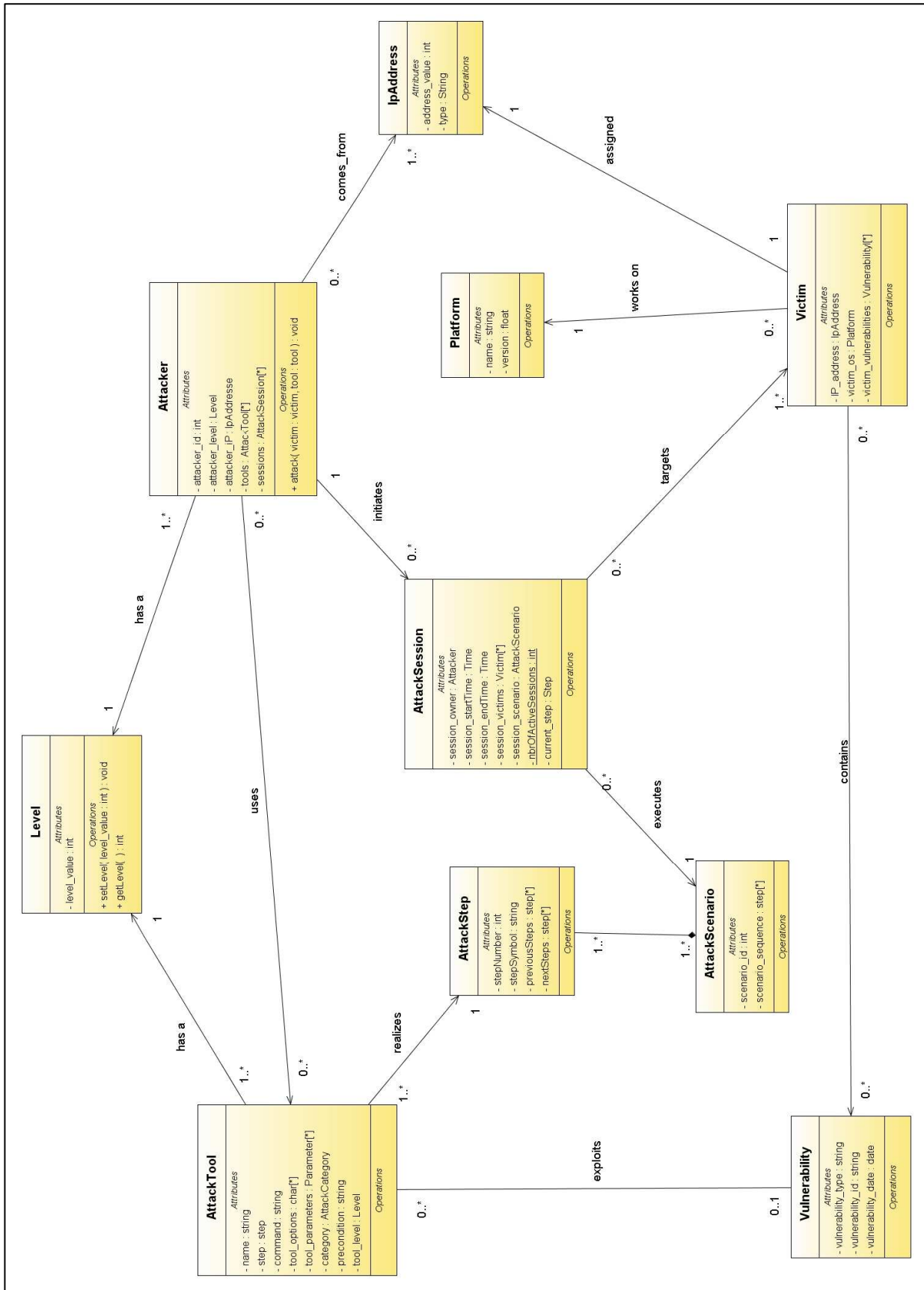


Figure V.9: Main components of AIT that correspond to main entities involved in attack scenarios generation.

B) Implementation

We have implemented AIT in *Ruby* {Ruby08}, the programming language that was used in developing *Metasploit*. The decision to use *Ruby* is evident in order to benefit from its *REX* library that provides a wide range of classes and methods to manage exploitation. Moreover, AIT can be simply used as a plug-in of *metasploit*. To achieve that, we have applied some modifications in *metasploit* source codes extending its functionality to support executing automatic scenarios. The modified components of *metasploit* as well as a portion of important *REX* libraries that we have used are shown in Figure V.10.

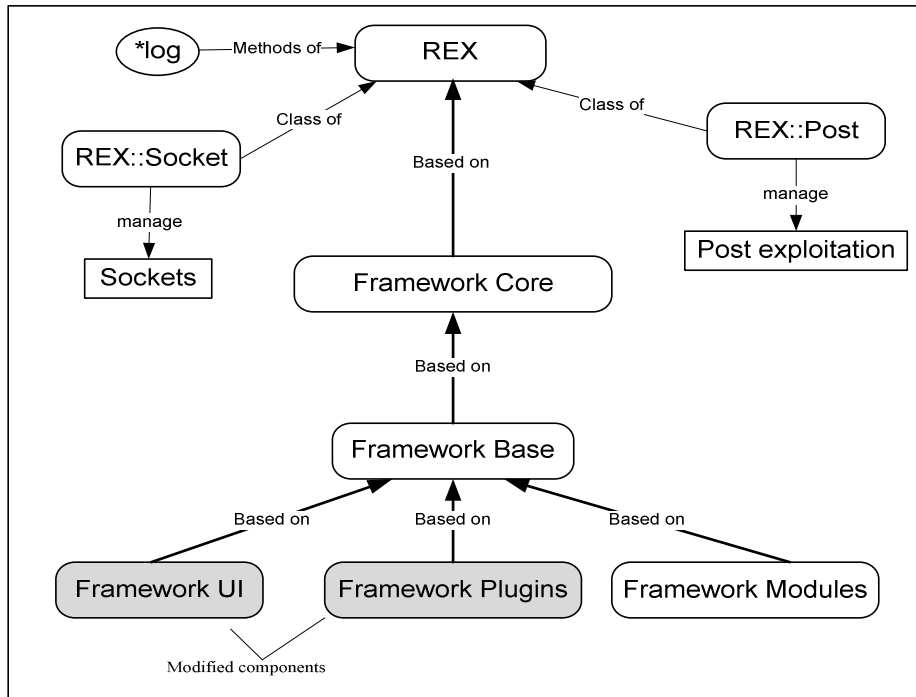


Figure V.10: Modified components and classes used of metasploit/REX.

5.4.3. Traffic Trace Replayer (TrTR)

IDS can be evaluated without any background activities at all. However, background dataset is essential for two reasons: (1) to measure the capability of the IDS to distinguish normal activities from attacks and to ensure that the IDS does not generate too many false alarms, (2) to measure the performance of the IDS when it falls under heavy loads (i.e., stress test).

We have two main options to generate background datasets for IDS evaluations: either using a synthetic dataset or using real datasets. DARPA datasets are an example of synthetic background dataset that is artificially generated. Several tests still use synthetic traffic and new papers proposing better approaches to generate fake traffic are published every year {Antonatos04}. This could be a great solution if the background generation methods effectively produce reliable datasets that are closely similar to the real datasets. However, these methods, even the more sophisticated ones, failed to produce such realistic datasets.

Real datasets are usually captured from real operational networks or hosts. They are then replayed on the evaluation platform. Because the dataset in this case is a real trace, they keep nearly all the characteristics of the environment from where it originates. However, when such dataset is used for evaluations carried out in different environments, it might be no longer representative of the real activities of the new environment.

More importantly, the real traces contain data that contains itself sensitive information or indirect information that can be inferred from the trace (e.g., inferring the network topology from IP addresses in the network traffic). The use of real traces has several organizational and legal restrictions to protect confidential data and the privacy of users. This may interpret why no such real traces are publicly available. A sanitization process of the real traces to remove or hide the sensitive information can be a good solution. However, sanitization should be done carefully to avoid destroying the original environment characteristics.

Another big problem of using real datasets is the difficulty to ensure that it is clean (i.e., it does not contain attacks). In fact, since the main hypothesis regarding the background dataset is that it is “normal” and contains no attack, the presence of malicious events can distort the results of the evaluation. For example, a clean background dataset is often used to train anomaly-based algorithms. An attack in the training dataset would be learned as “normal behavior”, making the intrusion detection system ineffective against that type of attacks.

Those who are working on network simulation or network emulation know how it is difficult to generate network traffic that closely reflects the characteristics of real networks {Paxson97-a}. Analyzing network security or testing network-based security mechanisms is more difficult because they are sensitive to both the context, the traffic content as well as low-level characteristics (volume, packet sizes, etc.), while in network performance analysis and evaluation, other parameters are more significant, such as delays, congestion control, packet loss, etc., on which the content and context have minor effects.

Using real traces and replaying them on the test bed seems to be a candidate solution. Although this solution has the advantage to keep the characteristics of the original traffic, it also has some limitations. First, it is difficult to collect a sufficient number of traces or a sufficient diversity of traces to test security mechanisms against various network profiles that correspond to different environments. Therefore, we need to merge several traces that, unless merged carefully, could present inconsistent characteristics. Second, we need to inject attack traces into the background traffic. The injected attack may be easily detectable if they are artificially inserted without taking into account the characteristics of the background traffic.

To overcome these weaknesses while evaluating network-based IDS (e.g., *Snort* {Roesch99}), we naturally have surveyed the literature for any tool that allows the manipulation of traffic traces. We found tools that allow the fabrication of customized individual packets from scratch such as *Packit* {Packit08}. The closest tool to our needs is *TcpRewrite*, which is a part of the *Tcpreplay* suite {Tcprewrite08}. It allows several modifications at layers 2-4 either trace-wide or at the session-level.

Unfortunately, almost none of the tools allow fine-grained modifications (e.g., session-level, or payload contents) as well as trace-wide modifications (e.g., replace an IP address by another in the whole trace) while providing, at the same time, packet insertion functions. More importantly, in our context, we need a tool that keeps all the security-relevant characteristics such as timing issues and addressing issues. Therefore, the main goal of this work is to fill the gap by providing a security-guided manipulation.

In the previous chapter (Section 4.8), we have identified the most security-relevant characteristics of network traffic. In this section, we describe a traffic manipulation and replaying tool that we have developed to be used in the context of IDS evaluation. We first outline the basic requirements that we want the tool to fulfill, and then we describe briefly its architecture and its implementation.

A) Requirements

In our context, the expected tool should provide the following functions:

1) Trace analysis functions:

- Analyze IP traffic and sort it according to source/destination.
- Report on sessions found in the trace by size, duration, address source and destination, port, etc.
- Report the trace duration.
- Report the internal and the external addresses.
- Distinguish between servers and clients IP addresses.

2) Trace manipulation functions:

- Apply user defined modifications, consistently with the characteristics of the rest of the trace, at the packet, the session and the whole trace levels. Possible modifications include:
 - MAC Address
 - IP source and destination addresses
 - Source and destination ports
 - Time to Live (TTL)
 - Time stamps
- Produce a larger trace from several existing traces by merging and harmonizing the characteristics of traces in a consistent manner while keeping the temporal characteristics.
- Expand an existing trace by replicating sessions while keeping the original temporal characteristics as much as possible. Replicated sessions can be modified according to user parameters or simply repeated without modification except the time stamps.

- Inject attack packets and attack sessions created by the user consistently into an existing trace.
- Recalculate the checksum field whenever the packet is modified.

B) Design and architecture

Given that the existing tools do not correspond to our expectations where we need to combine the analysis, the modification and the insertion of packets in the same tool, we had to write our own.

We kept the design of the tool as simple as possible while maximizing the use of classes and methods provided by the *Jpcap* [Jpcap08]. As shown in Figure V.11, the tool consists of 3 modules: the *Preprocessor*, the *Analyzer* and the *Modifier*. The *Preprocessor* module, reads the trace, extracts the basic information (i.e, IP addresses, time stamps, and other header fields) in an indexed data structure (to be easily accessible later). From this data structure, the *Analyzer* computes measurements, such as session duration, traces duration, and extracts information such as IP addresses, MAC addresses and other useful information from the data structure and displays them in an organized manner. The *Modifier* receives the new parameters that the user wants to modify or the storage path of another trace or another packet she/he wants to inject. The modifier module edits the trace by applying the user-supplied parameters or injects the new attack trace into the original trace and produces a new trace.

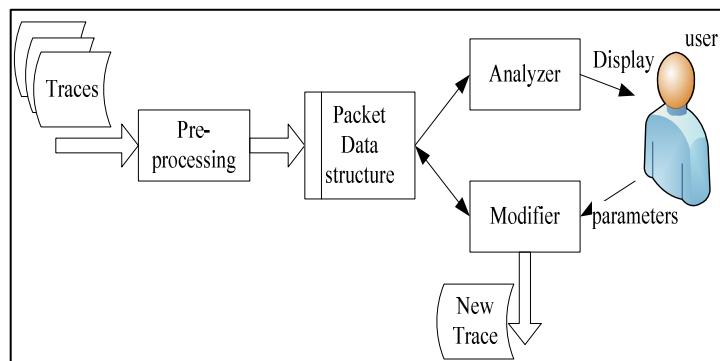


Figure V.11: Architecture of TrTR.

The class diagram shown in Figure V.12 presents the main classes that we have used. We decided to use *Java* for the implementation of *TrTR* for portability reason, to be OS and platform independent. We used the *Jpcap* library, which is the equivalent of *LibPcap* [Tcpdump08], the main *C* library for traffic capturing and manipulation. Note that although *Jpcap* is not as powerful as *LibPcap*, we managed to satisfy nearly all the requirements presented above.

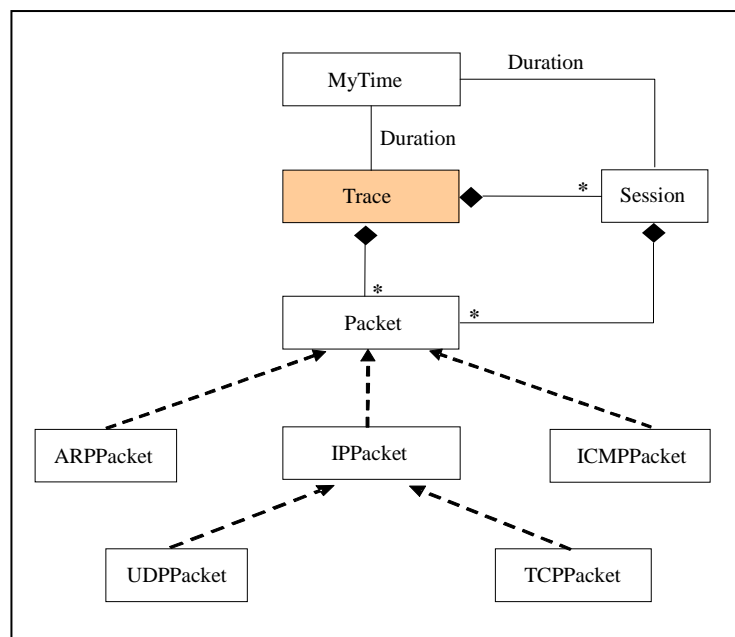


Figure V.12: A simplified class diagram for classes used in TrTR.

Basically, in our implementation, a trace (T) is a file containing a set of packets (P) = $\{p1, p2, p3, \dots, pm\}$, which were captured from a network interface by using traffic capturing tools such as *TCPdump* {*Tcpdump08*} or *Jpcap* itself.

The packet format is already defined in the library *Jpcap*, thus we can directly obtain not only the header's fields as well as the payload but also the time stamp. The later corresponds to the moment of capturing the packet. The time stamp $t(pi)$ is an essential information as far as it determines in which order the packets were sent and this allows replaying trace on a network interface while respecting the delays between the consecutive packets.

