KUNGL
TEKNISKA
HÖGSKOLAN

HØGSKOLEN
I GJØVIK

NISlab
Norwegian Information
Security Laboratory

Using benchmarking to improve IDS configurations

Morten Tvenge

Institutionen för
Data- och Systemvetenskap

Examensarbete 20 poäng
i data- och systemvetenskap
inom magisterprogrammet i informations- och kommunikations säkerhet,
Kungl Tekniska Högskolan

Examensarbete
Nr 2004-x-164
2002

Preface

This MSc thesis is the final project of the Masters course in Information Security at Gjøvik University College (Høgskolen i Gjøvik)

With my background with a bachelor degree in computer technology it felt natural to choose a more technical angeled topic. Since many of my fellow students chose information security from a management perspective, I thought it would be nice to contribute with a more technical thesis.

IDSs have been of interest since I took a general computer security course in my bachelor education for more than two years ago. For me, the challenge was to find the right angle of this thesis, which I have spent a lot of time finding out in the previous fall semester, and into this spring semester. IDS is a broad topic, and narrowing the research field has also been a great challenge.

I have always been fascinated by the Open source / GPL environment, by their efforts and contributions to non-commercial software. Many of them are using their spare time to work with developing software as a free alternative to more or less expensive commercial software. I personally believe that many businesses, especially the smaller ones, can save money using open source software as an alternative. Many does already, as an example Linux distributions, Web servers, Office applications as well as security applications are being used.

After searching in some mailing lists on the Internet, I found out that many questions were like "How can I benchmark my IDS?" and "Is tool X any good when testing an IDS?" This triggered me into studying of IDS and benchmarking. In this thesis I find it interesting to see how the Open source / GPL software performs in an IDS benchmarking setting.

NISlab
Gjøvik University College
Høgskolen i Gjøvik
30.06.04

_____

Morten Tvenge

Abstract

Benchmarking Intrusion Detection Systems (IDS) can be an important way of finding out how your IDS performs. As a watchdog against crackers, malicious code and other potential threats from networks, it is important that you as certain as possible that the IDS works as intended. High rates of false-positives and false-negatives may undermine the reasons for using an IDS. False-positives occupies time and resources when the IDS generates false alerts. Even worse, are the false-negatives, which is all the attacks the IDS fails to detect. These high rates can make it difficult to justify the use of an IDS. These error rates may be a consequence of the IDS architecture and how it is configured.

It is a known fact that configuring an IDS to work properly is difficult, unless you are an expert in the field. Benchmarking has been for experts only, and the price tag is often high. If one cannot afford expensive equipment and hiring consultant services, one is left alone with own skills and experience. Recent work with IDS benchmarking does not reveal any unified approach for novices with limited resources. There is a jungle of software to choose from and there are many pitfalls.

So one approach can be to use a benchmark methodology that can focus on testing the configuration of the IDS. By testing an IDS with different configuration, we can find witch elements in a configuration that are important to reduce the error rates. Using existing and free software based on an adequate methodology, this thesis might be an aid for novices in the work with configuring or benchmark their IDS, with the goal to improve the configuration reducing error-rates and which may lead to improved security response and freeing up resources.

Sammendrag

Benchmarking av Intrusion Detection Systems (IDS) kan være en viktig metode for å finne ut om IDSen fungerer som den skal. Da IDS fungerer som en vaktbikkje mot angrep, skadelig programvare eller andre potensielle trusler fra datanettverkene, er det veldig viktig at IDSen fungerer som den skal. Høye rater av false-positives og false-negatives kan være en medvirkende årsak til argumentasjon mot å bruke IDS. False-positives opptar både tid og resurser fra IT personell når IDSen genererer falske alarmer. False-negatives er enda verre, siden dette er angrep som IDSen ikke klarer å detektere. Slike feilrater gjør det vanskelig å rettferdiggjøre bruken av IDSer. Årsaken til slik slike feilrater, kan ligge i begrensingene til IDS-arkitekturen og i konfigurasjonen.

Å konfigurere en IDS slik at den virker som den skal, kan være vanskelig med mindre en er ekspert på området. Benchmarking er en metode for hvordan man skal teste en IDS, og dens funksjonalitet. Benchmarking av IDSer blir stort sett utført av eksperter med en ganske så stiv prislapp. Hvis en ikke har råd til dyrt utstyr og leie av konsulenttjenester, er en overlatt til seg selv. Arbeid innen IDS benchmarking gir ikke noen entydig metode for IT personell og andre amatører med begensede resurser til å teste IDSer. Det er mye programvare og metodikker å velge mellom, men sannsynligheten for å lykkes er ikke stor. Det er mange feller som fører til dårlig eller lite tilstrekkelige resultater.

Måten å angripe det på kan da være å finne en benchmarking metodikk som kan brukes medfokus på å teste IDSens konfigurasjon. Ved å teste ulike konfigurasjoner av IDSen, kan vi finne hvilken elementer, eller deler av konfigurasjoen som reduserer feilratene. Ved å bruke eksisterende og gratis programvare med grunnlag i en god metodikk, kan denne oppgaven brukes til å hjelpe IT personell og andre med små tilgjengelige resurser, slik at også denne gruppen kan gjenmnomføre en IDS benchmark selv, eller som hjelp til å vise hva som bør vektlegges i en IDS konfigurasjon. Dette vil forhåpentligvis føre til forbedret konfigurasjon, slik at feil-ratene blir redusert, og IT personellet kan fokusere på andre viktige arbeidsoppgaver.

Table of Contents

List of figures

Appendix:

1. Introduction

1.1 Background

IDSs[1] have been around for many years now. From the most basic, the IDSs have evolved to sophisticated security software. Axelsson describes the techniques and taxonomy in [19] and [20].

Commercial enterprises are developing many of the IDSs on the market today. The support services for the IDSs are aiding their customers in the work with configuration. Several companies have specialized themselves in IDS benchmarking, and this can be very useful to validate that the IDS performs according to its specification or in a certification process. IDS benchmarking is also useful after deployment, to ensure it meets the operational requirements on the network. Especially in security critical organisations IDSs can be a part of the efforts made as result of a risk analysis, where the security requirements are higher. Benchmarking can be a time consuming and expensive affair. Not all companies have money to spare to use on such consulting services.

The security requirements varies from business to business, based on type of business and which information it stores and processes. IT staff probably want more budget fundings to increase security from top management, since acquiring information security very often is expensive [22]. When budgets are tight, one have to gain most security for each $. In such situations, commercial IDS and especially consultant services as IDS benchmarking or support services, will probably be a too big expense post, and it may be questionable if the effect are wroth its costs.

As an alternative to commercial IDSs, there are a number of IDS free of charge and based on the following types of licences:

- Open source licences (http://www.opensource.org)
- GPL licences (http://www.gnu.org/copyleft/gpl.html)
- BSD licences (http://www.opensource.org/licenses/bsd-license.php)
- Freeware

Software released and published with either of these licences, are free to use and modify as long as the licence type is followed. Today there are several IDSs, and of course other types of software available under these licences. Open source IDS software usually have no support directly from the vendor, except How-tos, mailinglists or other information sources found on the Internet or other information channels.

Benchmarking of IDSs, have been of great interest since mid 90's. This we can see from the the number of papers within IDS topic. Ideally, benchmarking can aid the work with

1Intrusion Detection Systems

picking the right IDS, but it can also be valuable after deployment to see if it works somewhat as planned or intended.

IDS benchmarking is often executed by consultant services, like NFR security, Neohapsis and many others. Other work with IDS benchmarks are funded for governmental purposes, like the DARPHA project [6], as an example. So if consulting services are to expensive, and other projects are not fully open, one really have no options than doing it self. The job with evaluating IDSs for can be a difficult [1], and if one is no expert in the field, the are so many pitfalls that the result can be most often be less than adequate.

1.2 Motivation

The goal itself, is not that everyone must, or should, benchmark their IDSs. It all depends on what needs and requirement the business has. But if one choose to have an IDS in the network, it ought to work as good as possible. Based on Ranum in [1] and [16], the main challenges for IDSs are:

- False-positives, alerts of an intrusion detection that is not real
- False-negatives, attacks that are not detected by the IDS

How the IDSs deals with this, is anchored in the architecture and how the IDSs is configured. To test or benchmark an IDS, one could just pick a testing software out of the blue, but the result might not be in harmony with the reality. If a benchmarking methodology is used, this helps as a guideline, given that the methodology is adequate, to test the important aspects of an IDS's functionality. Its intention is to increase the quality of the work when we are testing an IDS, so that the results can be as good as possible.