The library *Jpcap* also defines several types of packets organized by their layer protocol. In particular, we can access all the fields dedicated to the protocols IP, ICMP and ARP (at the network level) as well as the header fields of TCP and UDP packets (at the level of the transport layer). Therefore, according to the type of a packet we can consult and modify the addresses, the ports, the TTL (Time To Live), etc.

C) Implementation

The library *Jpcap* supplies adequate methods to open a trace file in the format *pcap*. We read the totality of the file and register all the packets in a vector structure, which allows us to access easily any packet in the trace.

1) Trace Analysis

We can determine easily the number of packets of the trace and calculate the duration of this trace by comparing the *time stamps* of the first and the last packets. We can also extract the IP addresses of all the packets.

By comparing the IP addresses with the non routable addresses and by analyzing the requests ARP packets we can determine the internal and external IP addresses.

The extraction of the sessions is made by filtering all the TCP packets and tracking down the connection establishment and tear down procedures. We can then associate a TCP packet to its session by using the *sequence number*.

For each session, we can list the addresses and the ports for sources and destinations. We also record all the packets belonging to the same session. Thereby, we can find the number of packets and calculate the session duration from the *time stamps* of the first and the last packets of the session.

2) Trace Modification

We implemented three different levels of modification. The first one (a) offers global modifications by processing (manipulating) the totality of a trace. The second level (b) allows working on some existing sessions, thereby fine grain modifications. Finally, the third level of modifications (c) corresponds to the insertion of new elements (packets or sessions) in a trace.

Global modifications: It is possible to replace single IP addresses of all the packets in the entire trace with different addresses. We can also change the TTL of all the packets by supplying a new value. Further, we can modify addresses in the trace based on ranges of IP addresses instead of single addresses.

Session-level modifications: The analysis phase creates a list of all the sessions of a trace. According to the session number, it is possible to extract the corresponding session object from the data structure. We can then apply transformations that concern only the packets of this particular session.

Packet insertion: To insert a packet into a trace, it is necessary to define its position in the trace and to shift all the subsequent packets. In addition, the time stamp of the packet that we insert has to be compatible with the chronological order of the precedent and subsequent packets in the trace.

It is thus preferable to insert a packet into an existing session. If the packet is inserted into a TCP session, it is necessary to be careful to the organization of packets within the session. Furthermore, we should initialize the header of the new inserted packet to match the addresses and the ports corresponding to this session. Besides that, the acknowledged sequence numbers of the session packets must be also updated. It is worth mentioning that inserting a packet in a way totally disconnected from the context of a trace could have negative side-effects on the detection methods of the evaluated IDS.

3) Saving trace modifications

The major problem of the library *Jpcap* is that it is not possible to save the modifications made on a trace: it does not allow rewriting directly on a *pcap* file. Consequently, to overcome this limitation, we made a work-around solution, which is presented in Figure V.13.

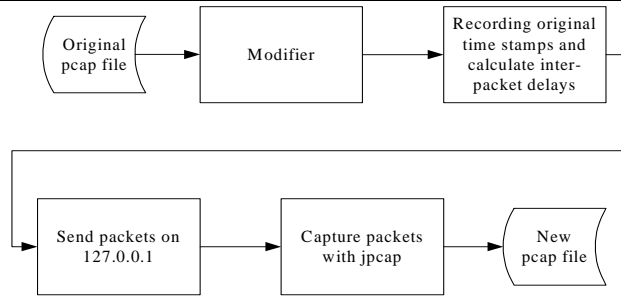


Figure V.13: The work-around solution for limitation in jpcap.

As the *Jpcap* allows capturing packets and saving them in *pcap* format, we have to replay all the packets of the trace on the network interface, listen to this local interface in order to capture packets and write them in a new *pcap* file. To ensure that traces that we have to replay are not mixed with other external packets, we send the packets on the loopback network interface (127.0.0.1), present on every machine.

To keep the temporal characteristics as close as possible to the original trace, a time-recording phase is completed before sending packets, which allows replaying packets with nearly the same inter-packet times. Since the period between the consecutive packets has been computed from their time stamps, it is easy to reproduce the temporal constraints of the traffic as present when the trace was recorded.

The advantage of this work-around solution is that all time stamps of the modified file will necessarily be correctly ordered and consistent. However, the time required for recording turns out to be the main inconvenience of this process given that it needs to replay the totality of a trace to produce the modified file.

5.5. Conclusion on Dataset Generation

Ready-made datasets such as DARPA datasets are relatively easy to replay for evaluating IDS. However, it is hard to maintain them because of the rapid evolution of existing products as well as the frequent arrival of new products. Besides, new attacks and variants of old attacks occur frequently. Therefore, these datasets become obsolete and outdated shortly after their appearance.

Attack dataset generators such as *Sploit* {Balzarotti06}, *Mucus* {Sommers04}, *Thor* {Marty02} and *Snot* {Snot07} represent a relatively better solution, as their updates allows the integration of recent attacks to design and implement different evaluation experiments. The main problem of these tools is that the characteristics of the synthesized datasets differ from real datasets. Moreover, they either focus on existing signatures such as *Snot* or focus on applying variation or mutation techniques on existing attacks such as *Sploit*, *Mucus* and *Thor*. Therefore, each tool covers only a small part of the attack space (i.e., attack types) and it is often difficult to combine several tools in a consistent manner. Moreover, they generate elementary attacks rather than attack scenarios.

Upon studying the related work in this area, we have realized that developing an attack “generator” tool can be more convenient and more flexible than gathering and replaying attack traces. Therefore, we proposed, in this chapter, a new approach to generate attack scenarios that alleviates the inherited limitations of ready-made datasets and traditional attack generation tools. Besides generating attack scenarios automatically, being flexible and extensible, it avoids the limitations of trace replaying approaches such as the storage cost and the rapid obsolescence.

We described, in this chapter our approach for generating attack dataset, which is mainly based on the models that we proposed in Chapter 4, and how the theoretical models were rendered “alive” in the Attack Injection Tool (AIT). Moreover, we presented another tool for replaying network traffic to serve as background dataset. We explained the architecture and the implementation of both tools.

Thanks to these tools, we argue that we can generate more realistic, more flexible and more representative attack scenarios than ready-made datasets or the synthetic datasets produced by traditional attack generators.

We intend by this tool to facilitate security testing and particularly to promote the performance of intrusion detection systems. By doing this, we hope to advance the state of knowledge and by no means intend to enable infringement. In the next chapter, we describe some experimentation performed by our proof-of-concept tool kit.

VI. Chapter 6: A Proof-of-Concept Evaluation

The philosophy behind the approach presented in the previous chapters can be better illustrated by an experimental evaluation. We believe that once evaluators have flexible tools for selecting attack test cases and generating evaluation datasets, they can set up their evaluations more easily and then spend more time on the design of experiments, the experimental evaluation itself and the analysis of the experiment results.

The main objective of this chapter is to report on the main features of our toolkit by making use of its main functions (i.e., selecting and generating representative attack test cases). This kind of exercise is very important, not only to figure out the implementation problems, but also to check if the tools can reach their expectations, i.e., in our case whether the characteristics of the generated datasets conform to the characteristics of real world workloads.

To explain how our approach can help in performing IDS evaluations more effectively and more efficiently, we opted to organize this chapter by following the steps of the evaluation methodology proposed in Chapter 3. We aim by this chapter to give a simple example¹⁸ of an IDS evaluation rather than a thorough one. Therefore, we present, hereafter, an illustrative evaluation of intrusion detection systems while focusing on how well the toolkit works.

6.1. Evaluation Goal / User Needs

In this simple experiment, the evaluation goal is to assess the detection capabilities of *snort* and *Bro*. We assume that the user or the client of the evaluation in this case is an administrator who intends to deploy a NIDS solution in his network and wants to benchmark the detection capabilities of *snort* and *Bro* to decide which one is more suitable for his own particular network.

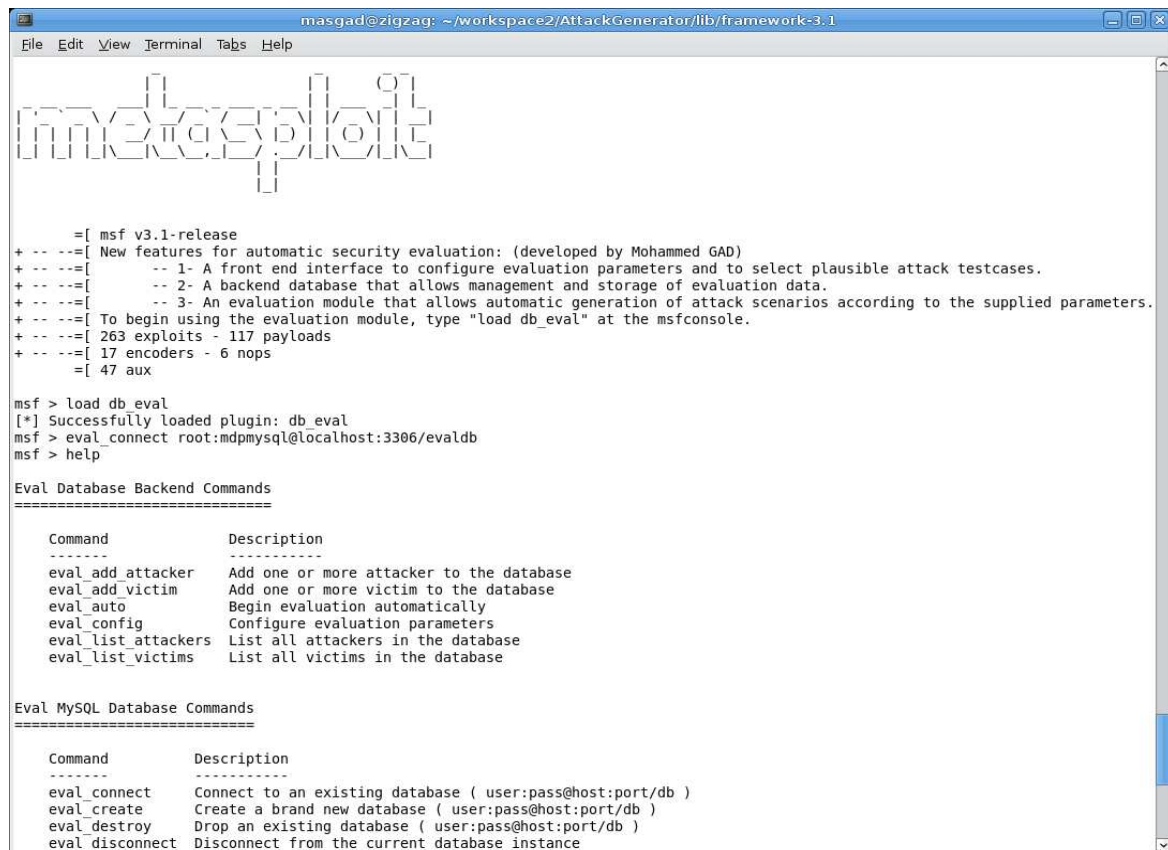
6.2. System and Workload Parameters

We can enumerate many parameters that can be tuned for the evaluation:

- Attack source: we can launch attacks from a remote source or from within the attacked machine. We concentrate here on remote attacks.
- Attack target: corresponds to the piece of software that is directly targeted by the attack.
- Attack carrier: this parameter indicates how the attack reaches its target or how it is invoked.
- Vulnerabilities: The vulnerability that is exploited by attacks. This parameter can have a global value as implementation or configuration vulnerability, or can take specific values corresponding to particular CVE/OSVDB vulnerabilities.
- IP addresses: this parameter concerns the IP addresses that can be assigned to both victim machines and attacking machines. It can either be arbitrary random addresses, or specified individual addresses or address ranges.
- OS platforms: obviously this can take a value amongst the well-known operating systems and their variations (e.g., Windows XP, Windows 2000, Sun Solaris, Mac OS, Free BSD, Linux, etc), as well as “universal”, which corresponds to cross-platform vulnerabilities. Here, we consider only windows and Linux variations.
- Attack level: this parameter reflects the attacker competence level and thus, the attack severity; it helps in defining some constraints when instantiating executable scenarios from abstract scenarios.

It may be noticed that several of these parameters correspond either to the classification attributes or issues from the other models (Chapters 4 & 5). Evaluators can define their values in the *evaluation manager* by invoking the command “*eval_config*” at the command prompt of *metasploit* framework (*MSF*). Figure VI.1 shows *metasploit* console after loading our tool plug-in and Figure VI.2 illustrates the configuration panel of our evaluation manager.

¹⁸ This evaluation is so simple that some steps of the evaluation methodology are deliberately shorten or even bypassed completely.



```

masgad@zigzag: ~/workspace2/AttackGenerator/lib/framework-3.1
File Edit View Terminal Tabs Help

  _____
 /_ _ _ _ _ \
|  _ _ _ _ |
| | _ _ _ _ |
| | _ _ _ _ |
| | _ _ _ _ |
|_| _ _ _ _|

= [ msf v3.1-release
+ -- -- [ New features for automatic security evaluation: (developed by Mohammed GAD)
+ -- -- [ -- 1- A front end interface to configure evaluation parameters and to select plausible attack testcases.
+ -- -- [ -- 2- A backend database that allows management and storage of evaluation data.
+ -- -- [ -- 3- An evaluation module that allows automatic generation of attack scenarios according to the supplied parameters.
+ -- -- [ To begin using the evaluation module, type "load db_eval" at the msfconsole.
+ -- -- [ 263 exploits - 117 payloads
+ -- -- [ 17 encoders - 6 nops
+ -- -- [ = [ 47 aux

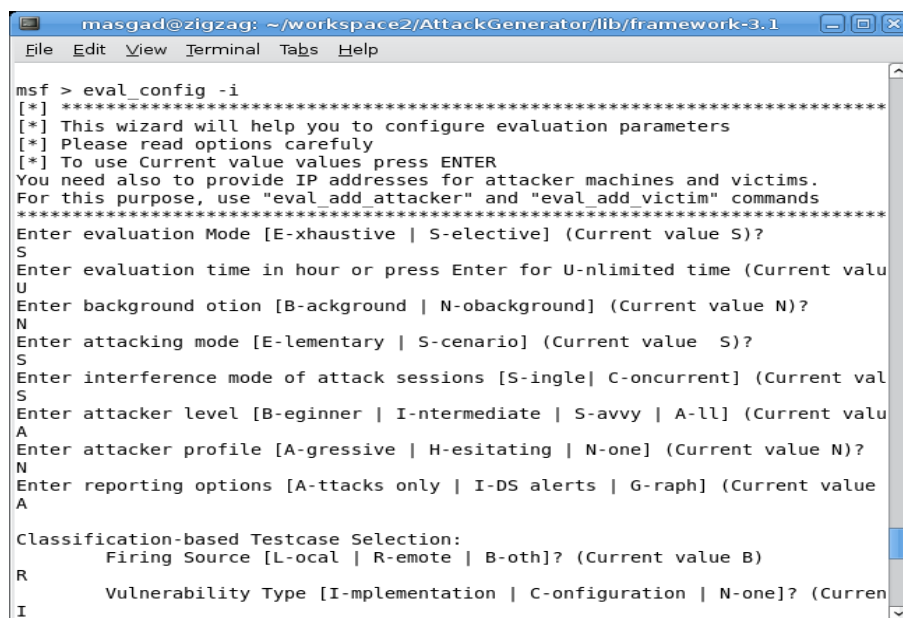
msf > load db_eval
[*] Successfully loaded plugin: db_eval
msf > eval_connect root:mdpmysql@localhost:3306/evaldb
msf > help

Eval Database Backend Commands
=====
Command      Description
-----
eval_add_attacker  Add one or more attacker to the database
eval_add_victim    Add one or more victim to the database
eval_auto          Begin evaluation automatically
eval_config        Configure evaluation parameters
eval_list_attackers List all attackers in the database
eval_list_victims  List all victims in the database

Eval MySQL Database Commands
=====
Command      Description
-----
eval_connect  Connect to an existing database ( user:pass@host:port/db )
eval_create   Create a brand new database ( user:pass@host:port/db )
eval_destroy  Drop an existing database ( user:pass@host:port/db )
eval_disconnect Disconnect from the current database instance

```

Figure VI.1: A screenshot of metasploit console after loading our plugin.



```

masgad@zigzag: ~/workspace2/AttackGenerator/lib/framework-3.1
File Edit View Terminal Tabs Help

msf > eval_config -i
[*] *****
[*] This wizard will help you to configure evaluation parameters
[*] Please read options carefully
[*] To use Current value values press ENTER
You need also to provide IP addresses for attacker machines and victims.
For this purpose, use "eval add attacker" and "eval add victim" commands
*****
Enter evaluation Mode [E-xhaustive | S-elective] (Current value S)?
S
Enter evaluation time in hour or press Enter for U-nlimited time (Current value U)?
U
Enter background option [B-ackground | N-obackground] (Current value N)?
N
Enter attacking mode [E-lementary | S-cenario] (Current value S)?
S
Enter interference mode of attack sessions [S-single | C-oncurrent] (Current value S)?
S
Enter attacker level [B-eginner | I-ntermediate | S-avvy | A-ll] (Current value A)?
A
Enter attacker profile [A-gressive | H-esitating | N-one] (Current value N)?
N
Enter reporting options [A-ttacks only | I-DS alerts | G-raph] (Current value A)?
A

Classification-based Testcase Selection:
  Firing Source [L-ocal | R-emote | B-oth]? (Current value B)
  R
  Vulnerability Type [I-mplementation | C-onfiguration | N-one]? (Current value I)
  I

```

Figure VI.2: The configuration panel of the evaluation manager.