When using an IDS, it is not desirable that the IDS occupy unnecessary resources of the work force, where they use hours of scanning IDS logs and alerts. Most of them probably are fake[2]. To achieve lesser working hours with the IDSs, one must reduce rates of false negatives and false-positives, so that the IDS gives most performance for lesser effort. One approach could be to reduce the IDS sensors alert sensitivity, but this again will increase the risk of false-negatives, which might be much worse than false positives. False positives leads to more work analyzing the IDS alert logs, and occupies time and resources. False negatives on the other hand, is an attack that goes unnoticed. It may not necessary lead to a security break, but reactive countermeasures cannot be taken, of either the IDS or IT staff.

Therefore one must find a way in the middle. If an IDS architecture is already chosen, the IDS configuration may be a suitable place to alter the baseline for false positive and false negative rates. There are, as I am aware of, no publically open IDS benchmark that

2 False positives

are dealing specific with testing different IDS configurations to see how the IDS performs under different configuration. By doing this, we hopefully can pinpoint what elements of an IDS configuration that gives the lowest error-rates. This increases efficiency in resource use, as well as the IDS is doing up to the best the architecture can perform. Configuring an IDS to work with low rates of false-negatives and false-positives is not always easy, and it require some skills and experience [16].

When searching the Internet in the topic of IDS benchmarking, the questions asked on mailing lists, like [31] and others, are quite often "What tool should I use when testing my IDS?", "Is tool X any good?" or "Where can I find relevant literature?", "What testing methodologies shout I use?".These questions arise when one is in the process of testing and evaluating an IDS. It is clear that people responsible for performing an IDS benchmark has too little knowledge and need something to guide them with the steps when benchmarking their IDS. Therefore the motivation of this work, is to help the inexperienced IT staff or system administrators that have deployed their IDS[3], and is not happy with its performance due to high false positive rates, and wants to find out how they can use benchmarking to improve its performance, due to configuration changes. By improving performance, it is in this thesis defined as reducing the number of false positives and enlighten the aspects of false negative rates. Better performing IDSs might reduce work load for IT staff and maybe contribute to a better understanding of the results generated by the IDS.

In a bigger scale, this work might contribute to focussing on a more standardized approach on IDS benchmarking, and also be useful to IDS users on the work with configuring an IDS to perform better.

1.3 Problem description and research questions

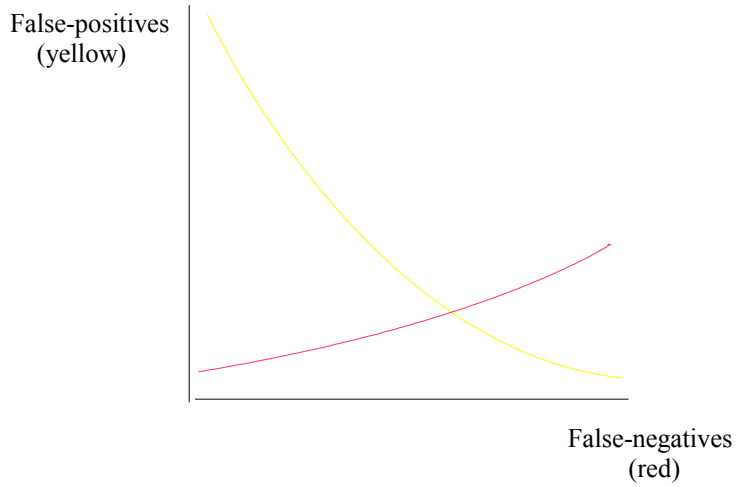
In order to answer the problems raised in this thesis, we have to find out with an existing benchmarking methodology how we can benchmark an IDS. To be able to see improvements in performance, the benchmarking should focussing on testing several configurations so that the results can be compared. The differences in the benchmarking results may be an indicator of how good the changes in the configuration were, and what good effect on performance such change might give.

The changes have to be based on the IDS's reaction to testing tool based on the benchmarking methodology. The number of alarms or alerts raised have to be compared based on which configuration the IDS is configured with. From the alerts we have to find more information whether it is a genuine attack, a false positive, or if it is possible to determine some false negatives. The decisions must be made based on the characteristics of the tools used to benchmark the IDS.

3 Mainly a free of charge IDS, but also cheaper commercial IDSs where support services are too expensive.

The goal of which results that are ideal, is shown in Figure 1.1, where the results should be somewhere near where the false positive rate and false negative rate crosses. The figure is derived from Ranum's figure in [16]. Please note that that only is a model for illustration and understanding of the concept, and is not a exact scientific model. When the configuration is changed by, as an example, detection sensitivity the false positive rate might drop when reducing detection sensitivity. But if this sensitivity is to low, real attacks might be missed, and the false negative rate will increase. The idea of using configuration changes together with benchmark, is just to tune performance closer to where the two rates crosses. The benchmarking results will be a pinpoint to in what direction the results move.

Figure 1.1: The model of error rates in an IDS, when reducing the alert sensitivity and/or increases packet inspection.



With benchmarking of the IDS configurations, it is desired to find some elements of the configuration that can be aiding a configuration of an IDS, and maybe a corner stone for the work with configuration guides.

This reveals three main sections of information it is necessary to find out :

- Available benchmarking methods
- Important configuration test criteria based on the methodology
- Existing penetration tests used on IDSs, i.e. tools used for testing

Of this the following research questions are essential:

- To what extent can benchmarking be used to improve IDS configurations
- To what extent is this approach in IDS benchmarking useful
- Which benchmarking methodologies may be suitable
- What are the advantages and weak sides of the chosen methodology and software

- Based on the methodology, what software can be used
- If we use freely available testing software, are they adequate according to the methodology
- How can the different configurations be classified to ease comparison among other IDSs
- To what extent can IDS benchmarking be used in a IDS configuration guide
- Which elements of the configuration gives better performance for lesser tuning
- How can we trust the results

1.4 Stakeholders and target groups

This thesis is aimed to IT staff and decision makers in smaller businesses that wishes to deploy an open source / GPL / BSD licensed or freeware IDS where it is desirable to find out how the IDS performs, and what can be done to improve its performance.

People that are working with IDS benchmarking or writing tools for security penetration testing may also find this thesis interesting.

1.5 Research method

In order to be able to find answer to the research questions, it is necessary to use one ore more scientific techniques. These will provide methods using existing research and gaining new knowledge with this work. Based on the research questions in section in section 1.3, the methods are selected to how these can be most useful gaining the answers needed. The method is based on information and guidance in [26].

In the this thesis, it will be natural to use an qualitative experiment, a quasi experiment, to find the answers we need. The experiment is used to see if the concept and a way of thinking when performing an IDS benchmarking with the main goal to improve the configuration of an IDS, really is applicable and meaningful. The qualitative approach is due to the fact that work with IDS benchmarking is time consuming. There are not enough time within the limits of this thesis work, to perform enough tests to get the statistical material, as of an quantitative approach. If a quantitative approach had been used, as far as I can see, every single configuration benchmark should have been tested a vast amount of repeatable times to get any statistical material. Since there are many possible configurations and limited time, such approach is not possible within the scope of this thesis. When benchmarking an IDS, a selection of configurations has to be chosen. I theory, the number of possible configurations is enormous, due to many configuration options. Therefore a little selection of possible configurations will be used in the testing to see if it is possible to identify an track changes in the results and find out what configuration that was the reason for this change.

The practical sides experiment is based on setting up an test environment with computers, so that the IDS can be installed and configured on a machine and tested with its various configurations. More details are described in chapter 3.

To get a pinpoint on the reliability of the testing results, as little quantitative experiment will be used. The base for this is using a tool and a configuration and run the test 40 times to see the distribution of alerts an error rates presented by the IDS. This experiment will be used when analyzing to what extent we can trust the results and how repeatable they are.

To find an adequate benchmarking, related work must be found. Citeceer and Google are the primary information sources, supplied with literature from library and own technical books. The idea is find a methodology, that is suitable for the targets:

- Opensource/BSD/Freeware licensed software, both IDS and testing tools
- Not to complex, and preferably suited for IDS or network security devices.
- Not to specific for a certain kind of IDS
- Completely open for public view
- Based on experiences and best practices

To be able to put a benchmark into a real IDS test, software must be used. The software is found mainly by google. Recommendations from previous work will be evaluated, as well as own experiences, and recommendations from mailinglists and forums on the Internet.

To make a complete survey on benchmarking and available tools, is too extensive in this thesis, therefore a brief "dive" in the most common and popular research material will be taken. Of course, better methodologies and especially tools will probably be developed in the future, and this can be enhanced into this work, unless it's radically changed in completely new way to think of IDS benchmarking.

1.6 Delimitations and assumptions

The topic IDS may seem narrow most people. The fact is that it is not. There are Host based IDSs (HIDS), Network based IDSs (NIDS), Network Node IDSs (NNIDS) [1], and there are different approaches in the detecting technologies, like signature based, anomaly based and audit records [8]. In addition, there are many IDSs on the market today, both commercial and of non profit licenses. To cover all these types of IDSs is a tremendous amount, and expensive work. As a graduate student with limited time and resources, it will be a too wide research area. Therefore I have chosen to narrow the research topic to intrusion detection based on network traffic. Due to limited resources, I will concentrate on the freely available NIDSs. Hereby NIDS will also be referred to as IDS, but notice the real difference between these two.

When it comes to the testing tools, there are also limited to open source / BSD/ GPL licences or freeware. This might lead to more complicated testing, with the challenges that follows, since more complete test suites for IDSs are commercial.

The test environment used for testing IDSs and their configurations, are also limited, due to lack of computers and access to network devices as routers, authentic servers or authentic networks.

Due to these limitations, I believe it is important to show that benchmarking can be done with limited resources, by starting on the work with creating a more standardized method for benchmarking IDSs. Limited time might reduce the width and depth in proving that this can be done. Event though if the result details are not complete, the method to produce the results will be provided so that test settings and further results can be reproduced.

In the latest years there has been a lot of discussion whether IDS really are useful, as in [23]. Benchmarking also has weaknesses , but none of these discussions are covered in this thesis because this thesis are intended to address the technical challenges of benchmarking after the decision to deploy an IDS have been taken.

The organisational consequences and aspects of an IDS is not covered in this thesis. Though this might be as well as important as the IDS performance itself, this is not a part of the scope of the thesis. Here we inspect the technical sides of the IDS.

1.7 Thesis layout

Chapter 1 Introduction

General information about the research goals and methods, also background and motivation of the work is presented.

Chapter 2 State of the art in IDS benchmarking

A presentation of the existing research material available. This is used as the theoretical base for
methodologies and techniques. The chapter also describes this work's contribution.

Chapter 3 The experiment

A detailed presentation of the test experiment. This includes choice of methodologies and choice of software. The chapter should describe most aspects of the practical settings of the experiment, along with choices and delimitations. The intention is to show how the work can be reproduced and used in later research.

Chapter 4 Results

This chapter presents the output and results of the experiment The results are commented and discussed, where the reasons for some deviations are explained. Some of the research questions will be answered in this chapter.

Chapter 5 Discussion of the results

Interpretations of the work. Discussion of the results at a higher level, where objectivity and critical angling of the results are important.

Chapter 6 Future work

Due to the broad topic, and thereby delimitations, it is natural that this work leads to further research. The chapter presents what could be done better, and pinpoints subtopics that needs to be further developed.

Chapter 7 Conclusions
Evaluation of the work. Sums up whether the research questions have been answered, and if this works contributes to the IDS research as intended. Some of the more general research questions will also be answered here

Appendix A Test results
The detailed results from the IDS testing. For special interested this appendix might be used the context of understanding the results and presentations in chapter 4.

Appendix B: Reliability test
This is the results and discussion of small experiment describing the reliability of the results.

Appendix C: Technical problems
Review of some technical problems encounted during this work.

2. State of the art in IDS benchmarking

2.1 Available benchmarking methodologies

There are several IDS benchmarking methodologies, [5], [21], [24] and [6] as examples of some previous work. Puketza, Zhang, Chung, Mukherjee and Olsson [5] has developed a methodology for benchmarking IDSs. This has been a reference in many projects in IDS benchmarking. However, this is 8 years old, and is not general enough to be used as methodology. Although, it has some elements of their experience that is useful. It will show an example of IDS testing in real life, not just a method on the paper. Their experience, and the work with the DARPHA off-line Intrusion Detection Evaluation [6] will aid the work with utilizing a methodology into a real-life test. This work is not public, so it will not be used directly as a methodology, but some of their experiences will come in handy. Ranum has written some good papers, [1] and [16], dealing with pitfalls and common errors when benchmarking IDSs, based on experiences in his career with benchmarking IDSs. Axelsson, [17], [19] and [20], has contributed with issues in benchmarking, and with definitions and useful taxonomy. The work of Ranum and Axelsson will be very helpful in avoiding some of the most common errors done in IDS benchmarks, especially when drawing conclusive lines. There are also some other work with benchmarking, but either they are not general enough, or they are written for anomaly intrusion detection, like Maxion and Tan [27] . But the interest of this article lies in the fact that it presents a benchmarking method that hopefully enlightens aspects of benchmarking principles of IDSs

There is a lack of a unified and simpler approach in benchmarking IDSs. Some of the benchmarks are inadequate, or too specialised for a specific setting. If one should directly use more formal methods, the Common Criteria [25] could be an alternative. But after my opinion, these are probably too complex for other than specialized consulting businesses. A methodology that is somewhat in the middle, anchored in good security practice, and has a relatively low learning threshold together with a general approach that suits av wide range of IDS, is preferable. Research on mailinglists on the Internet, like [31], the OSEC NIDS [3] test criteria and OSSTM [2] have been recommended. These will be the methodology base for the work in this thesis. They are further described in chapter 3.

2.2 Software used in the experiment

In order to actually perform a test on an IDS, testing software are being used. From the methodology, we can discover what to test on the IDS, and thereof pick out suitable software to be used. There are a lack of good and complete testing software. Nidsbench [9], is an effort to create such suite. Puketza, Zhang, Chung, Mukherjee and Olsson has also deployed a test suite together with their methodology in 96. But these are not general enough, and is also quite old. Otherwise there are numerous single tools available, that are licensed as mentioned in previous chapter. There are also a lot of commercial tools available, as example IDS informer from Blade Software, but commercial IDSs is not a scope in this thesis. The goal is to find everything we need of software that is free to use. A presentation of the software used in this theisis

2.3 Areas with weak literature coverage

There is a lack of a unified method for testing intrusion detection systems, a method that is commonly accepted as an adequate method, with a set of tools can test all the elements described in the methodology. Many methodologies have been presented, but they are are not directly usable within the goals of the thesis. One problem is that many of the experienced people working with IDS benchmarking use proprietary methods or the more complex formal methods like Common Criteria [25]. When testing an IDS for businesses with low resources, economy is important, and s simple methodology that is understandable for other than experts is necessary. Hopefully the methodology used in the thesis, will have such qualities.

There is a tremendous amount of tools available for security penetration testing. Though some are good, many others are not so good. Many also have the same or partially the same functionality. Without a good methodology for testing, it is hard to find the right software. There is currently a lack of IDS testing suites, so this has to be created according to what software exists and what is needed based on methodology. Some of the software is old. I the worst cases, it probably wont compile on newer operating systems.

Another problem with IDS testing, is that there are detected new security breaches every day, and therefore it is difficult to stay up-to-date with all the latest attack patterns in a penetration test. Therefore it is smarter to concentrate the testing on general types of attack instead of dealing with all the latest specialized attacks. This also gives this work more validity in the future, if the methodology tests if IDS can detect common and general techniques used in most attacks.

2.4 Claimed contributions of this work

Existing literature provides information about methods and principles of how to benchmark and test IDSs. There are several ways of doing this according to previous work. The work with this thesis will provide new information and knowledge in areas that have not been covered before. These areas are basically :

- Based on a methodology, this thesis will present a way to benchmark the different configuration of an IDS. I have not found any IDS benchmarks that explicit concentrates on different IDS configurations.

- This work will hopefully aid IT staff with very limited budgets with benchmarking and maybe improving their IDS configuration, because this work might help the process of pinpointing what elements in an IDS configuration that leads to better performance of the IDS. At best the consequences might be lesser false-positives and false-negatives.

- This work also will be a further step in the process of standardizing benchmarking of IDSs, which, after my experience and many other in the literature's opinion, is almost absent.

3. The Experiment

3.1 Introduction

This chapter describes methods used in the work with testing IDSs. The methods used here are based on related work on IDS benchmarking. Basing this work on previous methods and principles, will hopefully help avoiding some pitfalls and flaws done in previous IDS testing. As an red line for this, Marcus J. Ranums work in [1].

This chapter will also cover details on the test setup. This includes hardware configuration of network and computers used in the test scenario. Which operating system and what services running on them will also be discussed. Choice of software, both IDS and attacker/testing software will be described. Based on the method and research questions it will also be presented what elements in an IDS to test. All this is essential to create a test scenario that gives as good results as possible, and so that the results can be reproduced in later experiments.

3.2 Brief introduction to the experiment

The main goal of this experiment, is to see how one can use benchmarking to improve an IDS configuration. To do this, first a suitable benchmarking methodology has to be chosen. Then a computer network has to be established as a test environment. At least three roles has to be possessed; an attacker, a honeypot and an IDS.