6.3. Evaluation Technique

To perform such a benchmark evaluation, we need a test bed. We kept the design of the test bed as simple as possible to lower the cost and to simplify the management of its resources. We describe hereafter the main components that comprise the test bed.

6.3.1. Hardware Equipment

From the hardware viewpoint, our test bed consists of a network hub and three to five PC machines with moderate capabilities. They have no additional hardware option except a second network card that can be dedicated to control purpose. The machines are simply connected to the hub as shown in Figure VI.3. A first machine is used for managing the evaluation and generating attacks. The A second machine is used as a background generator. Victim machines can be installed as virtual machine guests on two virtual host machines. On a fifth machine we can install the evaluated IDS. The test bed can be minimally comprised of only three machines by using a single computer to host all victim virtual machines and by running the background generator on the same computer as the attack generator and evaluation manager, but this requires the use of more powerful machines.

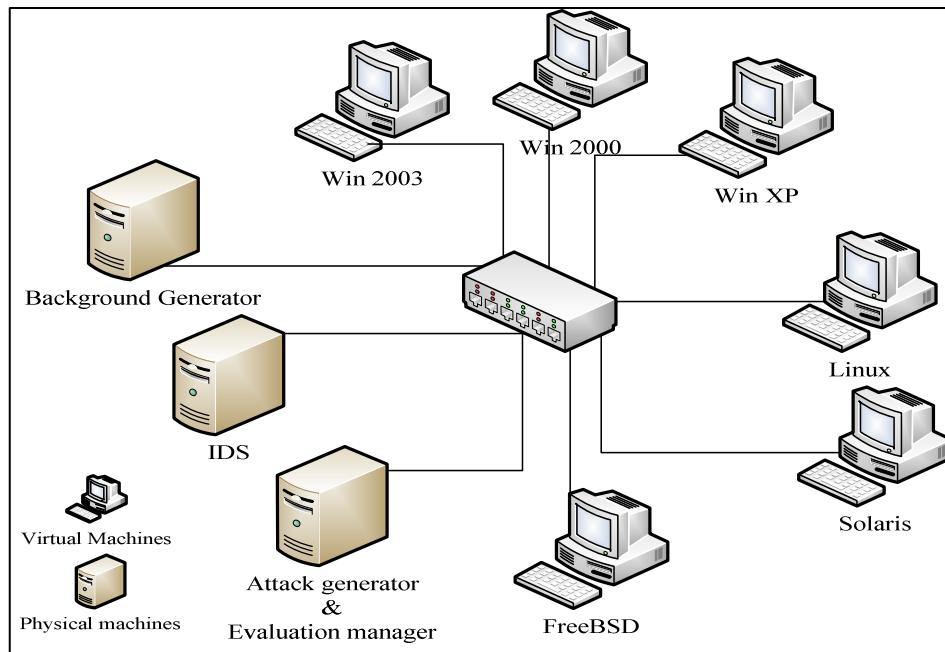


Figure VI.3: The evaluation test bed.

6.3.2. Software Equipment

- *Evaluation toolkit and attack tools:*

The main advantage of the design of our evaluation toolkit is that we can use existing attack tools in a natural way to generate attack scenarios (cf. previous chapter). Currently, we can use *metasploit*, *nmap*, as well as common-use utilities such as *ftp*, *ping*, *wget*, etc.

- 2. *Vulnerable software:*

In order to allow complete and successful scenario execution, we have ensured the existence of working vulnerable applications. This is not an easy task because most of vulnerable applications are either proprietary or ancient versions that are no longer available. Moreover, it would require downloading and installing a large number of vulnerable applications. For these reasons, we have only installed a few vulnerable applications, since this test bed is to be used just as a just a proof-of-concept of our approach.

In real life evaluations, this may be less difficult for administrators who want to evaluate IDSes on a test bed representative of their real environment because they often have installation CDs of software versions installed on their systems. Furthermore, the trend is now to intensively deploy servers and workstations as virtual machines. Therefore, administrators can clone working virtual machines and use their images directly in the evaluation platform.

- *Virtualization software:*

As stated above, victims are installed as virtual machines (VM). Thanks to the easiness of managing VM images (creating deploying, cloning, etc) and the possibility of managing them (e.g., starting, stopping and reverting to a previous snapshot), we can clean up the test bed after any compromise due to

attacks. We use for this purpose both *Sun's VirtualBox* {Vbox08}, which is a free and open source solution and *VMware server* {Vmware08}.

▪ *Evaluated IDSes:*

The last piece of software is the IDS target of evaluation. In this case, we have deployed *snort* version 2.8 and *Bro* 1.4 on a standalone real machine. It is worth noting that we make an out-of-the-box installations of *snort* 2.8 and *Bro* 1.4 without making any tuning to their signature rules.

6.4. Evaluation Metrics

Given the evaluation goals stated above, we use the detection related metrics presented in Chapter 3 (Section 3.5.7). More precisely, we compute the Overall Detection Ratio (*DR*) and the detection ratio per attack type (*DRPAT*). Let us recall that these two metrics are defined as follows:

$$DR = (\text{Number of detected attacks} / \text{Total number of attacks included in the dataset})$$

$$DRPAT = (\text{Number of Detected attacks of a particular type} / \text{total number of attacks of this type})$$

Both metrics are calculated in Table VI.1, based on baseline experimentations as explained bellow.

6.5. Design of Experiments for Selective Evaluation

As stated above, in addition to the main objective of the experimentation (to test the features that have been already implemented within the toolkit), we aim also to illustrate the applicability of our approach and the versatility of using such a toolkit. To achieve that, we carried out a set of experiments to test specific functionalities of our tool. In particular, we are interested in the following issues:

- Testing the evaluation management console and whether the generated attacks correspond to the desired ones (according to the entered parameters).
- Testing attack traffic generation and whether it reaches the victim machine.
- Testing attack success or failure (limitation: currently, this is done manually; it would be interesting to develop an automatic tool to verify attack success).
- Testing the victim part of the platform, e.g., if they are managed as they should be (started, stopped, restored, etc. at the scheduled time)

To stay consistent with the evaluation methodology, we have run two broad sets of experiments: the first set launches attacks selectively against the IDS without background traffic, whereas the second launches a set of selected attacks with background traffic to test the IDS false alarm issues.

6.5.1. Selecting Attack Test-cases

First of all, we verify the ability of *snort* to correctly detect the baseline attacks without background traffic; then, we calculate *DR* and *DRPAT*. Afterwards, we repeat the experiment against the same set of attacks but this time with a representative background traffic; then we calculate *FAR* and *FARPAT* in addition to *DR* and *DRPAT*.

Let us remind that our evaluation toolkit supports attack selection based on various criteria, including by OS, by vulnerability (by its CVE or OSVDB identifiers) or by date. In the following, we give some examples for attack selection.

Let us assume that the evaluator is more specifically interested in remote attacks that provide *system* access by exploiting implementation/design vulnerabilities and come over network traffic without triggering any native action. By using the command “*eval_config -i*” we can define the classification parameters shown the opposite box. The command “*eval_auto -l -t*” will display all the matched elementary attacks (corresponding to *metasploit* exploits). In our configuration, this command produces 19 attacks as shown in Figure VI.4

Firing source: Remote
Vulnerability:
 Implementation/Design
Privilege Escalation: System
Network Carrier: All
Local carrier: None
Target: All

Selecting attacks by “vulnerability” or by “operating system” is simpler. It will be sufficient to enter a particular *vulnerability id* or a particular OS name after executing “*eval_auto*” while using either “-v” or “-o”, respectively. Supposing that we are interested in attacks against *linux* and *bsd* platforms, executing the command “*eval_auto -e -o*” produces 12 attacks (see Figure VI.5).

```

Metasploit
File Edit View Terminal Tabs Help
msf > eval_auto -e -o
Enter operating systems in comma separated, then Press ENTER:
linux, BSD
Those are the attacks that will be launched (Total: 12 attacks):
linux/misc/ib_jrd8_create_database
linux/misc/ib_open_marker_file
linux/misc/ib_pwd_db_aliasd
linux/pptp/poptop_negative_read
linux/proxy/squid_ntlm_authenticate
linux/samba/lsa_transnames_heap
linux/games/ut2004_secure
linux/http/gpsd_format_string
linux/http/peerstream_url
linux/ids/snortbopre
linux/madwifi/madwifi_giwscan_cb
linux/misc/ib_inet_connect
Continue? [Y or N]

```

Figure VI.4: An example of attack selection by OS.

```

Metasploit
File Edit View Terminal Tabs Help
msf > eval_auto -e -t
Those are the attacks that will be launched (Total: 19 attacks):
windows/http/edirectory_imonitor
windows/iis/ms03_007_ntdll_webdav
windows/http/trendmicro_officescan
windows/smb/ms03_049_netapi
windows/smb/ms04_007_killbill
windows/smb/ms04_011_lsass
windows/smb/ms04_031_netdde
windows/smb/ms05_039_pnp
windows/smb/ms06_025_rasmans_reg
windows/smb/ms06_025_rras
windows/smb/ms06_040_netapi
windows/smb/ms06_066_nwapi
windows/smb/ms06_066_nwks
windows/smb/msdns_zonename
netware/smb/lsass_cifs
windows/ssl/ms04_011_pct
windows/mssql/ms02_056_hello
windows/wins/ms04_045_wins
linux/madwifi/madwifi_giwscan_cb
Continue? [Y or N]

```

Figure VI.5: An example of attack selection by category.

6.5.2. Baseline Experiments

During the experiment, we use the test cases corresponding to the attributes shown in the above section (Remote, Implementation/Design, System, All, None, All).

The results presented in Table VI.1 show that *snort* (with the out-of-box configuration), surprisingly, detects one attack out of the 19 test cases. On the other hand, *Bro* has correctly detected more than half of the attacks included in the dataset. This does not mean that *Bro* is better than *snort* but rather this may be due to the default configurations, with more modules or signatures activated by default in *Bro*. Further investigation is required to determine if one is really outperforming the other. Here, the IDS fault tree technique, which we have proposed in Section 3.7.2, comes into mind since it can aid in interpreting and diagnosing the causes of such behavior for both IDSes. Another interpretation can be consistent with our expectations, i.e., that signature-based IDS are dependent on the existence of the signature of an individual attack instance. For reference, a full list of the 19 attacks included in the experiment and their detection status is given in Table VI.2.

Table VI.1: Example of calculated metrics.

Attack category	DRPAT (Bro)	DRPAT (Snort)	DR (snort)	DR (Bro)
Category 746	1/1	0/1	1/19	12/19
Category 1565	8/14	1/14		
Category 3494	0/1	0/1		
Category 3798	3/3	0/3		

6.5.3. Background Experiments

At the time of carrying out this experiment, we have no reliable traffic traces that can be manipulated and replayed by our tool. In order to generate background traffic, we have used the Distributed Internet Traffic Generator (D-ITG) [Botta07]. The advantage of such tools is that it can produce traffic at packet level that accurately replicate appropriate stochastic processes for both IDT (Inter Departure Time) and PS (Packet Size) random variables, conforming with exponential, uniform, Cauchy, normal, Pareto or other distributions. It can also generate traffic at network, transport, and application layer. This experiment gave results very similar to the one without background traffic, with very few false alarms. This may be due to the regularity of the synthesized background traffic.

Table VI.2: List of attacks (metasploit exploits) included in the evaluation dataset and their detection status.

Attackname	Category	Detection status	
		<i>Bro</i>	<i>Snort</i>
windows/http/edirectory_imonitor	746	Detected	
windows/iis/ms03_007_ntdll_webdav	1565	Detected	Detected
windows/smb/ms03_049_netapi	1565		
windows/smb/ms04_007_killbill	1565	Detected	
windows/smb/ms04_011_lsass	1565	Detected	
windows/smb/ms04_031_netdde	1565	Detected	
windows/smb/ms05_039_pnp	1565	Detected	
windows/smb/ms06_025_rasmans_reg	1565		
windows/smb/ms06_025_rras	1565		
windows/smb/ms06_040_netapi	1565		
windows/smb/ms06_066_nwapi	1565	Detected	
windows/smb/ms06_066_nwwks	1565		
windows/smb/msdns_zonename	1565	Detected	
netware/smb/lsass_cifs	1565	Detected	
windows/ssl/ms04_011_pct	1565		
linux/madwifi/madwifi_giwscan_cb	3494		
windows/http/trendmicro_officescan	3798	Detected	
windows/mssql/ms02_056_hello	3798	Detected	
windows/wins/ms04_045_wins	3798	Detected	

6.6. Results

The test results show that the tool kit implementation comprises several of the desired features. Table VI.3 compares our toolkit with other broad categories of evaluation tools. However, the experiments have also revealed some limitations, either as technical problems or as a lack of desired features in the current implementation, which leave additional room of improvements.

For example, we discovered that single-criterion selection is not sufficient because there are more sophisticated situations where evaluators need to select very specific attacks, based on combined criteria. Furthermore, we need sometimes to select attacks based on a particular application type (e.g., web servers) or a particular application product (e.g., *Apache2*) rather than on the platform specificities.

Another example of missing features is the automatic generation of evaluation reports that combine and correlate information about generated attack scenarios and the IDS alerts, and accordingly calculate metrics. These tasks have proved to consume as much time and effort as the selection and generation of datasets and the construction of the test bed. It would also be interesting to build a user-friendly graphical user interface that unifies parameterization and management of the entire evaluation. This implies grouping a configuration panel for both attack and background traffic generation; a control panel for

starting, stopping and monitoring the evaluation process; and a report panel for displaying and analyzing results.

Regarding the technical problems, the most annoying one is related to receiving outputs from victim machines over communication pipes when executing commands that correspond to attack actions. Apparently, the current implementation of *IO* pipe (e.g., *IO::popen*) streams in *Ruby* is problematic as it often blocks if the stream buffer is empty or even becomes broken if the sent data goes beyond the buffer size. The *Metasploit* team has developed some classes such as *BidirectionalPipe* and *sessions* classes that partially resolve some of pipe communication problems. However, receiving data is still tricky as we often need to know a priori how much data will be received.