Based on a methodology, software has to be picked out. It should be open source / GPL / BSD licensed or freeware. The software should cover the aspects of an IDS testing presented in the methodology, as good as possible. The IDS used should also be based on free software. The software tools should run from the attacker machine. The honeypot will be installed with several services like web, ftp, ssh, etc. to simulate a host that contains something an attacker wants or wants to abuse. The IDS should be set up with a configuration X. Then then attacker runs the software tools, and we register how many attacks the IDS detects. Thereafter, we make some changes to the IDS configuration and run the software tools once more, and register the results. This procedure have to be done several times. At the end we have IDS benchmark results from many different configurations. Now we can look after variations in the results. If we see positive effects in results from one configuration compared to another, this might indicate that we have found a parameter in the configuration that might be a improving factor of the configuration.

3.3 Benchmarking methods

3.3.1. Why using a benchmarking methodology

Ranum's experiences in IDS benchmarking, described in [1], proves that it is easy and very common to create poorly designed benchmarks. They often suffer from inadequate validity, i.e. the results from what we are measuring or testing is not really what we should measure. As an example, lets say we are measuring the throughput of an IDS. In this case it would be natural to have a load generator that simulates network traffic. From the test results we get a throughput of some Mb/s. But what if the IDS throw away packets if it is under heavy load? If this is not dealed with in the test, we probably are not measuring what we hoped for, and therefore might be fooled to believe that the IDS has greater throughput than it really has. This is one trivial pitfall out of many that may lead to insufficient and results that might be a bad representation of reality.

In the literature there are a lot of benchmarking methodologies. So which one should be used? The choice relies on either to develop a new methodology that is tailor made for the IDS that is to be tested, or an existing benchmarking methodology can be used. The methodology might not be 100% compatible for every needs, so some tuning will probably have to be done. In this thesis I have chosen to base the work on existing methodologies, so that the focus can be more concentrated on the testing. By using an existing benchmark methodology, previous work and experiences can be utilized through these methodologies. The point would be to choose a methodology that is widely used, and should be based on known "best practices" and industry standards, e.g. ISO or similar.

I have chosen the benchmarking methodology based on 2 methodologies:

1. OSSTM 2.1 (Open Source Security Testing Methodology)
2. OSEC (Open Security Evaluation Criteria)

These are mainly chosen because they are open for everyone, not only to use, but also to contribution and experiences. As a parallel to open-source and GPL based software, communities can be a part of evolving the products in a hopefully positive direction, as described in [4]. In addition, some important elements in IDS testing from [15], will also be used in creating an adapted methodology for this thesis.

3.3.2. OSSTM 2.1

The Open Source Security Testing Methodology [2] is the result of the work of ISECOM (Institute for Security and Open Methodologies), managed by director Pete Herzog. ISECOM is the initiative taker for assembling this security benchmark methodology for benchmarking security products, as firewalls, IDSs and many other

security and network devices. Herzog pinpoints that although the test sheets in this manual [2], are not revolutionary, it is as a whole, a benchmark for the security testing profession. The goal of this manual is to be an wide accepted benchmarking methodology for security testing. The advantages of using OSSTM are many, but the most appealing are its foundation in best practices from ISO 7799/ISO 17799, NIST, MITRE, SET and many others. In addition it is founded on several legislation acts in information security and personal data protection from western countries like USA, Germany, UK, Australia and Spain. See [2] for the complete list. Even though this broad foundation on best practices and legislation, it is not a guarantee for success. Therefore one must treat this methodology as a guideline. One other strong advantage of using this methodology is that even though ISECOM[4] is responsible for updating the manual, everyone can access and use the methodology. Everyone can contribute on this methodology, either as constructive criticism, or added contribution of own experiences. By this, the methodology can evolve to something better on everyones experience in the information security profession.

3.3.3. OSEC

OSEC, Open Security Evaluation Criteria [3], is a similar project of open security evaluation. This is very similar to OSSTM, described above, but is concentrated on security products, mainly NIDS and firewalls. Its initiative taker is Neohapsis (http://www.neohapsis.com), a company with years of experience in security consulting and product testing. OSEC is driven by the lack of standardized methods in evaluating security products. OSEC is based on the company's years of experience. But the most important is that OSEC is open for all to use and review. As OSSTM OSEC evolves by constructive criticism and contributions of other companies, end-users and security communities. In this thesis, OSECs NIDS testing criteria will aid, together with the IDS section of OSSTM to give a benchmark based on best practices and others experience in IDS testing. The OSEC criteria can also be used in certification processes of network equipment. The criteria is structured in a checklist of what to examine of the IDS. The checklist is quite detailed. What software to use is not mentioned, except of some few tests requiring some specific tools. The tools mentioned suits the license goals of this thesis. The OSEC criteria are more detailed, and will be used more directly than OSSTM. OSSTM also has a list of important factors, but is not so detailed.

4http://www.isecom.org

3.4 Test environment

3.4.1. Delimitations

Due to my limited resources, a test network as OSEC and OSSTM describes in [2] and [3], cannot be achieved. With the resources available, I have 3 computer, a hub and an unfiltered Internet connection. Despite that Ranum in [1] claims that a successful IDS testing relies on large investments in infrastructure and time, this is not something I can achieve with my resources. So I have to do the best out of the equipment that I have access to.

Most IDS tests are concerned of the speed and throughput performance of the IDS, according to Ranum in [16]. This means the traffic capacity of the IDS, which are most important for NIDS because they are based on analyzing network traffic. But in this thesis, speed will not be focused on, due to that this might not be an important issue in smaller businesses where the flow of data traffic is statistically lesser than in bigger networks. In bigger networks commercial IDSs are more likely used, and capacity is of greater importance in such environment. In addition, the machine hosting the IDS in this experiment, is fairly slow, and this will most certainly be a bottleneck in a speed/capacity test. But it *is* interesting to see how the IDSs performs when there are limited computational power available.

In this thesis usability and user-friendliness are not a important factor and goal for the testing. It is sure that usability and user-friendliness is important for end users, e.g. system administrator and decision makers, in order to get a right response to an intrusion alert. This is mentioned by S. Axelsson in [17]. The reason for not focusing on these elements is that I chosen to concentrate on dealing with the detection capabilities of the IDS sensor.

3.4.2. Network properties

The network used in this test environment, consists of :

- One separated and unfiltered Internet connection with its own subnet, switched Ethernet connection.
- A “3com OfficeConnect Dual Speed Hub 8” used when not connected to the Internet

According to the methodologies mentioned in [1], [2], [3], [5] and [6], background traffic is an important element when testing IDSs. Therefore In the test setting, the IDS should be tested with:

- No background traffic to easier determine false negatives

- Generated background traffic to check for false positives in addition to the traffic from the software tools
- True network traffic connected to the Internet

It is not very likely that hosts are not protected by any firewalls, but in this test, it will be interesting to see how the test results may vary according to the type of background traffic.

3.4.3. Hardware

The equipment available to me in the testbed is the following hardware:

2 stationary computers with the following hardware:

- Intel Pentium II 266 MHz
- 64 MB RAM
- 4 GB SCSI hard drive
- CD-ROM
- Network Interface Card

1 laptop with following hardware:

- Intel Celeron 400 MHz
- 192 MB RAM
- 20 GB 4200 RPM IDE hard drive
- CD-ROM
- PCMCIA Network Interface Card

These computers are certainly not state-of-the-art, and probably not what smaller businesses uses[5], but these are the only available to me for now. I will also be interesting to see how these relatively slow machines perform during the tests.

3.4.4. Software

3.4.4.1. About the choices

In this thesis, I have chosen to use open-source / GPL / BSD licensed software or freeware. One of the reasons for this, is my own limited resources and the target group is smaller businesses that have a tight economical budget, that cannot afford expensive

5or anyone at all

investments in information security and especially here, IDSs.

Many of the smaller businesses have very few employees in IT staff, due to limited budgets. In addition, the skilled people are often engaged in larger companies, where salaries are higher due to larger budgets and better economy.
Therefore I choose to install common software from the open-source community that it is most likely to choose, both because of no buying costs and the availability of the software. The availability is here meant as how easy it is to get hold on the software, as example by using the common search engines on the Internet.

3.4.4.2. Operating system

Fedora Core release 1[6], previously known as the workstation edition of Redhat Linux. This operating system will be used on all 3 computers. According to Distrowatch [7], Fedora is the most popular Linux distribution the last year. This is not a scientific survey, more a hit counter for the distributions web pages. There are no recent scientific surveys on this at the moment that are of recent date. But anyhow it gives a pinpoint, and backs up the idea that Fedora included former Red Hat, is one of the most popular, if not the most popular, Linux distribution in the spring of 2004. In addition former Red Hat and now Fedora is widely supported and compatible by software vendors. This make Fedora a reasonable choice in operating system.

3.4.4.3. IDS software

The choice of IDS software relies on one of the principles in this thesis, to use open source/GPL/BSD licensed software or freeware. This also include the IDS software itself. There are many types of IDS software to choose from, but with the commercial ones ruled out, the amount is getting quite smaller. The most commonly used IDSs available will be the most interesting to uses, since the results from this work might be of greater interest. But in addition, it also would be interesting to see how a so called simple and trivial IDS will perform in the testing. This might tell us if it is needed to use the most complex and advanced IDS on the market, or if it is sufficient with a more simple and more easy managed IDS.

There are also several types of IDS detection techniques, according to Stallings in [8]. Audit records, statistical anomaly detection, rule based detection and distributed intrusion. In this thesis I have limited the detection techniques to mainly cover rule based detection, also known as signature based. By doing this, I can get a small and well arranged group of test candidates. I prefer to test each IDS candidate more thoroughly if the time allows, instead of a more shallow test of several more IDSs. Time is limited, so for this thesis I will try to cover at least 1 IDSs, hopefully more.

6Codename Yarrow

Therefore I have chosen following IDS:

1. Snort
   (http://www.snort.org)

IDS evaluated as possible candidates but not tested. These are candidates of further work based on this thesis:

1. Prelude Hybrid IDS
   (http://www.prelude-ids.org)
2. BRO, A system for detecting network intruders in real-time,
   (http://www.icir.org/vern/bro-info.html)
3. Libnids based IDS
   (http://libnids.sourceforge.net/)
4. Firestorm NIDS,
   (http://www.scaramanga.co.uk/firestorm/)
5. Shoki (NIDS),
   (http://shoki.sourceforge.net/)
6. Tamandua NID (NIDS),
   (http://tamandua.axur.org/)

3.4.4.4. IDS testing tools

There are some IDS test suites on the market, but they are commercial. To follow the open-source/GPL license principles of the goals of this thesis, one must choose among the tools within this kind of license, or at least as freeware. There is no available package of tools that tests all aspects of an IDS, as far as I have discovered. Song and Undy [9] confirms that there are a lack of such tools. Fortunately there are many stand-alone tools that can test one or more attributes of the IDS. Therefore we have to use the methodology to find out which element of the IDS we shall test, and which test tools that can match and test the current property. Based on the methodology we can build up our own test suite of separately available software. The choice of tools will be presented in chapter 3.5.2

3.4.4.5. Running services

To attract attackers, at least one host has to run interesting services[7]. Ideally the services should be run on different hosts. This is probably the most common in real life based on speed and availability of the services. This is especially the case of larger companies with quite heavy traffic load. Though some smaller businesses might have several services running on a machine, due to small budgets. It is not unlikely to imagine that

7 This means something that in one or another way can be abused.

some might have a dedicated machine as “server” to host all of their needed services. So even though the scenarios created here are after the methodology standards, they might be representable for some small businesses, which are one of the target group for this thesis.

The choice of services to run on one of the machines are picked by terms of widely use, and of course the hardware limitations on the machine. Since the machines are running Fedora Core Linux, it feels natural to choose the services that follows as default with the distribution. One of the reasons for that, is that some might think it is most convenient and lesser work having the Linux' installation procedure set up the services automatically. Instead of downloading the most recent versions of the desired services, and compile, install and configure them manually, which more skillful and experienced system administrators would preferably have done. Services coming along with the Linux distributions are often not up-to-date, regarding latest version numbers and security patches. Therefore these can be more vulnerable, since there might have been discovered flaws or security holes in the software. It will be interesting to see if outdated software can affect any parts of the test.

Which type of specific services that should run on the host, are based on what can be typical in an smaller business combined with what services that comes along with Fedora Linux:

- Apache web server with PHP
- MySQL
- Vsftp FTP server
- SSH remote shell
- Sendmail
- Because of limited hardware resources, the TWM window manager are used instead of the default Gnome desktop. This preserves some memory and cpu time.

3.4.5. Logical Test Network

Due to limitation in hardware, I cannot use the reference network layout described in [2] and [3]. Therefore it cannot have multiple subnets, routers or DMZs. In this case one must limit the test to take place internally in a DMZ or subnet only.

There are 3 computers used in the scenarios. One attacker, which will launch attacks on the victim. The victim is a kind of honeypot, where all interesting services are running. This machine have something that the attacker in some or other way wants.

The IDS machine passively monitors traffic on the network. Since NIDS are the test target, the network interface card must be in promiscuous mode to access the network traffic. The IDS machine will normally be secured. According to Ptacek and Newsham [10], IDSs 2 biggest threats are:

1. Insufficient information from the packets the IDS can access. This means that the IDS may not see the same packets as the victim, due to fragmentation of packets, attackers evasion techniques or different packet acception/rejection policies on the IDS and the victim or packet loss. Other problems can be timeouts, lag, encryption/VPN and physical network properties, such as different subnets or switched networks.

2. Denial of service[8]. The host running IDS is vulnerable to DoS attacks, since launching such attacks may put the IDS out of function. If an attacker wants to evade the IDS, he would probably launch a DoS or Distributed DoS attack against the IDS.

So how is this dealed with in this test? Real-life is complicated, and the things mentioned in 1. above will be too extensive to cover in this thesis. Therefore the test setting must be limited as follows:

- No use on encryption of traffic, e.g. VPN.
- Use of signature-based IDSs. Fluctuation caused by network lag, differences in packet acception/rejections, timeouts or packet loss must be further discussed in the results

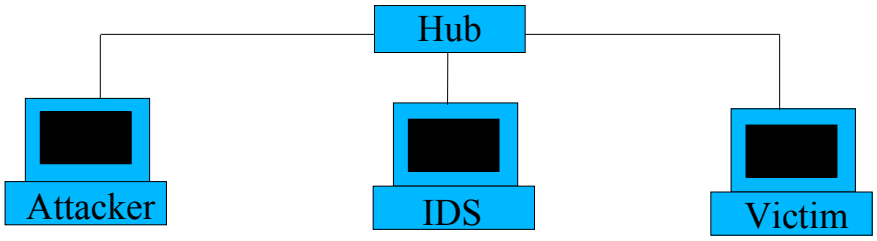
To avoid or reduce the probability to be attacked of a DoS, there are some countermeasures one can take. First, the the interface where the IDS is capturing packets, there could be no IP address assigned to the interface. This eliminates IP based attacks from other networks, since routers work on the IP layer, layer 3 in the OSI model [11], and therefore packets cannot be sent to an interface with no IP address. But on the local subnet, the attacker can use the MAC address of the interface card instead of the IP address to launch a DoS attack. Therefore this method is not 100% bulletproof. Another method is to cut the Tx (transmit) wire on the interface card, or the wire. This makes it physical possible for the machine to give any leads that it is online, since it only can receive data. Unless the attacker know that it is online and its MAC address, the machine hosting the IDS will most likely not be a subject for IDS, since it cannot give away MAC address or online status by arp-requests [12]. IDSs may also use an administrative interface, an own communication channel physically separated from the production network with its own NIC and cables. This is described in the OSEC methodology [3]. This is not used in this test setting, since there is a lack of computers and infrastructure.

In this test setting, there will be no use of modified NICs or wires, because it will be interesting to see how the IDSs reacts on DoS. It will not have an IP address so that it cannot be attacked directly from the Internet, when connected.

The 3 different test scenarios are described in the following figures. The 3 different test situation seen as a whole will hopefully tell us how much the environment has to say for the test results.

8DoS

Figure 3.1: Test scenario with no Internet connection



As shown in Figure 3.1, there are no Internet connection available. This can simulate that the attacks comes for the inside by an insider on the local network, or from the outside through a hijacked host. The victim will be attacked by the attackers software suite described later. The IDS should passively monitor an trigger on the attacks. Having no background traffic possible false-negatives can be detected, generally by subtracting number of detected attacks from the total number of deployed attacks. This is also described in section B in the OSEC NIDSv1 criteria.