Unfortunately, these problems have, up to now, prevented us from testing the generated scenarios effectively because we cannot analyze commands outputs, which is necessary for building sophisticated scenarios. Moreover, we cannot execute interactive programs correctly. Consequently, we cannot check attack success or failure automatically and have to do this manually. Moreover, the limitations of reading and writing from pipes have negative effects on the management of virtual victims. According to our design (Chapter 4), we use virtual machines as targets of the generated attacks. To reduce the cost of the test bed, we dynamically allocate a virtual machine to an attack session. Therefore, we only start the selected victim machine during a window of time corresponding to the estimated time of attack session. Then, we need to stop it and revert it to a clean snapshot.

We have surveyed existing virtualization products but unfortunately, remote operations such as starting, stopping, reverting virtual machines, which is necessary for managing evaluation automatically, are hardly supported. There are some products that do not support remote management of guest machines at all, whereas others, such as *VirtualBox* or *Vmware workstation*, partially support a command line console management after having established an *ssh* connection. The only one that we found fully supporting remote management with its own commands is *Vmware server*. However, it is very heavy and requires a powerful machine. There is another limitation of using virtual victims: some operating systems, such as Mac OS, cannot be virtualized (i.e., installed as guest machines).

Another important limitation is related to the number and the diversity of attack tools already classified and included in the evaluation toolkit. They are currently very limited when compared with the number of available signatures. The current version of *snort* (2.8), for example, has about 9000 rules while the number of attacks that we have classified is about 300. Even with the assumption that an attack can have many signatures; the number of classified attacks is so limited that it does not allow performing reliable evaluations.

In what concerns our tool for manipulating and replaying traffic traces, the traffic analysis function works well with small-size traces but it has shown a poor scalability when using large ones. The characteristics of the generated trace globally conform to the desired characteristics, but it needs deeper analysis for special issues such as timing and addressing characteristics and whether it keeps the characteristics of the original trace.

Returning to the comparison with other approaches reported in Table VI.3, we observe that almost no individual tool provides all features that are critical for the evaluation. Fortunately, we managed to implement most of the desired features in our toolkit to fill this gap in the field. Moreover, it is worth noting that the automatic generation of executable attack scenarios which is present in our toolkit was completely missing in all the others.

As will be discussed in the next chapter, the shortcomings of our tool kit give a solid basis for enhancements, which we can plan in the near future.

Table VI.3: Comparison of the main features between our toolkit and the other tools.

	Elementary Attack	Automatic scenarios	Background traffic	Integrated attack selection	Recording attack traces	Evaluation management	Detection avoidance Techniques	Modifying attack Parameters	Live-Evaluation	Evaluation Report	Various attack types
Attack datasets (DARPA, CRC)	✓		✓		✓						✓
Network scanners (nmap)	✓						✓	✓	✓	✓	
Vulnerability scanners (nessus)	✓					✓	✓	✓	✓	✓	
Penetration testing tools	✓			✓			✓	✓	✓		✓
Attack mutation tools	✓						✓		✓		
Our toolkit	✓	✓	✓	✓	✓	✓		✓	✓		✓

6.7. Conclusion on our experiments

Our preliminary experiments show that the ideas presented in the previous chapters are applicable and implementable by using existing technologies. However, it shows also that the evaluation toolkit still needs more tweaking to be more robust and more user-friendly.

Even though limited, the experiment presented above shows to which extent our approach and evaluation toolkit can facilitate the evaluators' tasks. It allows performing selective evaluation based on various criteria where interesting attack test cases can be selected according to our classification's attributes (an approach similar to CTM described in Chapter 4): according to specific vulnerabilities, by operating system, etc. Accordingly, an evaluation dataset will be generated either in scenario mode evaluation or elementary attack mode.

Because of some technical problems (described above), the scenarios generated during this evaluation are quite superficial as they consist of sequential executions of commands without regarding the output of the executed commands and whether the attack was successful or not. However, the effects of this limitation are concealed by two facts. First, recent studies such as {Alata07} have shown that attacker actions are not necessarily executed successfully all the time. Attack scenarios contain misused or misspelled commands, or are even sometimes attempted in a trial and error manner. Second, the objective of this work is not to launch attacks, penetrating into and compromising systems or networks. It rather aims to generate attack datasets similar to what may be seen in the real world. The analysis of honeypot data proved that real world attacks consist of both successful, complete attack scenarios as well as incomplete, failing scenarios.

Generally, the results are encouraging and seem to be promising but this approach requires a close cooperation of the community because some activities such as the classification should be a collective work and cannot be done individually. We have already discussed with the *Metasploit* team about adding more classification attributes to their module description. Similarly, integrating the classification process in the main work stream of creating exploits or analyzing malware would be very beneficial. We suggest, in the next chapter, some directions to improve both the approach and our toolkit.

VII. Chapter 7: Conclusions

In this last chapter, we draw our conclusions of the whole work carried out throughout the thesis. First, we present a brief overview of the subject and the carried work. Then, we enumerate the contributions of our work. Finally, to give a clear insight of our future vision about the subject, we outline the possible research and empirical work directions that can be a pursuit of our work.

7.1. An Overview

Although the research domain was very active in the last few years, enhancements in intrusion detection and prevention are not proportional to the efforts and the budgets dedicated to this purpose. One of the main reasons for this issue is the lack of effective methods and tools for evaluating new detection techniques and algorithms.

Generally, security-related evaluation is a delicate subject because it raises several challenges. In particular, IDS evaluation should consider not only its normal use during usual operation or its predictable abnormal use, but it should consider also the operation under unpredictable or even unknown conditions. Rather than using systems and networks by following usage instructions of the accompanying manual, attackers usually use them in unfamiliar ways and by entering unexpected inputs. Moreover, the evaluation of such systems is multi disciplinary by nature since it requires various knowledge from different domains such as security basics and tools; attack techniques and tools; software testing; performance evaluation; reverse engineering; operating system and network administration; etc.

However, because we believe that any significant improvement in the intrusion detection and prevention field must pass through careful evaluation either performed by IDS developers or IDS users. This thesis aims at helping with such evaluations.

Before proposing any solution for the problem of evaluating intrusion detection systems, we have analyzed most of the previous published evaluation experiments to identify their strengths, weaknesses and why they often produce biased results. We have identified some common problems in these evaluations, the most significant ones being the lack of a systematic methodology and the use of non-representative datasets. Consequently, we have defined two main objectives for this work: (1) providing a systematic methodology and (2) creating representative evaluation datasets. The first allows evaluators to perform IDS evaluations easily in a well-structured manner, and the second allows performing non-biased and more comprehensive evaluations.

While working on the evaluation methodology, we were confronted with several questions that are closely related and should be answered in order to achieve the aforementioned broad objectives. For example, what are the properties of the real workload of IDSes? And how can we provide representative datasets?

To answer the first question we have carried out a thorough analysis of real workloads of different IDS types to figure out their main characteristics. To simplify the analysis, we divided IDS workload into two main parts: an attack dataset and a background dataset. We have characterized both components, but with more emphasis on the attack dataset.

Regarding the question of providing representative datasets, two sub questions are raising: how can we characterize such a representative dataset? Then, how to generate it? In fact, characterizing pertinent attack datasets is a non-trivial task because of the huge number of attacks, the unpredictable behavior of attackers, and the ambiguity related to attack tools and methods. Furthermore, even if we can identify relevant attack test cases, generating an attack dataset that correspond to such representative attack test characteristics is also a difficult task by itself.

In order to characterize representative attack test cases, we have proposed a classification scheme accompanied by a selection scheme based on the classification tree method (CTM). The new classification scheme was created by analyzing the characteristics of elementary attacks as well as studying existing classifications to avoid their limitations. The objective from creating such a new classification is to enable performing classification-based selections of attack test cases. The idea is not new, since it was inspired from a software testing concept known as “equivalence classes”. Even if this approach has been already suggested for IDS evaluation by *Puketza et al.* {Puketza97} and there was an

attempt to evaluate IDSes based on attack classes {Alessandri04}, it remained as a theoretical suggestion and had not been applied yet for security testing.

Then we tackled the problem of generating evaluation datasets. The background datasets were briefly treated by identifying their security-relevant characteristics that may affect evaluation results. By contrast, we concentrated on attack datasets by working on two axes: (1) characterizing real attacks and (2) analyzing the output of existing attack generation tools. Therefore, we continued analyzing attack characteristics, this time as scenarios not as elementary attacks. For this purpose, we have analyzed attack information available in high-interaction honeypot logs, malware analysis reports as well as attack incident reports. We managed to derive a model for attack processes that describes attacker actions abstractly.

On the other hand, we have noticed that almost all attack generation tools generate fictitious datasets with characteristics far away from real attacks. For example, attack datasets were often composed by forging network packets containing the very specific signature part that invokes the alerts while ignoring the rest of an attack trace. Therefore, to make the attack traffic more representative, we decided to create attack datasets by using attack tools similar to those that may be used by attackers.

Better solutions to generate attacks can be based on penetration testing tools such as *metasploit*. However, we are sometimes interested not only in breaking (penetrating) into systems but also we are interested in post-access actions, which can be very significant for security analysis or evaluation (in evaluating state-full IDSes by example). Therefore, we decided also to generate attack scenarios that correspond to the different scenarios enabled by such tools.

In order to generate scenarios automatically, we have derived some theoretical models (an attacker competence model and a statistical parameterization model) added to the attack process model. Then, we proposed an approach that integrates all these models in addition to a constraint-based approach to transform the abstract scenarios into executable scenarios that are adapted to the targeted system.

Finally, we have developed a proof-of-concept implementation of the whole approach for selecting and generating attack datasets. The implementation is based on *metasploit*, which already contains "access" actions as well as other useful auxiliary attack tools (e.g., DoS, scanners, etc.). Actually, our implementation extends *metasploit* framework by a new plugin module to imitate (and automate) post-penetration actions that can probably be carried out by attackers. For example, executing a sequence of commands to browse the victim machine, upload a piece of malware, connect back to another machine, etc.

7.2. Summary of Contributions

Despite the challenges inherent in IDS evaluation, we managed to identify several directions that can lead together to a plausible solution. The contributions of this work complement each other with an ultimate goal to improve IDS evaluation and consequently the IDSes themselves. Although the techniques we have developed are mainly intended for IDS evaluations, they can be used to test or evaluate other security tools (e.g., intrusion prevention systems). Furthermore, they can be used for global security assessment of systems and networks.

Our main contributions can be summarized as follows:

- We have elaborated an evaluation methodology that considers IDS evaluation as a systematic process. The aim is to improve the whole evaluation process and to render it more structured and well engineered.
- In an attempt to apply the concept of equivalence classes, we have established a classification scheme for elementary attacks combined with a test case selection mechanism to select relevant attack test cases. Both the classification scheme and the selection mechanism will help in converting class-based evaluation from abstract notions to concrete, representative and suitable attacks.
- Based on the characterization and focusing on the attack dataset, we have constructed an attack process model to represent attack scenarios. The advantage of this model is that it can be used in generating representative attack scenarios in a manner that is practically feasible. In our approach, the selected scenarios can be refined by constraints related to the targeted environment, the attacker behavior, the most likely attacks, etc.

- We have developed a set of tools that support the different tasks of the evaluation process (i.e., dataset characterization, test-case selection and dataset generation):
- A classification database to store information about elementary attacks. Evaluators can access and modify the database via its front-end.
- A program to automate the analysis of data provided by high interaction honeypots. Thanks to this program, we can extract commands and carry out scenario-based analysis (according to the attack process model). It produces a list of executed commands and tools, statistics about commands themselves and their sequences.
- An attack injection tool (AIT), which integrates the ideas that we have proposed. It benefits from the above tools to extract information necessary for the selection of attack test-cases (attack classification) and attack test-case generation (using the attack process model as well as the statistical parameterization model).
- A network traffic manipulator for background datasets that allows the analysis of recorded traffic traces. It provides information about addresses ports, sessions, timing, etc. Moreover, it allows the modification of these fields. Therefore, we can maintain the consistency between the attack dataset and the background datasets. For example, to keep address consistency, we can either use addresses extracted from the trace in the attack dataset or replace addresses in the trace according to the addresses used in the attack dataset, which may be deduced from honeypot data or explicitly specified by the evaluator.
- We have also implemented an evaluation platform that, in addition to AIT, contains a scheduler and a mechanism for managing platform resources (i.e., creating and destroying virtual machine victims).

7.3. Future work

This subject is so rich that we can suggest many ideas while yet leaving room for improvements. Hereafter, we cite some future research and work directions that can be immediately initiated or implemented. Although we have proposed several ideas in this dissertation, this is just the beginning and there are still many future works that can be established in both short and long-term as a pursuit of our work. For clarity, we divided the future work section into two main parts: future research and future development and empirical work. Such a separation is not clear-cut as both are closely related and we can find in the following lists some works that lies in between.

7.3.1. Research Work

We can distinguish three research axes related to attack datasets, background datasets, and the platform respectively:

A) Attack Dataset:

- **Attack classification:** The classification is in its early version and may require more refinement. To achieve it, we need to consolidate the classification scheme as well as classifying more instances of elementary attacks. Having a solid and rigorous classification scheme can encourage the integration of this scheme into existing tools such as *metasploit* and into de-facto standards such as CME, or even the creation of a new standard classification for elementary attacks. At long term, a stable classification can allow class-based scores to evaluate security tools, (e.g., how many classes can be detected by an IDS). Furthermore, we can elaborate standard benchmarking suites for the same purpose (e.g., a suite of elementary attacks that contains a representative instance from each attack class).
- **Refine and improve the attack process model:** The proposed model needs to be updated, validated and refined on a regular basis because attack trends, tools and techniques vary with time. Moreover, although we have analyzed many malware and attack incidents in order to produce the current version of the model, it may still be incomplete or inaccurate. Therefore, we should extend the analysis to cover more attack scenarios in order to answer the following questions:

- Are there additional attack steps that are not taken into account in the current model?
- Are there attack actions that are so significant that they need to be considered as a separate step? For example, establishing a back connection can be very important.
- Do we need to split steps into finer-grain steps? For example, the step *EP* (execute program) can be split further to distinguish executing a malicious program from executing normal commands. Moreover, studying the convergence of our model with other malware behavioral models such as {Jacob08a} may lead to a modified model with finer grained steps or perhaps result in a completely new model for malware attack process.
- Should we need to explicitly distinguish between server-side and client-side actions?
- Should we consider separately attack steps that are particular to specific attacks such as Cross Site Scripting (XSS)?
- **Extend scenario computing to include multi-hop attack scenarios:** Our current approach supports single-hop attacks that often comprise a single victim or several victims starting from outside attack machines. However, attackers may attack more victim machines that are accessible from the first victim.
- **Refine and improve attacker competence and statistical parameterization models**
 - Regarding the attacker competence model, it may be useful to search in some other related domains such as psychology, ergonomics, artificial intelligence, etc.
 - Unfortunately, the statistical model that we have used is a simplistic one because of the limited amount of available data. Nevertheless, when sufficient data are gathered by honeypots, we should analyze their characteristics against known statistical model or create another suitable model.
- **Consider detection avoidance techniques:** In order to be sufficiently representative, the attack dataset must contain different forms of attacks that result from applying evasion and illusion techniques. Unless these techniques are considered, the representativeness of attack datasets could be questionable. However, for the sake of simplicity, we preferred to postpone applying detection avoidance techniques into attack datasets as future research.

B) Background Dataset:

- A deeper characterization is required for the HIDS background dataset that considers not only the common features such as addressing and timing, but also the contents and the nature of background activities run by the host.
- Providing mechanisms to distinguish and isolate benign activities from malicious ones in both network traffic and host based datasets. This will not only lead to gather good datasets of malicious and benign traffic but may lead also to a good detection mechanism.

C) Platform:

- Study the possibility of using available test beds such as *PlanetLab* {Planetlab08}, *ReAssure* {Reassure08}, *EMULAB* {Emulab08} and *DETERlab* {Deterlab08} for the purpose of testing security.
- Study the side effects of using virtual machine victims rather than real victims.