Figure 3.2: Test scenario with a traffic generator source

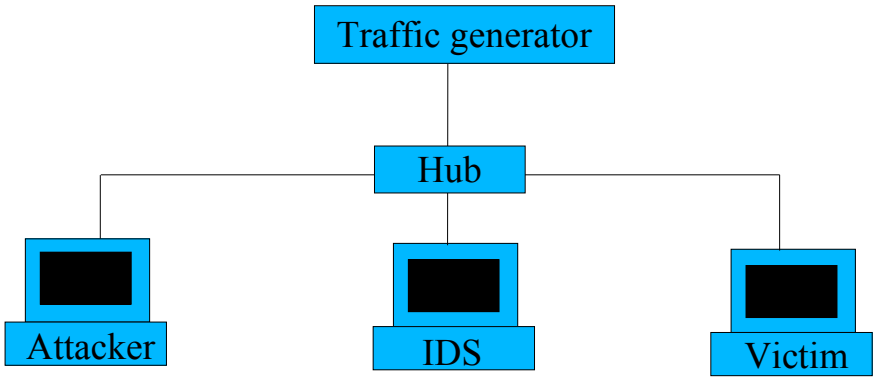


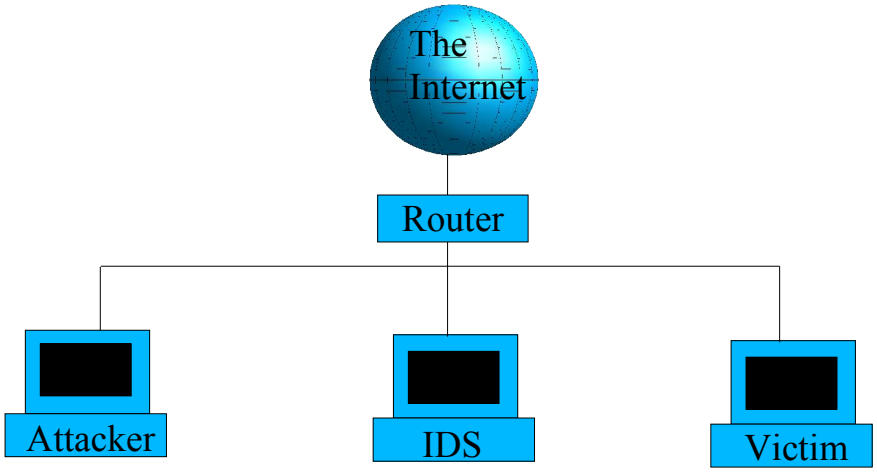
Figure 3.2 is is almost the same as Figure 3.1 except from the traffic generator. Due to the methodology followed, there should be a source of background traffic so that the false-positive rate can be determined. According to Ranum [1], a successful traffic generator should be based on the real traffic in the zone the IDS should operate. This includes packet capturing over a larger time period to statistically determine which type of traffic that is closest to real. TCPreplay[9] can achieve this. In this thesis such work will be too extensive, so I have to use a lesser "perfect" method, and that is to use existing software in load/traffic generators. This must then simulate traffic based on the services running on the victim. Since there are only 3 machines available, the traffic generator must run on one of the hosts. Ideally, there should be a dedicated host doing this, but in this situation the attacker have to do this. Because of this, Figure 3.2 differs a bit from the real test situation, but it is done this way to better show that a traffic generator is the

9 TCPreplay is found at http://sourceforge.net/projects/tcpreplay/

difference from the test situation in figure 3.1.

In this situation we can determine false-positives. The generated traffic may trigger alarms that really isn't any attacks. In addition to the background traffic, real attacks will be deployed from the attacker. This mixture of background traffic combined with real attacks tests the IDSs ability to differ traffic from real attacks. The results can be compared with the results from figure 3.1, and then we can discover what semi-real traffic affects the IDS ability of detecting real attacks. The principle of having background traffic is used in OSEC NIDSv1 section C.

Figure 3.3: Test scenario with real Internet connection



In this last test situation, see Figure 3.3, the subnet is connected to the Internet through an unfiltered connection. In this last setting, the IDS will try to detect the attacks based on real Internet traffic. This is a step closer to a real environment for the IDS. Though it is not always common to operate without a firewall, and the placement of the IDS may vary, see [13]. Since this network are a simulated DMZ, the traffic would normally be filtered so that the only traffic allowed in, would be addressed to the services running on the victim. But allowing all types of traffic from outside the Internet, the IDS can be utterly more stressed by real traffic.

3.4.6. Last notes

For those not reading chapter 4, 5, 6 or 7 i must be underlined that there was no time to test all 3 scenarios as presented in 3.4.5. Only the first scenario were tested, but the two other are still presented here to demonstrate the original thinking and purpose behind

these scenarios and the experiment. The test of these must the be left as future work, see chapter 6.

It must also be mentioned that the dedicated IDS machine broke down in the early stages of the IDS test. Therefore the victim host[10] had to act as both honeypot and IDS at the same time. The consequences are further discussed in chapter 4. Though the figures have not been altered, due to the argumentation above about leaving the initial experiment description intact for later use.

10 The honeypot

3.5 Methodology and research questions

3.5.1. Why a methodology and IDS test

In this thesis it is necessary to use the benchmarking methodology and the software tools as guidance to answer the research questions of this thesis. It might look like this thesis is just another IDS test. This is just somewhat true. An IDS test is the experiment that produces the results needed to answer this thesis' research questions. It is necessary to adapt the methodology in [2] and [3], and partially [15] so that the information we are interested in, emerge from the results.

3.5.2. Relating research questions to the methodology

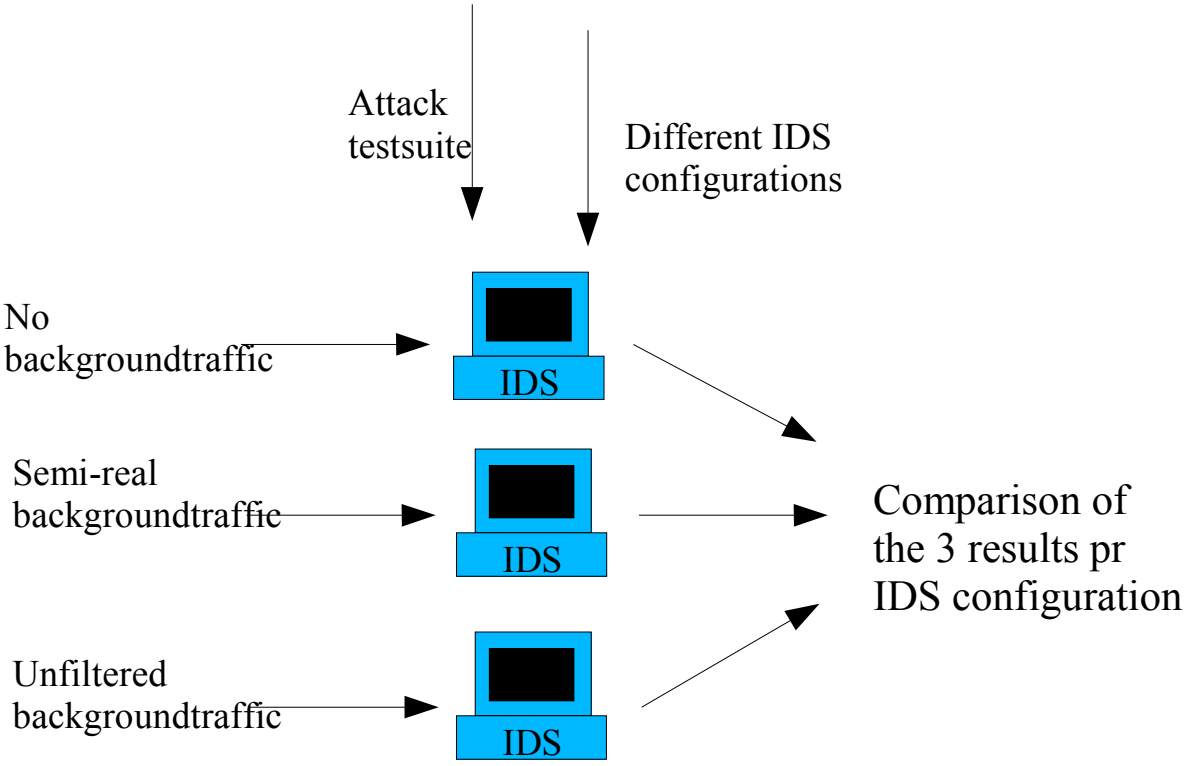
3.5.2.1. Differences from other IDS tests

The thing that differs with this thesis' IDS testing methodology, is that it is the IDS *configuration* that is the most interesting test subject. The other tests from the literature are based on testing an IDS with some configuration. It is very rare, from what I have found, that the configuration is somewhat mentioned in a benchmark methodology. In [14] and [15], the configuration is mentioned as important in use of alerts to system administrators and the false-positives challenges.

As shown in Figure 3.4, we use a specific configuration scheme[11] during a test of an IDS. This configuration will be tested in the 3 different environments described in 3.3.5. After the testing has been completed with all environment, a new configuration scheme will be used to test the same IDS. According to this thesis' goals, several configuration schemes will be tested. Each scheme will have variations in the IDS configuration for the other schemes. As far as it is possible, a configuration scheme should be used among the different types of IDS, specifically NIDS, participating in the testbed. The scheme can than be used when comparing two NIDS of different brands as long as they are compared on the same configuration scheme.

11A specific IDS configuration based on some guidelines of what elements of an IDS configuration that should be "activated" or tuned. This concept has been developed for this IDS testing, and is presented in chapter 4

Figure 3.4 : A test scenario where the IDS is tested with several configuration schemes



3.5.2.2. Analyzing the results

In the literature there are very little information on how to analyze the results from an IDS test. Most articles and papers are dealing with methodology and the aspects of this, so that the test itself is in the center. How to analyze the results is almost not covered at all. But after all, it does not have to be any known method to do this, as long as one understand the results one are dealing with, and draw conclusions based on the quality of the methodology. Ranum [1] argues that bad conclusions can be drawn, if one isn't well aware of the limitations of the benchmarking methodology. If one are aware of flaws in the methodology or software one are using, and the results are good, one cannot conclude that the IDS performs as well as the test might say. If flaws are obvious, this has to be in the conclusion, so the that it not should mislead any readers, that might not have knowledge to detect such weaknesses in the test.

3.5.2.3. Extract new knowledge from the results

In order to answer the research questions, the information to do this, are found in the results for the IDS tests. The first thing to analyze, is the results from one IDS. Since each test of an IDS consists of running the test suite minimum 1 time per configuration,

so there will be output from the software in the test suite for each configuration, in three environments. All test results from one scenario are analyzed, and then we can tell which configuration that brings the smallest rate of false-positives and false-negatives. Then the results from the three scenarios is compared, so that we can see if background traffic, load generated or real, has any effect on the results. Comparing results among different IDSs, correlations in the configurations will imply important areas in IDS configurations.

3.5.2.4. Challenges determining false positives and false negatives

The IDS used in this experiment uses the term 'alert', when something suspicious is detected, refer to the user manual [28] for more details. So how should this be interpreted? Is an alert a detected attack only? It is obvious that an alert is an event that the IDS believes is suspicious, based on its signature database and configuration. An alert can certainly be an attack, but it is also important to have in mind that one attack, does not necessarily generate just 1 alert. An attack can be base on using several tools and exploits with traffic that generates many alerts. Before an attacker decides to attack, he might be probing for running host and services, and might use a vulnerability scanner as well as an port scanner, before he might use an exploit to break in utilizing a vulnerability. Al this traffic may result in several alerts. It is therefore difficult to say exactly what portions of the traffic should be classified as the attack, and if any, what should be false positives. Maybe the IDS did not detect the exploit, and should then this be characterized as a false negative? There are no straight answers to these problems, but one way to solve it, is that the security policy should be a central guidance document for these issues. This document should define what traffic is classified as legal and accepted, and what is not accepted. This might help us when trying to figure out what a port scan should be classified as. If it is legal and detected by the IDS, then it is a false positive. If it is illegal and detected, then it is registered as an attack. If it is bypassed by the IDS we have a false negative. But the final problem still remain, if a port scan generates 20 alerts and port scanning is defined as accepted, do we have 1 or 20 false positives? This is hard to tell, and is very context specific. Therefore I choose not to deal with the problems directly. In the results the goal is to detect the attacks[12], but reduce the number of alerts to a minimum at each attack.

From this, it might be tempting to play with the thought of a decision system above the IDS, which compares the alert with security policy and other information about the network[13].

12 Unless it is an false positive generator

13 A similar project has been worked on at NISlab.

3.6 Test criteria

3.6.1.What to focus on

When testing the IDSs, several tools must be used in order to test all elements of the IDS configuration. What to test, is anchored in the methodology. So the next question is, how shall we test? Nidsbench [9] is a attack suite for testing NIDS, though it does not completely suit OSSTM [2] or OSEC [3], some of the software used can also be used in the testing here. There also other test suites available, e.g. IDS Informer[14], but these are commercial and is not applicable for the scope and goals of this thesis.

Therefore a list of software to be used, must be created out from the test criteria in [2] and [3]. As we can read in the media that new security issues are detected on a daily basis. Tools that exploits these security issues, are spread at an enormously speed across networks connected to the Internet. Therefore signature based IDSs must be updated very often. If we make an optimistic assumption:

- If the IDSs can detect every attack in its signature database, then it is unnecessary to test every attack of recent date, since the IDS have shown that it detects all attacks of which it has been developed a signature for.

The point of such assumption is that basing the test suite just on the recent attacks, makes this IDS benchmark useless in the future, because all these attacks will be encountered for in software patches. So if we use the methodology and test suite in this thesis and only tested the most "popular" attacks at this moment (spring 2004), this thesis would, after my opinion, be of little interest both for public and scientific point of view.

Therefore I choose to use more general attack tools, instead of just focusing on the most recent exploits, that tests the IDS for common techniques used in attacks, as well as methods for evading the IDS. Del Carlo has written about this in [18]. Examples of this can be:

- Obfuscation – where data is manipulated
- Fragmentation – where the attack is spread over several packets and in different order
- Encryption – where the attack is encrypted
- Denial of Service (DoS) – where the attacker tries to put the IDS out of function

By generalizing the attack suite, we can create a test suite that is valid, not just for now, but also in the future until new and totally different attack types evolves. The benchmark methodology used in this thesis are designed for this principle. Though as an example on custom exploits attacks, an old and known attack, and a new attack, preferably unknown to the IDS, should be tested on the IDS to see how the IDS deals with these.

14 http://www.bladesoft.net

3.6.2.The software suite used in the test

In this chapter the software used to test the IDSs are presented. A brief description of what they do and where they can be found is explained. It is important to remember that all software used are either freeware, BSD or GPL licensed or open source. No commercial products are used.

It must be emphasized that software used for a specific type of IDS, also could be used testing other IDSs. As a describing example, some packet- and traffic generator tools are designed to use a Snort signature database as a source for generating traffic, these *same* rules must also be used when testing other IDSs.

3.6.3. Test suite

3.6.3.1. Introduction

The following sections describes a proposal for an test suite that can be used based on the methodology used. Each software has it brief description and where it can be obtained. In addition, it is described in which part of the methodology the software is anchored, i.e. which property of the IDS it should test. Note that this is a proposal of a test suite based on the most common software found on the Internet using Google. This might not be the most best or ideal test suite, but its intention is to show an example of how this can be done. It is possible that there exist better tools, or that some new tools have been made public, but that has to be a topic for further work with this project. All tools used, are found during February 2004.

3.6.3.2. Nmap

Software number	S 1
Description	Portscanner that reveals listening ports on a remote host.
Location	http://www.insecure.org/nmap/
Anchored in methodology	OSSTM 2.1 Module 9.15, 9.17 OSEC NIDSv1 A.1

3.6.3.3. Nessus

Software number	S 2
Description	Portscanner and vulnerabilityscanner
Location	http://www.nessus.org
Anchored in methodology	OSSTM 2.1 Module 9.15, 9.17 OSEC NIDSv1 A.1, A.2, A.4

3.6.3.4. Snot

Software number	S 3
Description	IDS evasion tool, a packet generator that triggers snort alerts taking a snort rules file as input.
Location	http://www.stolenshoes.net/sniph/index.html
Anchored in methodology	OSSTM 2.1 Module 9.3, 9.4, 9.5, 9.6, 9.9, 9.10, 9.11, 9.12 and 9.14, OSEC NIDSv1 C.1, C.3-C.10, D.1, D.2-D.6, D.7-D.11

3.6.3.5. Sneeze

Software number	S 4
Description	Packet generator that triggers snort alerts taking a snort rules file as input. Works like Snot, for generating false positives
Location	http://snort.sourceforge.net/sneeze-1.0.tar
Anchored in methodology	OSSTM 2.1 Module 9.5, 9.6, 9.9, 9.10, 9.11, 9.12 and 9.14, OSEC NIDSv1 C.1, C.3-C.10, D.1, D.2-D.6, D.7-D.11

3.6.3.6. Nikto

Software number	S 5
Description	A web server scanner that performs various test using the HTTP protocol and CGI scripts. Has techniques for evading/fooling IDSs.
Location	http://www.cirt.net/code/nikto.shtml
Anchored in methodology	OSSTM 2.1 Module 9.6, 9.8, 9.10, 9.11, 9.12, 9.14 OSEC NIDSv1 E.2-E.8, F.17-F.19, F.20-F.33, G.2

3.6.3.7. Fragroute

Software number	S 6
Description	Intercepts, modifies and rewrite egress traffic payload
Location	http://monkey.org/~dugsong/fragroute/
Anchored in methodology	OSSTM 2.1 Module 9.7, 9.8 OSEC NIDSv1 D.1, D.2-D.6, D.7-D.11