7.3.2. Development and Empirical Work

Regarding the evaluation tool kit, there are several enhancements either to improve the tools that we have already developed or creating more tools. Improvement examples include but are not limited to the following:

A) Data Analysis:

- **Honeypots:**
 - Improve the data capture mechanism currently installed on the LAAS high-interaction honeypot. The current mechanism allows recording commands executed by attackers but it is not aware of commands output or the subsequent data typed into interactive programs (e.g., editing a file into *vi* or even simply typing back the

- password after executing the command *passwd*).
- Implement more vulnerabilities in the high-interaction honeypot as a source of data for scenario-based analysis in addition to the dictionary attack against weak *ssh* passwords that is actually implemented.
- Integrate attack process analysis mechanisms into the high-interaction honeypot of *LAAS* and perhaps into the honeynet of *Leurrecom*.
- **Malware analysis:**
 - Automate the analysis of malware programs according to our scenario-based approach. An automatic analysis will dramatically reduce the time and effort required for this task.

B) Dataset Selection and Generation:

- **Attack dataset:**
 - Currently, our tool supports attack selection based on a single criterion (either classification-based, by operating system, by vulnerability or by date). The next version is intended to enable multi-criteria selection.
 - Up to now, we have focused on generating human-interactive scenarios. Actions executed in malware scenarios differ in the level of granularity since they are often executed at system call or instruction levels. We believe that generating scenarios with low level interactions corresponding to malware scenarios is very important for evaluating not only intrusion detection systems but also to evaluate other security tools such as antivirus tools.
 - Improve the transformation of abstract scenarios into concrete executable scenarios. Actually, the transformation is rudimentary and might be quite superficial. For example, more sophisticated algorithms to chain commands can produce more intelligent command sequences or associate a block of commands to an abstract attack step.
- **Background dataset:**
 - Implement tools for analyzing and generating background datasets for host-based IDS evaluations. There is a severe lack in this area as it needs more effort to be done.

C) Evaluation Platform

- Extend the current implementation to support large scale evaluations in terms of the network size, the number of attacks and victim machines.
- Add more vulnerable software and attack tools beside *metasploit* to the tools repository.
- Develop a graphical user interface to manage and configure the whole process evaluation; to monitor attack sessions progress and to view results.
- Enhance logging capabilities and add automatic analysis of IDS alerts to our evaluation toolkit.

D) Experimental Evaluation

- **Perform more comprehensive IDS evaluations:** During this work, we have concentrated on the analysis of evaluation problems and how to find appropriate solutions. This results in the creation of procedures and tools help in performing more robust evaluations. However, this was -unfortunately- at the cost of time left for carrying out IDS evaluations. We have already performed some exploratory evaluations but the next step is to perform more serious evaluations that cover different issues of intrusion detection. The time spent in characterizing attacks and developing tools is fruitful because thanks to the evaluation toolkit, evaluations are expected to take much less effort and time.
- **Evaluate IPSes and other security tools:** We kept our approach as general as possible to be applicable to other security tools, which often share similar input workloads even though they have different purposes. As an evaluation target, the first candidate would

be intrusion prevention systems that have underlying techniques closely similar to detection techniques. However, an IPS has substantially different focuses from an IDS, and therefore its evaluation ought to be handled quite differently.

E) Simulation-based Evaluations:

The theoretical models that we have proposed can be used as a basis for simulating the attack process, which can be useful for educative purposes or limited security tool evaluation. We have already some ideas for implementing this. Moreover, at long term it can be enriched by other models to characterize networks and hosts, in order to simulate the whole network.

Appendix A: Malware Analysis

At the time of writing, CME list contains 39 malware representing the most famous and most dangerous ones. We present, hereafter a brief description of attack steps followed by each malware. In addition to MITRE's site, we relied on information from other security sites on the Internet such as: www.ca.com and www.viruslist.com.

Each attack step is given a unique symbol as follows (The meaning of attack steps is further explained in Section 4):

- R: Reconnaissance
- VB: Victim Browsing
- EP: Execute Program
- GA: Gain Access
- IMC: Implant Malicious Code
- CDI: Compromise Data Integrity
- DoS: Denial of Service
- HT: Hide Traces

We recall the directives that we have mentioned in Chapter 4 for abstracting attack steps:

- In fact all attack steps can be viewed as a program execution (EP). To guide the assignment of abstract steps, we define a partial order relation to determine which abstract step should be assigned to an attack step:
 - $IMC > CDI > EP$
 - $TH > CDI > EP$
 - $VB > EP$
 - $TH > DoS$
- If the executed attack step contributes to the installation of the malicious code, therefore it is classified as IMC. Else, if it modifies the file system, the configuration files, the registry keys, or the environment variables it should be considered as a CDI. Otherwise, it is considered as EP.
- If the executed attack step hides information or block access to information about the malicious code, therefore it is classified as TH. Else, if it modifies the file system, the configuration files, the registry keys, or the environment variables it should be considered as a CDI. Otherwise, we consider it as EP.
- Searching information remotely from the victim or a potential victim is a reconnaissance step (R). On the other hand searching information locally on the victim is a victim browsing step (VB).
- If the attack step blocked/stops/compromises access to services that provides information about the malicious activity, it is considered as a trace hiding step (TH). If the blocked/stopped/compromised service does not hide information, consider it as a DoS step.

CME-245: 2004-11-22

CA: Win32.Bagle.AR

Kaspersky: Email-Worm.Win32.Bagle.au

Description: Win32.Bagle.AR is a worm that spreads via e-mail attachment and peer-to-peer file sharing. The worm harvests addresses from the local address book and installs a proxy server. This Bagle variant spreads either as Windows PE EXE file or a Windows Control Panel Applet (CPL) file, both about 20 KB in size.

Description of Attack Steps	Abstracted Step of attack process
1. received as attachment or put in shared P2P directories	Implicit AG, IMC
2. Wait until executed by the user	EP
3. Query operating system to determine the location of the current System folder	VB
4. Copy itself to %system%\wingo.exe where %system% is the installation directory of the windows operating system. There are several variations of the copied file (i.e., wingo.exe)	IMC
5. Modify register to ensure that this copy is executed at each Windows start	CDI
6. It searches for files with e-mail addresses, as well as any directories whose names contain the string "shar".	VB
7. Listen on TCP port 81 (open Backdoor)	EP
8. Delete registry files belonging to security tools such Antivirus and firewall tools to avoid detection.	TH
9. Terminate processes whose name contains strings that may exist in security tools.	TH
10. Download arbitrary files from pre-specified URLS	IMC
11. Execute downloaded files	EP
12. Send itself to collected email addresses	EP

CME-473: 2004-11-22

CA: Win32.Bagle.AQ

Kaspersky: Email-Worm.Win32.Bagle.at

Description: A variant of the Bagle worm..

Description of Attack Steps	Abstracted Step of attack process
1. received as attachment or copied to shared P2P directories	Implicit AG, IMC
2. Wait until executed by the user	EP
3. Query operating system to determines the location of the current System folder	VB
4. Copy itself to %system%\wingo.exe where %system% is the installation directory of the windows operating system. There are several variations of the copied file (i.e., wingo.exe)	IMC
5. Modify register to ensure that this copy is executed at each Windows start	CDI
6. It searches for files with e-mail addresses, as well as any directories whose names contain the string "shar".	VB
7. Listen on TCP port 81 (open Backdoor)	EP
8. Delete registry files belonging to security tools such Antivirus and firewall tools to avoid detection.	TH
9. Terminate processes whose name contains strings that may exist in security tools.	TH
10. Download arbitrary files from pre-specified URLS	IMC
11. Execute downloaded files	EP
12. Send itself to collected email addresses	EP

CME-901: 2005-02-28

CA: Win32.Mydoom.AZ

Kaspersky: Email-Worm.Win32.Mydoom.am

Description: A variant of the Mydoom worm. It spreads via email through SMTP, gathering target recipients from the Windows Address Book, the Temporary Internet Files folder, and certain fixed drives. Notably, it skips email addresses that contain certain strings.

Description of Attack Steps	Abstracted Step of attack process
1. Use search engines such as Google and yahoo to collect email addresses	R
2. Received as email attachment	Implicit AG, IMC
3. Wait until executed by the recipient user	EP
4. Query operating system to determines the location of the current System folder	VB
5. It copies itself to %windows%\java.exe	IMC
6. Modify windows registry to ensure that it will be executed at each windows startup	CDI
7. drops the file %windows%\service.exe	IMC
8. Modify windows registry to make execute service.exe at startup	CDI
9. Searches the local fixed drives for email addresses	VB
10. 10. Save the collected addresses in a temp file	CDI
11. 11. attempt to download arbitrary malicious files	IMC
12. Execute the downloaded file	EP
13. It sends itself via email attachment	EP

CME-414: 2005-04-21

CA: Win32.Sober.M

Kaspersky: Email-Worm.Win32.Sober.n

Description: A mass-mailing worm arrives in an email messages that is designed to trick users into thinking that someone else is receiving their email. It has been distributed as a 73,541-byte, UPX packed Win32 executable or as a 73,699 byte ZIP archive.

Description of Attack Steps	Abstracted Step of attack process
1. Received as email attachment	Implicit AG, IMC
2. waits until executed by the recipient	EP
3. Query the operating system for the location of the directory of temporary files %Temp%	VB
4. Once launched, it creates a txt file	CDI
5. opens the txt file in windows notepad	EP
6. Queries the operating system for the location of windows directory %windows%	VB
7. It copies itself to %windir%\config\system\services.exe	IMC
8. Modifies windows registry to load itself every time the system is rebooted	CDI
9. Creates auxiliary files and drop them in %windows%\config\system folder	CDI
10. Searches email addresses in local files	VB
11. Saves harvested addresses at maddys.xyz file	CDI
12. Terminates the process mrt.exe	TH

CME-456: 2005-05-02

CA: Win32.Sober.N

Kaspersky: Email-Worm.Win32.Sober.p

Description:

A mass-mailing worm that sends itself as an email attachment to addresses gathered from the compromised computer. It uses its own SMTP engine to spread. The email may be in either English or German. It has been distributed as a 53,554-byte, UPX packed Win32 executable and as a 53,728 byte ZIP archive.

Description of Attack Steps	Abstracted Step of attack process
1. Received as email attachment	Implicit AG, IMC
2. waits until executed by the recipient	EP
3. Displays a message box	EP
4. Queries the operating system for the location of windows directory %windows%	VB
5. copies itself in to %Windows%\Connection Wizard\Status in three files : csrss.exe, smss.exe, services.exe	IMC
6. executes services.exe which than runs smss.exe and csrss.exe	EP
7. Modifies windows registry to load itself every time the system is rebooted	CDI
8. Searches email addresses in local files	VB
9. deletes files on the infected system, with names matching the following criteria: a*.exe, luc*.exe, ls*.exe, luu*.exe	CDI
10. Terminates the process mrt.exe	TH
11. displays a message box titled "AntiVirus-AntiSpyware", with the message "No Viruses, Trojans or Spyware found! Status: OK"	EP
12. drops auxiliary files in the directory: "%Windows%\Connection Wizard\Status and the directory %System%	CDI

CME-766: 2005-06-01

CA: Win32.Glieder.AG

Kaspersky: Email-Worm.Win32.Bagle.bo

Description:

A Trojan that interferes with the operation of security software by ending processes, stopping services, removing registry entries, and deleting files.

Description of Attack Steps	Abstracted Step of attack process
1. Received as email attachment	Implicit AG and IMC
2. Waits for execution by recipient	EP
3. Copies itself to %System%\winshost.exe or the file wiwshost	IMC
4. Modifies windows registry to ensure it will be executed when Windows is started	CDI
5. Download arbitrary files	IMC
6. Execute the malicious downloaded files	EP
7. Kill processes associated with antivirus and other security-related applications	TH
8. Modify windows registry to disable or lowering security setting	TH
9. Rename files related to antivirus and other security-related applications	TH
10. alters the %System%\drivers\etc\hosts so that users of the infected machines will be unable to access hard coded url of security-related sites to prevent users from removing the worm	DoS

CME-978: 2005-07-04

Trend Micro: TROJ_DLOADER.UX

Description:

A Trojan downloader that downloads malware from several different Internet addresses. A Trojan application is a malware with no capability to spread into other systems. They are usually downloaded from the Internet and installed by unsuspecting users.

Description of Attack Steps	Abstracted Step of attack process
1. Downloaded from the Internet or installed by unsuspecting users	Implicit AG, IMC
2. Waits for execution by some user	EP
3. Copies itself to windows folder	IMC
4. Waits for an active Internet connection to download the file HHTZ.EXE from a particular site.	IMC
5. Modifies windows registry to ensure its automatic execution at every system startup:	CDI
6. Executes the downloaded file which provides a backdoor	EP

CME-402: 2005-07-04

CA: Win32.DlWreck

Description: A Trojan that downloads and executes other malware (adware and spyware programs) on the infected system. They also inject their main functionality into Internet Explorer in an attempt to hide their presence and bypass some personal firewalls.

Description of Attack Steps	Abstracted Step of attack process
1. Downloaded by the user	AG, IMC
2. executed by user	EP
3. queries the following registry key to determine the location of Internet Explorer: HKLM\Software\Microsoft\Windows\CurrentVersion\App Paths\IEXPLORE.EXE	VB
4. executes an instance of Internet Explorer without displaying any windows	EP
5. It injects code into this process	CDI
6. downloads files which are usually other malware from different domains	IMC
7. Executes the downloaded files	EP

CME-746: 2005-07-08

CA: Win32.SillyDl.RW

Description: A Trojan downloader that downloads an executable from a malware Web site.

Description of Attack Steps	Abstracted Step of attack process
1. Received by email or downloaded by the user	AG, IMC
2. executed by user	EP
3. downloads a file from the domain "distributed-****.com" and saves it to the file location "C:\temp\tmp.exe"	IMC

CME-323: 2005-07-08

CA: Win32.Conferox

Kaspersky: Trojan-Downloader.Win32.Small.arf

Description: A password stealing trojan that downloads a dll, which is used to intercept user keystrokes. The collected data is posted to another web site.

Description of Attack Steps	Abstracted Step of attack process
1. Received and downloaded via email	AG, IMC
2. Executed by user	EP
3. the trojan copies itself to the %System%/service folder as "explorer.exe"	IMC
4. queries the operating system to determine the location of '%System%'	VB
5. Registers itself as a service (on Windows NT/2K/XP) or modifies the registry in order to execute at the next reboot (on Windows 9.x)	CDI
6. drops a 20,380-byte DLL file called "dll.dll" and uses it to intercept an affected user's keystrokes	IMC
7. collects some sensitive and confidential data, such as. user e-mail account details, OS details, IP address, cached passwords, banking details, etc	R
8. Posts the collected information to an external site.	EP

CME-875: 2005-07-15

CA: Win32.Reattle.A

Kaspersky: Net-Worm.Win32.Lebreat.c

Description: A mass-mailing worm that opens a back door and attempts to propagate by exploiting the Microsoft Windows Local Security Authority Service (LSASS) Remote Buffer Overflow (as described in Microsoft Security Bulletin MS04-011) on TCP port 445.

Description of Attack Steps	Abstracted Step of attack process
1. Received as email attachment	Implicit AG and IMC
2. Waits for execution by the recipient	EP
3. Query the operating system for the location of the directory of temporary files %system%	VB
4. Copies itself to %System%\windows.exe also copies itself to %System%\attach.tmp	IMC
5. modifies the registry so that this copy is executed at each Windows start:	CDI
6. Search all fixed drives for email addresses	VB
7. Save the gathered addresses on the local file system	CDI
8. Sends itself to the collected emails	EP
9. Runs an FTP server to upload copies of the worm	EP
10. Modifies system registry to lower or disable security settings	TH
11. Download arbitrary files	IMC
12. Perform a DoS attack against www.symantec.com	DoS

CME-540: 2005-08-17

CA: Win32.Tpbot.A

Kaspersky: Net-Worm.Win32.Bozori.a

Description: A worm that opens an IRC controlled backdoor and exploits the Microsoft Windows Plug and Play Buffer Overflow Vulnerability (described in Microsoft Security Bulletin MS05-039 at <http://www.microsoft.com/technet/security/Bulletin/MS05-039.msp>) on TCP port 445.

Description of Attack Steps	Abstracted Step of attack process
1. Checks the vulnerability of randomly generated IP addresses	R
2. Exploits the vulnerability	AG, IMC
3. Instructs the target to connect back to the source system	EP
4. Downloads the worm using TFTP	IMC
5. Executes the worm	EP
6. creates a mutex file on the local system	CDI
7. Query the operating system for the location of the directory of temporary files %system%	VB
8. Copies itself to %System% directory as wintbp.exe	IMC
9. modifies the registry to execute this copy at each Windows start:	CDI
10. Creates a batch file	CDI
11. Launches the batch file to delete the original executables	CDI
12. connects to an IRC server	EP

CME-702: 2005-08-25

CA: Win32.Drugtob.A

Kaspersky: Backdoor.Win32.IRCBot.et

Description: A worm that opens a back door and exploits the Microsoft Windows Plug and Play Buffer Overflow Vulnerability (described in Microsoft Security Bulletin MS05-039 at <http://www.microsoft.com/technet/security/Bulletin/MS05-039.msp>) on TCP port 445. The worm also acts as an IRC-controlled backdoor, allowing a controller unauthorized access to the infected machine.

Description of Attack Steps	Abstracted Step of attack process
1. searches random IP addresses for potential targets, checking for vulnerable systems via port 445.	R
2. If it successfully exploits this vulnerability, the worm opens a remote shell on the target system	AG
3. instruct the target to connect back to the source system and download the worm using the Windows TFTP client	EP
4. It downloads the worm using a file name in the form run<number>.exe	IMC
5. Run the downloaded worm file	EP
6. When initially executed, Drugtob checks if the following registry entry exists: HKLM\Software\Drudgebot\Halt, if it exists then displays a message box and exit	VB
7. determines the location of the current Program Files folder and the windows folder by querying the operating system	VB
8. The worm terminates processes related to other malware and adware	TH
9. deletes folders and files from the %Program Files% and %System% directories	CDI
10. Deletes Registry Values	CDI
11. Connect to IRC server	EP

CME-637: 2005-08-25

CA: Win32.Drugtob.B

Description: A worm that opens a back door and exploits the Microsoft Windows Plug and Play Buffer Overflow Vulnerability (described in Microsoft Security Bulletin MS05-039 at <http://www.microsoft.com/technet/security/Bulletin/MS05-039.msp>) on TCP port 445.

Description of Attack Steps	Abstracted Step of attack process
1. searches random IP addresses for potential targets, checking for vulnerable systems via port 445	R
2. opens a remote shell on the target system	AG
3. uses this shell to instruct the target to connect back to the source system	EP
4. downloads the worm using a file name in the form 'run<number>.exe'	IMC
5. Runs the downloaded file	EP
6. checks if the following registry entry exists: HKLM\Software\Drudgebot\Halt	VB
7. If its value is "TRUE", the worm displays a message box and then exits	EP
8. determines the location of the current Program Files folder by querying the operating system and the system directory	VB
9. copies itself to "%System%\wbev\windrg32.exe"	IMC
10. sets registry value so that this file is executed at each Windows start	CDI
11. acts as a very basic FTP server on the originating system, listening on port 24463	EP
12. creates the mutex "windrg322" to avoid running multiple copies of the worm at the same time	CDI
13. executes a simple batch file that deletes it from the path it was originally executed from	TH
14. terminates processes, which are related to other malware and adware	TH
15. deletes folders from the %Program Files% directory	CDI
16. deletes files from the %System% directory	CDI
17. Deletes registry values	CDI

Appendix B: Datasets and Dataset Generation Tools

1. Ready-made Datasets

A) DARPA's Datasets

The background traffic was synthesized according to statistics collected from different networks in several Air Force bases (about 50 air bases). The attack component consists of attack scripts collected from specialized sites and mailing lists on the Internet or written by hand in addition to some live attacks.

A network test-bed was implemented to create a live traffic, which contains various types of traffic similar to that may be generated by hundreds of users on thousands of hosts. Seven weeks of training data, which contains background traffic and labeled attacks in addition to two weeks of unlabeled test data was recorded.

Table B.1 Main features of evaluation datasets

Dataset	Main Features	Advantages	Disadvantages
DARPA dataset {Kendall99}, {Lippmann00a}, {Lippmann00b}	<ul style="list-style-type: none"> background + attack traces dataset in tcpdump, system log formats network traffic + Host-based logs 1998 version: (32 attack types, 120 instance, No windows logs) 1999 version: (56 attacks, win NT) simulated human users on real machines victims are real machines 	<ul style="list-style-type: none"> established and consolidated the main concepts of dataset generation envisaged anomaly based as well as signature based, HIDS and NIDS contains attack scenarios 	<ul style="list-style-type: none"> quite obsolete (not updated since 2000) non representative [Mchu]
CRC Dataset {Massicotte06}	<ul style="list-style-type: none"> traces in tcpdump format only attack traces, no background traffic contains traces for successful and failed attack attempts Vmware machines Especially for testing signature-based NIDS 128 exploit corresponds to 92 vulnerability & 108 target configuration 10446 traces result from combinations of Vul, Config and target + 3549 result from applying evasion on successful attacks 	<ul style="list-style-type: none"> quite recent (2006) well documented apply evasion techniques large number of traces large number of victim systems configuration (About 200 OS versions) about 2343 attack traffic traces 	<ul style="list-style-type: none"> only elementary attacks launched by VEP (Vulnerability Exploit program)
Karalon's Traffic IQ {Trafficiq08}	<ul style="list-style-type: none"> security audit tool test packet filtering security devices traffic library containing more than 1000 standard protocol and threat traffic files background and malicious 	<ul style="list-style-type: none"> highly configurable through a GUI The traffic library is updated regularly with approximately 50 new threat traffic files added each month 	<ul style="list-style-type: none"> only NIDS attack insertion only via threat traffic files. commercial tool non flexible addition of attack scenarios

There were two groups of datasets: the first consists of network traffic for testing Network Intrusion Detection Systems (NIDS). It was collected by sniffing packets from certain points on the test bed. The other group was particularly gathered to test host-based intrusion detection systems (HIDS). It consists of audit data from Solaris hosts, full disk dumps from UNIX victim machines and sun basic security module

(BSM). All were obtained from victim hosts.

About 300 instances of 38 different attack types were used in DARPA 1998 and 201 instances of 56 attack types in 1999 version {Kendall99}. They were categorized mainly in four categories: *Probe or scanning*, *Remote to local (R2L)*, *User to Root (U2R)*, and *Denial of Service (DoS)*. Attacks and attack scenarios were selected according to the following rules:

- 1- Use a transcript of an actual intrusion if available to develop attack scenarios
- 2- Otherwise, use the publicly known attacks, which can be found on the specialized sites or mailing lists.
- 3- The novel attacks were created by taking yet unexploited vulnerabilities or weaknesses.

Generated packets artificially result in “perfect” traffic that does not contain broken or strange packets not conforming to protocol specifications. This kind of spurious packet is quite common in the Internet traffic {Coredump08}.

Moreover, it was noticed that packets are more regular than expected in the reality {Mahoney03}. For example:

- SYN packets have always a 4-byte set of options whereas in reality it ranges from 0 to 28 bytes of options.
- TCP window size has seven fixed values. However, it usually ranges from 512 to 32120.
- 29 distinct source addresses account for 99.9% of the traffic. In real world networks with similar characteristics, over 24,000 unique addresses were counted.
- TTL values are similar in most of the packets where 9 values out of 256 were used. In reality, 177 different values can be observed.

Similarly, for the TOS there only four different values were used whereas in real traffic about 40 values can be observed.

2. Dataset Generation Tools

A) Background Traffic Generators

Generic traffic generators such as IPERF, SmartBits and ttcp generate IP, TCP and UDP packets using pseudo-random techniques. This works well for network devices (e.g., routers, switches, bridges) because they often do not care about the packet payload. Unfortunately, intrusion detection systems do care about the packet payload contents. Thus, randomly generated sequences of bytes are not suitable for testing IDS. When the IDS encounters such sequence of bytes, it can decide to report it as anomalous or just drop the packet without any further analysis.

More intelligent traffic generators have been created to solve this problem. Harpoon {Sommers04}, by example, generates TCP and UDP traffic based on real parameters (i.e., packet lengths, temporal and spatial characteristics) automatically extracted from routers in live environments. Other tools are aware of network protocols and use protocol automata to, for instance, to generate an emulated traffic that corresponds to what a real user would generate {Barford98}. This category of tools can create HTTP, FTP, Telnet and mail sessions that look syntactically correct. Unfortunately, this approximated representation of protocols is quite idealistic and does not correspond to what real users do.

B) IDS Stimulators

Table B.2: IDS stimulators.

Tool	Main Features	Advantages	Disadvantages
IDS Stimulators Snot & Stick	<ul style="list-style-type: none"> generate attacks from existing IDS signatures to test other IDS 	<ul style="list-style-type: none"> IDS Inter-comparison 	<ul style="list-style-type: none"> relies only on publicly available signatures
IDSwakeUp	<ul style="list-style-type: none"> a suite of tools to generate false attacks that mimics well known ones a bourshell script that allow the execution of hping2 and iwu 	<ul style="list-style-type: none"> TTL, source and destination addresses are changeable Iwu can sends a buffer as datagram 	
Mucus	<ul style="list-style-type: none"> signature-based stimulator 	<ul style="list-style-type: none"> provides a cross-testing technique by using signature of one NIDS to test another NIDS 	<ul style="list-style-type: none"> lack of publicly available signature sets

C) Vulnerability Scanners and Network Mappers

In addition to the previous list, scanning tools such as Nessus {Nessus08} and Nmap {Nmap08} are often used to supply the reconnaissance attacks.

Nmap:

Nmap ("Network Mapper") is a free and open source (license) utility for network exploration or security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. It was designed to rapidly scan large networks, but works fine against single hosts. Nmap runs on all major computer operating systems, and both console and graphical versions are available.

Nessus:

The free open source Nessus vulnerability scanner has become the de facto standard. Nessus risk assessments are powered by its robust database of vulnerability checks called Nessus Attack Scripting Language (NASL) scripts. NASL is a powerful scripting language for writing security checks for Nessus to perform.

The vulnerability database is updated daily. Administrators can access the database simply by running a command-line tool that ships with Nessus (nessus-update-plugins). This tool connects to the Nessus site and downloads all of the latest attack plugins available. Unlike commercial off-the-shelf vulnerability scanners, Nessus scans devices and identifies remote flaws in the system. In addition, it checks the host for vulnerabilities and identifies missing patches.

D) Penetration Testing Frameworks

Immunity's *CANVAS* {Canvas08} and *Core Impact* {Coreimpact08} from Core Security Technologies are even more sophisticated. They include in a single application a network mapper, a vulnerability scanner, an exploit execution environment, and a report generator. They also contain some basic functionalities to perform "stealth" attacks, i.e., to apply some form of obfuscation to the executed attacks. On the open source side, we find *metasploit*, which we have described in details in Section 4.7.

Table B.3: Penetration testing tools

Tool	Main Features	Advantages	Disadvantages
Metasploit {Metasploit08}	<ul style="list-style-type: none"> a framework for penetration testing, IDS signature development, and exploit research written in Ruby language Current version (v. 3.0) contains: <ul style="list-style-type: none"> 177 exploits 104 payloads 17 encoders 5 NOPs 30 aux 	<ul style="list-style-type: none"> Free Open source Extensible tool: new exploits and features can be added as modules and plugins Maintained updates Has a web interface and a GUI 	<ul style="list-style-type: none"> scheduling the whole test is quite fastidious and attack scenarios should be programmed separately
Core Impact {Coreimpact08}	<ul style="list-style-type: none"> automated penetration testing tool automates steps of attack process automatic report generation - automatic clean up after test 	<ul style="list-style-type: none"> has a GUI frontend regular updates of well-tested exploits fully customizable possibility of integration with other security tools 	<ul style="list-style-type: none"> Commercial
CANVAS {Canvas08}	<ul style="list-style-type: none"> Automated exploitation system over 150 exploits (by november 2007) 4 exploits/month, in average 	<ul style="list-style-type: none"> ability to add custom exploits Has a GUI 	<ul style="list-style-type: none"> Commercial

E) Attack Mutation Tools

Attackers want to be invisible by any mean to avoid detection. In addition to creating new variations of their malicious code, they also follow several evasion and illusion techniques to render the IDS either blind or confused {Ptacek98}.

The use of variations of attacks to test intrusion detection systems and other security mechanisms has recently received considerable attention. These tools combine the functionalities of an exploit execution environment with one or more IDS evasion techniques. One of the earliest works that systematically considered attack variations as a way to test intrusion detection systems was Raffael Marty's Thor {Marty02}. Thor's design included the possibility to generate variations at both the network and the application layers using mutant modifier to generate several variations of the network traffic from the same attack exploit. However, Thor's implementation is limited and the only mentioned result is the application of an evasion technique based on IP fragmentation to an HTTP-based attack.

Another interesting work is MACE {Sommers04}. MACE is a toolkit for malicious traffic generation written in Python. The malicious traffic is created according to three models: an exploit model that describes the parts of the attack, an obfuscation model that defines the obfuscation elements at both network and application layer, and the propagation model that controls the order in which the victim hosts will be attacked.

The tool AGENT {Rubin04} relies on a formal, logical induction of mutations. It generates variations of a single instance of attack automatically using inference rules. A more powerful tool called *Splloit* {Vigna04} that generates attack variations by applying mutant operators to an exploit template.

Appendix C: Glossary

<i>Activity</i>	An action or a set of actions that generates <i>events</i> in the system.
<i>Malicious activity</i>	An activity carried out by an attacker that aims to violate the security policy.
<i>Normal activity</i>	An activity carried out within the context of normal operations without the intent to compromise the security policy.
<i>Analytical evaluation</i>	It is usually based on some model of the system under study not the system itself and can be done at any stage during the development cycle.
<i>Attack</i>	A malicious technical interaction that attempts to exploit some vulnerability as a step towards achieving the final goal of the attacker.
<i>Attack scenario</i>	A set of organized activities, including malicious activities and apparently normal activities, which are executed in sequence or in parallel to achieve the attacker's goal.
<i>Availability</i>	The prevention of unauthorized retention of information.
<i>Benchmarking</i>	Is a specific kind of test; it is the process of comparing the performance of two or more systems by measurements. Often, a series of experiments are performed on systems using a reference set of benchmarks (datasets/programs).
<i>Classification</i>	Systematic arrangement, in groups or categories, according to established criteria
<i>Confidentiality</i>	The prevention of unauthorized disclosure of information.
<i>Counter-measures</i>	To avoid any confusion we prefer using the term " <i>counter measure</i> " for security mechanisms while keeping the term " <i>measure</i> " for its original use in metrology.
<i>Evaluation</i>	Is generally defined as "the act of placing a value on the nature, character, or quality of something".
<i>Evaluation Methodology</i>	Is a detailed plan that describes all the steps of the evaluation process.
<i>Evaluation Technique</i>	We can identify three main techniques that can be applied to evaluate computer-based systems or applications
<i>Event</i>	A thing that happens or takes place; a change in system state.
<i>Exploit (noun)</i>	A script, a program, a mechanism or other technique by which some vulnerability is used to realize an attack or a part of an attack.
<i>False negative</i>	An event corresponding to the incorrect decision to rate an activity as being not erroneous; also called a "miss".
<i>False positive</i>	An event corresponding to the incorrect decision to rate an activity as being erroneous; also called a "false alarm".
<i>Integrity</i>	The prevention of modification or unauthorized suppression of information.
<i>Intrusion</i>	A <i>malicious</i> fault externally induced resulting from an <i>attack</i> that

	has succeeded in exploiting some vulnerability.
<i>Intrusion Detection</i>	The set of practices and mechanisms used towards detecting errors that may lead to security failure, and diagnosing intrusions and attacks.
<i>Intrusion Detection System (IDS)</i>	An implementation of practices and mechanisms of intrusion detection.
<i>Intrusion Prevention System (IPS)</i>	A kind of IDS that prevents occurrence of attacks rather than simply detecting errors due to attacks. It extends the functionality of IDS by a response unit to prevent attacks and / or limit their effects. Typical responses to intrusions may include dropping suspicious traffic at the firewall, denying user access to resources as they exhibit anomalous behavior, etc.
<i>Malicious activity</i>	An activity carried out by an attacker that aims to violate the security policy.
<i>Measure (noun)</i>	This term has several different meaning
<i>Metrics</i>	The criteria used to evaluate the system. Any metric should have both a definition and a unit of measure. By example
<i>Normal activity</i>	An activity carried out within the context of normal operations without the intent to compromise the security policy.
<i>Security failure</i>	Any violation of a security property of the intended <i>security policy</i> . This includes any violation of the <i>confidentiality</i> , the <i>integrity</i> or the <i>availability</i> .
<i>Security policy</i>	A description of 1) the security properties to be fulfilled by a computing system; 2) the rules according to which the system security state can evolve.
<i>Simulation</i>	This technique is applicable at any stage too. System behaviors, interactions between system components and the inputs/outputs are simulated,
<i>Taxonomy</i>	The study of the general principles of scientific classification
<i>Test or measurement</i>	An actual implementation or a prototype of the system is evaluated against real or synthetic inputs (workloads or test dataset) in order to study the system behavior and its reactions.
<i>Test-bed or workbench</i>	The platform of test (software/hardware/network architecture) on which the test can be carried out.
<i>True negative</i>	The event corresponding to the correct decision to rate an activity as being not erroneous.
<i>True positive</i>	The event corresponding to the correct decision to rate an activity as being erroneous; also called a "hit."
<i>Vulnerability</i>	A fault created during development of the system, or during operation, that could be exploited to create an intrusion.

References

- [Abouelkalam03] Anas Abou El Kalam, "Modèles et Politiques de Sécurité Pour les Domaines de la Sante et Des Affaires Sociales", *LAAS Report* 03578, Institut National Polytechnique de Toulouse-INPT, Toulouse, France, 2003.
- [Alata06] E. Alata, V. Nicomette, M. Kaaniche, M. Dacier, M. Herrb, "Lessons learned from the deployment of a high-interaction honeypot", *Sixth European Dependable Computing Conference (EDCC '06)*, Coimbra, Portugal, pp. 39-46, 2006.
- [Alata07] Eric Alata, "Observation, caractérisation et modélisation de processus d'attaques sur Internet", *PhD thesis*, Institut National des Sciences Appliquées (INSA) de Toulouse, France, 2007.
- [Alessandri04] Dominique Alessandri, "Attack-Class-Based Analysis of Intrusion Detection Systems", *Ph.D. Thesis*, University of Newcastle upon Tyne, School of Computing Science, Newcastle upon Tyne, UK, 2004.
- [Allen00] J. Allen, A. Christie, W. Fithin, J. McHugh, J. Pickel and E. Stoner, "State of The Practice of Intrusion Detection Technologies", *Technical Report* CMU/SEI-99-TR-028, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.
- [Anderson08] Ross J. Anderson, "Security Engineering: A Guide to Building Dependable Distributed Systems", Book, 2nd edition, John Wiley & Sons, Inc, 2008.
- [Anderson80] J. P. Anderson, "Computer security threat monitoring and surveillance", *Technical Report*, James P. Anderson Company, Fort Washington, Pennsylvania, USA, April 1980.
- [Antonatos04] S. Antonatos, K. G. Anagnostakis and E. P. Markatos, "Generating Realistic Workloads for Network Intrusion Detection Systems", *ACM SIGSOFT Software Engineering Notes*, Vol. 29, Issue 1, pp. 207-215, 2004.
- [Athanasopoulos03] N. Athanasopoulos, R. Abler, J. Levine, H. Owen, and G. Riley, "Intrusion detection testing and benchmarking methodologies", *Proceedings of First IEEE International Workshop on Information Assurance (IWIAS 2003)*, Darmstadt, Germany, pp. 63-72, 24 March 2003.
- [Avizienis04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", *IEEE Transactions on Dependable and Secure Computing*, Vol. 1 (1), pp. 11-33, 2004.
- [Avtest08] Tests of Anti Virus Software (2008): <http://www.av-test.org/>.
- [Axelsson00] Stefan Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy", *Technical Report* 99-15, Department of Computer Engineering, Chalmers University, Goteborg, Sweden, 2000.
- [Axelsson98a] S. Axelsson, "Research in Intrusion-Detection systems: A Survey", *Technical Report* 98-17, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 1998.
- [Balzarotti06] D. Balzarotti, "Testing Network Intrusion Detection Systems", *PhD Thesis*, Polytechnical Institute of Milano, Italy, 2006.
- [Barford98] Paul Barford and Mark Crovella, "Generating representative Web workloads for network and server performance evaluation", *SIGMETRICS Performance Evaluation Review*, Vol. 26, pp. 151-160, 1998.
- [Bishop95] Matt Bishop, "A standard audit trail format", *In Proceedings of the 1995 National Information Systems Security Conference*, Baltimore, Maryland, USA, pp. 136-145, 1995.
- [Bishop99] M. Bishop, "Vulnerabilities Analysis", *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99)*, West Lafayette, Indiana, USA, 1999.
- [Botta07] Alessio Botta, Alberto Dainotti and Antonio Pescapè, "Multi-protocol and multi-platform traffic generation and measurement", *INFOCOM 2007 DEMO Session*, Alaska, USA, pp. N/A, 2007.

-
- [Bouti94] Abdelkader Bouti and Daoud Ait Kadi, "A State-Of-The-Art Review of FMEA/FMECA", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 1 (4), pp. 515-543, 1994.
- [Caida08] CAIDA: The Cooperative Association for Internet Data Analysis (2008): <http://www.caida.org/research/security/>.
- [Canvas08] IMMUNITY's CANVAS: An Automated Exploitation System (2008): <https://www.immunitysec.com/products-canvas.shtml>.
- [Capec08] CAPEC: The Common Attack Pattern Enumeration and Classification (2008): <http://capec.mitre.org/data/index.html>.
- [Cert08] Computer Emergency and Response Team, Software Engineering Institute, Carnegie Mellon University: <http://www.cert.org/stats/>.
- [Chen98] S. Staniford-Chen, B. Tung and D. Schnackenberg, "The Common Intrusion Detection Framework (CIDF)", *Information Survivability Workshop*, Orlando FL, USA, 1998.
- [Cheung03] S. Cheung, U. Lindqvist and M. Fong, "Modeling Multistep Cyber Attacks for Scenario Recognition", *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, DC, USA, pp. 284-292, 2003.
- [Cme08] CME: Mitre's Common Malware Enumeration (2008): <http://cme.mitre.org/>.
- [Coredump08] Museum of broken packets (2008): <http://lcamtuf.coredump.cx/mobp/>.
- [Coreimpact08] CORE IMPACT: Automated Penetration Testing (2008): <http://www.coresecurity.com/?module=ContentMod&action=item&id=591>
- [Cte08] CTE: Classification Tree Editor (2008): <http://www.systematic-testing.com>.
- [Cuppens00] Frédéric Cuppens, Rodolphe Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks", *Proceeding of the Third International Workshop on the Recent Advances in Intrusion Detection (RAID'2000)*, Toulouse, France, pp. 197-216, 2000.
- [Cve08] Mitre's Common Vulnerability and Exposure (CVE) (2008): <http://cve.mitre.org/>.
- [Dacier94] Marc Dacier and Yves Deswarte, "Privilege Graph: an Extension to the Typed Access Matrix Model", *Proceedings of the Third European Symposium on Research in Computer Security (ESORICS '94)*, London, UK, pp. 319-334, 1994.
- [Dacier99] Marc Dacier and Dominique Alessandri, "VulDa: A Vulnerability Database", *Proceedings of the 2nd Workshop on Research with Security Vulnerability Databases*, Purdue, USA, 1999.
- [Dahl06] Ole Martin Dahl and Stephen D. Wolthusen, "Modeling and Execution of Complex Attack Scenarios using Interval Timed Colored Petri Nets", *Proceedings of the Fourth IEEE International Workshop on Information Assurance (IWIA '06)*, Washington DC, USA, pp. 157-168, 2006.
- [Debar00] Hervé Debar, Marc Dacier and Andreas Wespi, "A Revised Taxonomy for Intrusion Detection Systems", *Annales des Telecommunications*, Vol. 55, Number: 7-8, pp. 361-378, 2000.
- [Debar02] Hervé Debar and Benjamin Morin, "Evaluation of the Diagnostic Capabilities of Commercial Intrusion Detection Systems", *International Symposium Recent Advances in Intrusion Detection (RAID 2002)*, LNCS 2516, Zurich, Switzerland, Springer, LNCS 2516, pp. 177-198, 2002.
- [Debar04] Hervé Debar, Benjamin Morin, Frédéric Cuppens, Fabien Autrel, Ludovic Mé, Bernard Vivinis, Salem Benferhat, Mireille Ducassé and Rodolphe Ortalo, "Détection d'intrusions : corrélation d'alertes", *Technique et Science Informatiques*, Vol. 23 (3), pp. 359-390, 2004.
- [Debar05] Hervé Debar and Jouni Viinikka, "Intrusion Detection: Introduction to Intrusion Detection and Security Information Management", *Foundations of Security Analysis and Design III*, Lecture Notes in Computer Science, Volume 3655, 2005. pp. 207-236.
- [Debar07] H. Debar, D. Curry, B. Feinstein, "The Intrusion Detection Message Exchange Format (IDMEF)", *RFC 4765*, Internet Engineering Task Force, IETF, 2007.
- [Debar98] Hervé Debar, Marc Dacier, Andreas Wespi and Stefan Lampart, "An Experimentation Workbench for Intrusion Detection Systems", *Research Report RZ 2998*, IBM Zurich Research Laboratory, Switzerland, 1998.

-
- [Debar99] Hervé Debar, Marc Dacier and Andreas Wespi, "Towards a taxonomy of intrusion-detection systems", *The International Journal of Computer and Telecommunications Networking*, Vol. 31, N. 9, pp. 805-822, 1999.
- [Denning87] Dorothy E. Denning, "An intrusion-detection model", *IEEE Transaction on Software Engineering*, Vol. 13, pp. 222-232, 1987.
- [Deterlab08] DETERlab Testbed: cyber-DEfense Technology Experimental Research LABoratory Testbed (2008): <http://www.isi.edu/deter/>.
- [Earl01] Earl Carter, "Cisco Secure Intrusion Detection System", Cisco Press, 2001.
- [Eckmann02] Steven T. Eckmann, Giovanni Vigna, Richard A. Kemmerer, "STATL: an attack language for state-based intrusion detection", *Journal of Computer Security*, Vol. 10, pp. 71-103, 2002.
- [Eeye08] EEYE published advisories (2008): <http://www.eeye.com/html/Research/Advisories/>.
- [Emulab08] Emulab: Network Emulation Testbed (2008): <http://www.emulab.net/index.php3?stayhome=1>.
- [Ermopoulos06] Charis Ermopoulos and William Yurcik, "NVision-PA: A Tool for Visual Analysis of Command Behavior Based on Process Accounting Logs (with a Case Study in HPC Cluster Security)", *Proceeding of IEEE International Conference on Cluster Computing (CLUSTER'06)*, Barcelona, Spain, 25-28 September, 2006.
- [Fragroute08] Fragroute: a TCP/IP fragmenter (2008): www.monkey.org/~dugsong/fragroute.
- [Futoransky03] Ariel Futoransky, Luciano Notarfrancesco, Gerardo Richarte and Carlos Sarraute, "Building Computer Network Attacks", *Technical Report*, CoreLabs, Core Security Technologies, USA, 2003.
- [Gadelrab06] Mohammed Gadelrab and A. Abou El Kalam, "Testing Intrusion Detection Systems: An Engineered Approach", *Proceeding of IASTED International Conference on Software Engineering and Applications (SEA 2006)*, USA, 2006.
- [Gadelrab07] Mohammed S. Gadelrab, Anas Abou El Kalam and Yves Deswarte, "Defining categories to select representative attack test-cases", *Proceedings of the 2007 ACM workshop on Quality of protection (QoP '07)*, Alexandria, Virginia, USA, pp. 40-42, 2007.
- [Gadelrab08a] Mohammed Gadelrab, Anas Abou El Kalam et Yves Deswarte, "Modélisation des processus d'attaques pour l'évaluation des IDS", *Act de la 3ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information*, Loctudy, France, 2008.
- [Garg06] Ashish Garg, Shambhu Upadhyaya and Kevin Kwiat, "Attack Simulation Management for Measuring Detection Model Effectiveness", *Proceedings of The Second Secure Knowledge Management Workshop (SKM 2006)*, Brooklyn, NY, USA, 2006.
- [Grochtmann95] M. Grochtmann and J. Wegener, "Test Case Design Using Classification Trees and the Classification-Tree Editor CTE", *Proceedings of the 8th International Software Quality Week (QW '95)*, San Francisco, USA, pp. 1-11, May 1995.
- [Hansmann03] Simon Hansmann, "A Taxonomy of Network and Computer Attacks", *Diplom Thesis*, University of Canterbury, New Zealand, 2003.
- [Helmer01] Guy Helmer, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller and Robyn Lutz, "A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System", *Proceedings of the 1st Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, USA, pp. 207-220, 2001.
- [Hoglund05] Greg Hoglund and Jamie Butler, "Rootkits: Subverting the Windows Kernel", Addison-Wesley Professional, 2005.
- [Honeynet04] The Honeynet Project, "Know Your Enemy: Learning about Security Threats", Addison-Wesley Professional, 2nd Ed., 2004.
- [Howard98] John Douglas Howard, "An analysis of security incidents on the Internet 1989-1995", *PhD Thesis*, Carnegie Mellon University, USA, 1998.
- [Httpperf08] Httpperf: web testing tool (2008): <http://www.hpl.hp.com/research/linux/httpperf/>.
- [ITSEC91] Information Technology Security Evaluation Criteria (ITSEC), Provisional Harmonised Criteria, Version 1.2, Commission of the European Communities, 1991.

-
- [Ida07] The ida vulnerability (2007): <http://research.eeye.com/html/advisories/published/AD20010618.html>.
 - [Ilgun95] Koral Ilgun, Richard A. Kemmerer and Phillip A. Porras, "State Transition Analysis: A Rule-Based Intrusion Detection Approach", *Software Engineering*, Vol. 21, pp. 181-199, 1995.
 - [Ilog08] ILOG Constraint Programming (2008): <http://www.ilog.com/products/cp/>.
 - [Itg08] Distributed Internet Traffic Generator (2008): <http://www.grid.unina.it/software/ITG/link.php>.
 - [Jacob08a] Grégoire Jacob, Eric Filiol and Hervé Debar, "Malware as Interaction Machines: a New Framework for Behavior Modeling", *Journal in Computer Virology*, Vol. 4, pp. 235-250, 2008.
 - [Jacop08] JaCoP: Java Constraint Programming Library (2008): <http://jacop.cs.lth.se/>.
 - [Jain91] Raj Jain, "The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling", Wiley- Interscience, 1991.
 - [John07] John the Ripper password cracker (2007): <http://www.openwall.com/john/>.
 - [Jones00] Anita K. Jones and Robert S. Sielken, "Computer System Intrusion Detection: A Survey", *Technical Report*, Computer Science Department, University of Virginia, Virginia, USA, 2000.
 - [Jonsson97] Erland Jonsson and Tomas Olovsson, "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior", *IEEE Transactions on Software Engineering*, Vol. 23, N. 4, pp. 235-245, 1997.
 - [Jpcap08] Jpcap (2008): <http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html>.
 - [Kaaniche06] Mohamed Kaaniche, Y. Deswarte, Eric Alata, Marc Dacier and Vincent Nicomette, "Empirical analysis and statistical modeling of attack processes based on honeypots", *Proceeding of Workshop on Empirical Evaluation of Dependability and Security (WEEDS), DSN'2006*, Philadelphia, USA, pp. 119-124, 2006.
 - [Kemmerer98] Richard A. Kemmerer, "NSTAT: A Model-based Real-time Network Intrusion Detection System", *Technical Report TRCS97-18*, University of California, CA, USA, 1998.
 - [Kendall99] K. Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems", *Master Thesis*, MIT Department of Electrical Engineering and Computer Science, USA, 1999.
 - [Killourhy04] Kevin S. Killourhy, Roy A. Maxion and Kymie M. C. Tan, "A Defense-Centric Taxonomy Based on Attack Manifestations", *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, Florence, Italy, pp. 102- 111, 2004.
 - [Kumar95] S. Kumar, "Classification and Detection of Computer Intrusions", *PhD thesis*, Purdue University, USA, 1995.
 - [Leurrecom08] Leurrecom Honeypot Project (2008): <http://www.leurrecom.org/>.
 - [LiChen03] L. Chen, "Modeling Distributed Denial of Service Attacks and Defenses", *PhD thesis*, Carnegie Mellon University, USA, 2003.
 - [Lindqvist97] U. Lindqvist and E. Jonsson, "How to systematically classify computer security intrusions", *Proceedings of IEEE Symp. on Security and Privacy*, Oakland, CA, USA, pp. 154-163, 1997.
 - [Lippmann00a] Richard Lippmann, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyszogrod, Robert Cunningham and Marc Zissman, "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation", *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, Los Alamitos, CA, USA, pp. 12-26, 2000.
 - [Lippmann00b] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba and Kumar Das, "Analysis and Results of the 1999 DARPA Off-Line Intrusion Detection Evaluation", *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID 2000)*, Toulouse, France, Springer, LNCS 1907, pp. 162-182, 2000.
 - [Lough01] D. Lough, "A Taxonomy of Computer Attacks with Applications to Wireless Networks", *PhD thesis*, Virginia Polytechnic Institute and State University, USA, 2001.

-
- [Lundin02] Emilie Lundin and Erland Jonsson, "Survey of Intrusion Detection Research", *Technical Report-02-04*, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 2002.
- [Maftia03] MAFTIA Consortium, "Conceptual Model and Architecture of MAFTIA", D. Powell and R. Stroud, Eds., MAFTIA (Malicious and Accidental Fault Tolerance for Internet Applications) project deliverable D21, LAAS-CNRS Report 03011, 2003.
- [Mahoney03] Matthew V. Mahoney and Philip K. Chan, "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection", *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID)*, Pittsburgh, PA, USA, pp. 220-237, 2003.
- [Marty02] R. Marty, "Thor: A Tool to Test Intrusion Detection Systems by Variations of Attacks", *Diploma Thesis*, Swiss Federal Institute of Technology, ETH Zurich, 2002.
- [Massicotte06] Frederic Massicotte, Francois Gagnon, Yvan Labiche, Lionel Briand and Mathieu Couture, "Automatic Evaluation of Intrusion Detection Systems", *Proceedings of the 22nd Annual Computer Security Applications Conference*, Washington, DC, USA, pp. 361-370, 2006.
- [Maxion00] Roy A. Maxion and Kymie M. C. Tan, "Benchmarking Anomaly-Based Detection Systems", *Proceedings of the 2000 International Conference on Dependable Systems and Networks*, New York, NY, USA, IEEE, pp. 623-630, 2000.
- [Maxion98] Roy A. Maxion, "Measuring Intrusion Detection Systems", *The First International Workshop on Recent Advances in Intrusion Detection (RAID-98)*, Louvain-la-Neuve, Belgium, pp. No printed proceeding, 1998.
- [McDermott00] J. P. McDermott, "Attack net penetration testing", *Proceedings of the 2000 workshop on New security paradigms (NSPW '00)*, New York, USA, pp. 15-21, 2000.
- [McHugh00a] John McHugh, "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory", *ACM Transaction on Information System Security*, Vol. 3, pp. 262-294, 2000.
- [Mchugh00b] John Mchugh, "The 1998 Lincoln Laboratory IDS Evaluation: A critique", *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID)*, Toulouse, France, pp. 145-161, 2000.
- [Mell03] P. Mell, V. Hu, R. Lipmann, J. Haines and M. Zissman, "An overview of issues in testing intrusion detection systems", *Technical Report: NIST IR 7007*, National Institute of Standard and Technology, USA, 2003.
- [Metasploit08] The Metasploit Framework (2008): <http://www.metasploit.com/>.
- [Michel01] Cédric Michel, Ludovic Mé, "ADeLe: an attack description language for knowledge-based intrusion detection", *Proceedings of the 16th international conference on Information security: Trusted information: the new decade challenge*, Paris, France, pp. 353-368, 2001.
- [Moore01] A. Moore, R. Ellison and R. Linger, "Attack modeling for information security and survivability", *Technical notes*, Software Engineering Institute, Carnegie Mellon University, 2001.
- [Morin07] Benjamin Morin and Ludovic Mé, "Intrusion detection and virology: an analysis of differences, similarities and complementarity", *Journal in Computer Virology*, Vol. 3, pp. 39-49, 2007.
- [Mozart08] The Mozart Programming System (2008): www.mozart-oz.org.
- [Mukherjee94] B. Mukherjee, L. T. Heberlein and K. N. Levitt, "Network intrusion detection", *Journal of IEEE Network*, Vol. 8, pp. 26-41, 1994.
- [Mutz03] D. Mutz, G. Vigna and R. A. Kemmerer, "An Experience Developing an IDS Stimulator for the Black-Box Testing of Network Intrusion Detection Systems", *Proceedings of the 2003 Annual Computer Security Applications Conference (ACSAC '03)*, Las Vegas, Nevada, USA, pp. 374-383, 2003.
- [Myers79] Glenford J. Myers, "The Art of Software Testing", John Wiley & Sons, 1979.
- [Mé01] Ludovic Mé and Cédric Michel, "Intrusion Detection: A Bibliography", *Bibliographic Report-SSIR-2001-01*, Supélec, Rennes, France, 2001.

-
- [NASL08] Michel Arboi, The NASL2 reference manual (2008): www.nessus.org/doc/nasl2_reference.pdf.
- [Nessus08] Nessus: Network Vulnerability Scanner (2008): <http://www.nessus.org/nessus/>.
- [Neumann89] Peter G. Neumann and Donn B. Parker, "A summary of computer misuse techniques", *Proceedings of the 12th National Computer Security Conference*, Baltimore, MD, USA, pp. 396-407, 1989.
- [Nikto08] NIKTO: Open Source web server scanner (2008): <http://www.cirt.net/code/nikto.shtml>.
- [Nmap08] NMAP: Free and Open Source Network Mapper (2008): <http://nmap.org/>.
- [Nss08] Network Security Services Group, NSS IDS Evaluation (2005): <http://www.nss.co.uk/ips>.
- [Ortalo99] Rodolphe Ortalo, Yves Deswarte and Mohamed Kaâniche, "Experimenting with Quantitative Evaluation Tools for Monitoring Operational Security", *IEEE Transaction on Software Engineering*, Vol. 25, Number 5, pp. 633-650, 1999.
- [Ossec08] OSSEC: Open Source Host-based Intrusion Detection System (2008): <http://www.ossec.net/>.
- [Osvdb08] Open Source Vulnerability Database-OSVDB (2008): <http://osvdb.org/>.
- [Packit08] Packit project at SourceForge.net (2008): <http://sourceforge.net/projects/packit>.
- [Pang05] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, Brian Tierney, "A First Look at Modern Enterprise Traffic", *Proceeding of ACM SIGCOMM/USENIX Internet Measurement Conference*, Berkeley, CA, USA, pp. 217-231, 2005.
- [Paxson97] Vern Paxson, "End-to-end Internet packet dynamics", *SIGCOMM Comput. Commun. Rev*, Vol. 27, pp. 139-152, 1997.
- [Paxson97-a] Vern Paxson; Sally Floyd, "Why we don't know how to simulate the Internet", *Proceedings of the 29th conference on Winter simulation*, Atlanta, Georgia, United States, pp. 1037-1044, 1997.
- [Paxson99] Vern Paxson, "Bro: a System for Detecting Network Intruders in Real-time", *Computer Networks (Amsterdam, Netherlands: 1999)*, Vol. 31, pp. 2435-2463, 1999.
- [Planetlab08] Planetlab: an Open Platform for Developing, Deploying, and Accessing Planetary-scale services (2008): <https://www.planet-lab.org/>.
- [Porras97] P. A. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", *Proceeding of 20th NIST National Information Systems Security Conference*, Baltimore, Maryland, USA, pp. 353-365, 1997.
- [Prelude08] Web Site of Prelude Hybrid Intrusion Detection Framework (2008): <http://www.prelude-ids.org/>.
- [Ptacek98] Thomas H. Ptacek and Timothy N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", *Technical Report*, Secure Networks Inc., Alberta, Canada, 1998.
- [Puketza96] N.J. Puketza, K. Zhang, M. Chung, B. Mukherjee, R. A. Olsson, "A Methodology for Testing Intrusion Detection Systems", *IEEE Transactions on Software Engineering*, Vol. 22, Number 10, pp. 719-729, 1996.
- [Puketza97] Nicholas Puketza, Mandy Chung, Ronald A. Olsson and Biswanath Mukherjee, "A Software Platform for Testing Intrusion Detection Systems", *Journal of IEEE Software*, Vol. 14, Number: 5, pp. 43-51, 1997.
- [Ramsbrock07] D. Ramsbrock, R. Berthier, M. Cukier, "Profiling Attacker Behavior Following SSH Compromises", *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '07)*, Edinburgh, UK, pp. 119-124, 2007.
- [Raynal04] F. Raynal, Y. Berthier, P. Biondi, D. Kaminsky, "Honeypot forensics part 1: analyzing the network", *Security & Privacy, IEEE*, Vol. 2, pp. 72-78, 2004.
- [Realsecure08] IBM Internet Security Systems (ISS) products (2008): <http://www-935.ibm.com/services/us/index.wss/offerfamily/iss/a1029097>.
- [Reassure08] Reassure: a Safe Virtual Imaging Instrument for Logically Destructive experiments (2008): <http://projects.cerias.purdue.edu/reassure/index.html>.

-
- [Roesch99] Martin Roesch, "Snort-Lightweight Intrusion Detection for Networks", *Proceedings of the 13th USENIX Systems Administration Conference (LISA'99)*, Seattle, Washington, USA, pp. 229-238, 1999.
- [Rubin04] S. Rubin, S. Jha, B. P. Miller, "Automatic generation and analysis of NIDS attacks", *Proceeding of 20th Annual Computer Security Applications Conference (ACSAC'04)*, Tucson, Arizona, USA, pp. 28-38, 2004.
- [Ruby08] Ruby Programming Language (2008): <http://www.ruby-lang.org>.
- [Samhain07] The SAMHAIN file integrity / host-based intrusion detection system (2007): <http://www.la-samhna.de/samhain/index.html>.
- [Schaelicke03] Lambert Schaelicke, Thomas Slabach, Branden Moore and Curt Freeland, "Characterizing the Performance of Network Intrusion Detection Sensors", *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, USA, Springer, LNCS 2820, pp. 155-172, 2003.
- [Schneier99] B. Schneier, "Attack Trees: Modeling Security Threats", *Dr. Dobb's Journal*, Vol. 24, Number 12, pp. 21-29, 1999.
- [Sheyner02] O. Sheyner, J. Haines, S. Jha, R. Lippmann and J. M. Wing, "Automated generation and analysis of attack graphs", *Proceeding of 2002 IEEE Symposium on Security and Privacy*, Oakland, California, USA, pp. 273-284, 2002.
- [Slammer07] Analysis of Microsoft SQL Server Sapphire Worm (2007): <http://research.eeye.com/html/advisories/published/AL20030124.html>.
- [Snapp91] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, S. Smaha, T. Grance, D. Teal, and D. Mansur, "DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and an Early Prototype", *Proceedings of the 14th National Computer Security Conference (NCSC'96)*, Washington, DC, pp. 167-176, 1991.
- [Snort08] Web Site of Snort intrusion detection system (2008): www.snort.org.
- [Snot07] Snot arbitrary packet generator (2007): www.securityfocus.com/tools/1983.
- [Somayaji98] Anil Somayaji, Steven Hofmeyr and Stephanie Forrest, "Principles of a computer immune system", *Proceedings of the 1997 workshop on New security paradigms*, Langdale, Cumbria, United Kingdom, pp. 75-82, 1998.
- [Sommers04] Joel Sommers, Vinod Yegneswaran, Paul Barford, "A framework for malicious workload generation", *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, Taormina, Sicily, Italy, pp. 82-87, 2004.
- [Staniford-Chen96] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, "GrIDS: A Graph-based Intrusion Detection System for Large Networks", *Proceedings of the 19th National Information Systems Security Conference (NISSC'96)*, Baltimore, Md, USA, pp. 361-370, 1996.
- [Systematic08] Systematic Testing web site: <http://systematic-testing.com>.
- [Tcpdump08] TCPDUMP/LIBPCAP public repository (2008): <http://www.tcpdump.org/>.
- [Tcprewrite08] Tcprewrite: pcap file editor (2008): <http://tcpreplay.synfin.net/trac/wiki/tcprewrite>.
- [Templeton00] J. Steven Templeton and Karl Levitt, "A requires/provides model for computer attacks", *Proceedings of the 2000 workshop on New security paradigms (NSPW '00)*, NY, USA, pp. 31-38, 2000.
- [Thompson97] K. Thompson, G. Miller, R. Wilder, "Wide-area Internet traffic patterns and characteristics", *IEEE Network Transaction*, Vol. 11, pp. 10-23, 1997.
- [Tidwell01] T. Tidwell, R. Larson, K. Fitch and J. Hall, "Modeling Internet Attacks", *Proceeding of the IEEE Workshop on Information Assurance and Security*, West point, NY, USA, pp. 54-59, 2001.
- [TrafficIQ08] Karalon's Traffic-IQ (2008): <http://www.karalon.com/trafficipro.htm>.
- [Trinoo07] Trinoo Analysis (2007): <http://staff.washington.edu/dittrich/misc/trinoo.analysis>.

-
- [Tripwire08] Tripwire (2008): <http://www.tripwire.com/>.
- [Vbox08] VirtualBox: Sun Virtualization System (2008): <http://www.virtualbox.org/>.
- [Vigna00] Giovanni Vigna, Steven Eckmann, Richard Kemmerer, "Attack Languages", *In Proceedings of the IEEE Information Survivability Workshop*, Boston, MA, USA, pp. 163-166, 2000.
- [Vigna04] Giovanni Vigna, William Robertson, Davide Balzarotti, "Testing network-based intrusion detection signatures using mutant exploits", *Proceedings of the 11th ACM conference on Computer and communications security*, Washington DC, pp. 21-30, 2004.
- [Vigna99] Giovanni Vigna, Richard A. Kemmerer, "NetSTAT: A Network-based Intrusion Detection System", *Journal of Computer Security*, Vol. 7, pp. 37-71, 1999.
- [Vmware08] VMware Virtualization products (2008): <http://www.vmware.com/>.
- [Weber98] Weber, Daniel James, "A taxonomy of computer intrusions", *Master Thesis*, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, USA, 1998.
- [Webster] Merriam-Webster Dictionary and Thesaurus online (2008): <http://www.merriam-webster.com>.
- [Whisker08] Whisker: vulnerability scanner (2008): <http://www.wiretrip.net/rfp/>.
- [Zanero07] Stefano Zanero, "Flaws and Frauds in the Evaluation of IDS/IPS Technologies", *Proceedings of the Forum of Incident Response and Security Teams (FIRST)*, Spain, pp. 167-177, June 2007.