3.6.3.8. Fragrouter

Software number	S 7
Description	Tests the IDSs ability to deal with fragmentation of traffic
Location	http://packetstorm.widexs.nl/UNIX/IDS/nidsbench/fragrouter.html
Anchored in methodology	OSSTM 2.1 Module 9.9, 9.12 OSEC NIDSv1 D.12, F.1-F.16, F.17-F.19

3.6.3.9. IDSwakeup

Software number	S 8
Description	Simulates false well-known attacks to detect false positives
Location	http://www.hsc.fr/ressources/outils/idswakeup/index.html.en
Anchored in methodology	OSSTM 2.1 Module 9.9, 9.10, 9.11, 9.14 OSEC NIDSv1 C.1, C.3-C.10, D.1, D.2-D.6, D.7-D.11

3.6.3.10. ADMmutate

Software number	S 9
Description	Tool for making buffer-overflow exploits polymorphic shellcode
Location	http://www.ktwo.ca/security.html
Anchored in methodology	OSSTM 2.1 Module 9.6, 9.13 OSEC NIDSv1 B.1, B.2

3.6.3.11. Apache 1.3.x – 2.0.48 remote users disclosure exploit

Software number	S 10
Description	Exploit using a vulnerability in the Apache web server
Location	http://www.k-otik.net/exploits/12.06.m00-apache-w00t.c.php
Anchored in methodology	OSSTM 2.1 Module 9.6, 9.13 OSEC NIDSv1 B.1, B.2

3.6.3.12. Apache 1.3.X Remote Exploit

Software number	S 11
Description	Exploit using an old vulnerability in the Apache webserver
Location	http://archives.neohapsis.com/archives/bugtraq/2002-06/att-0238/01-apache-scalp.c
Anchored in methodology	OSSTM 2.1 Module 9.6, 9.13 OSEC NIDSv1 B.1

3.6.3.13. 7plagues

Software number	S 12
Description	Denial of Service (DoS) tool testing the TCP/IP stack stability on a remote host
Location	http://packetstorm.linuxsecurity.com/DoS/7plagues.pl
Anchored in methodology	OSSTM 2.1 Module 9.5, 9.14 OSEC NIDSv1 A.4-A.8, C.2, D.1, D.2-D.6, D.7-D.11

3.6.3.14. ISIC

Software number	S 13
Description	IP/TCP/UDP/ICMP stack stability testing tool (stress test),
Location	http://www.packetfactory.net/Projects/ISIC/
Anchored in methodology	OSSTM 2.1 Module 9.5, 9.14 OSEC NIDSv1 A.4-A.8, C.2, D.1, D.2-D.6, D.7-D.11. D.12

3.6.3.15.Network Traffic Generator

Software number	T 1
Description	A tool for creating background traffic
Location	http://freshmeat.net/projects/trafficgenerator/
Anchored in methodology	OSSTM 2.1 Module 9.14 OSEC NIDSv1: As background traffic

4. Results and analysis

4.1 Introduction

This chapter presents the results with following analysis from the IDS benchmarking. The results are based on the test experiment described in chapter 3. Further presentation of results, will be divided based on the test scenarios. The results from the IDS and its configurations, according to the results from the testing software, will be grouped together under each scenario. The test results are stored in details in tables in Appendix A. In this chapter, the results will be displayed in graphs, due to increased readability. Reading lots of numbers from a table is probably not a good way of presenting the results, and it might be difficult to understand and draw any conclusions. The use of graphs gives a better picture of the results by seeing results relatively to each others. If more details of the tests are desired, Appendix A holds this information and is useful as a reference to the graphs. Before presenting the results, the set of configuration schemes used in the testing will be presented and explained.

4.2 Configuration schemes

4.2.1. Introduction

This sub chapter describes a proposal for configuration schemes that can be used in a testing procedure. Such schemes is a concept intended as a cross IDS configuration pattern where the goal is to find some common elements in the configuration that is common for most IDSs. With this, we can give a more reasonably way to compare test results of configuration benchmarking among different IDSs. This is not claimed to be the most ideal way to do it, and it may not suit every IDS or environment, but further research may improve this.

The configuration schemes are based on Snort configuration [28], but are generalized in a manner that should suit most NIDSs, since the schemes are based on general IDS techniques. The next chapters describes the configuration schemes used.

4.2.2. Configuration scheme 1: Default configuration

This scheme is the default configuration provided from the manufacturer. The configuration is usually stored in one or several files created by the installation procedure. In this scheme nothing of the configuration file(s) should be altered except settings for where and how output should be presented, or other specific changes

necessary in order to get the IDS to work. If this implies in changing elements that directly or indirectly affects performance and benchmarking results, this must be documented.

4.2.3. Configuration scheme 2: All signatures

In this scheme, every signatures available should be enabled, if not as default. This is to be sure that the sensor uses all its detection capabilities.

4.2.3.1. Customized signatures enabled

One can also use only signatures suited for traffic on the network according to security policy. From this the vendors signatures can be used, but it also opens for custom made signatures, if these are available to use. This is especially useful in a DMZ or on an internal network, that is protected with a firewall.

This option has not been tested and is not to find in any of the results, but this is an example that this could be a single standalone configuration scheme. Adapting the IDS to its operating environments have shown to be a good idea, and therefor this section is added to underline the importance of testing the IDS in its operating environments. Refer to Ranum [3].

4.2.4. Configuration scheme 3: No signatures enabled

Check the capabilities of the sensor if no signatures is enabled, if this is possible within the IDS. This is useful to see if the IDS can report anything, if it should, by mistake disabled the signatures.

4.2.5. Configuration scheme 4: IP fragmentation

This is the sensors ability to detect fragmented attacks relies on this function. Useful parameters here can be to change timeout of the IP fragments. By increasing this, attacks spread on many IP fragments and with longer delays might be detected. The drawback is higher memory usage.

4.2.6. Configuration scheme 4: TCP stream reassembly

This is the sensors ability to detect attacks in a TCP session. As for IP fragmentation, timeout and memory concerns are the same. In addition, other parameters as:

- IDS evasion detection
- TCP state errors
- Scan detection
- Stateful inspection
- Limit inspection on ports, incoming or outgoing

There can be many options to focus on here, but further details may not be general enough.

4.2.7. Summary

When benchmarking an IDS configuration, 4.2.2 - 4.2.6 sections are presenting elements of an general IDS configuration where each scheme has a set of parameters that can be altered either binary[15] or as value[16]. The idea is to use configuration scheme as a reference point, where the other alters the configuration in on area or section.

As an simplified example, lets say we are benchmarking IDS X with a false positive generator, tool Y. First we use Y against the IDS, where X is configured according to configuration scheme 1. A fictive result of, lets say 1000 alerts. Then we might want to reduce this number. If we are not sure what exactly is the reason for the "high" number of alerts (false positives), we can use, lets say configuration scheme 5, and tweak parameters like enabling IDS evasion detection or increasing timeouts. When we now run Y once more[17]. This time the number of alerts might have dropped to 600. This might imply that by altering the configuration based on the scheme, have reduced the number of false positives, and thereby improved the configuration on these kind of attacks. Alternatively, the number of alerts could have risen, and that implies we have moved in the wrong direction in the configuration.

The example is a very simplified, and a real situation will have to combine several attacking tools and have to utilize the schemes in a combined manner, with several tweaks of the parameters in each scheme. Comprehensive tests have to be done, with several different configurations and results among them must be compared to be able to draw any conclusions of what configuration is the best achievable. It is also worth mentioning that some elements of an configuration might perform well against one type of attacks, but might have undesirable consequences. As an example, if we reduces the TCP reassembly timeout, see 4.2.5, the number of alerts (false positives) may decrease. But as a result, some attacks using long delays between the TCP packets, the NIDS might throw the TCP session and miss the attack. Thereby increasing the number of false negatives. This demonstrates how hard it is to find the "golden middle-way". Many test runs are necessary, and it is an economical issue of how long and thorough these tests should go.

15 on | off, 1 | 0
16 Numbers or strings
17 Of course Y must use the same criteria or parameter for each run

The configuration schemes are useful in a more structured approach on altering an IDS configuration for benchmarking purposes. It is also an effort to find common elements in most NIDSs so that the benchmarking results may easier be compared, in the settings where this makes sense.

4.3 NIDS benchmarking candidates

4.3.1. Snort

Snort[18] is the most known and most common open source IDS available. I think it is a good IDS to demonstrate the use of the benchmarking methodology and testing tools, so that the research questions can be answered. Together with Snort, ACID [29] is used for analyzing the results.

4.4 Scenario 1: No background traffic

4.4.1. Configuration of Snort

4.4.1.1. General settings

Snort was configure to send its output to a MySQL database. The reason for this is that ACID depends on database access of the results. The analysis tool ACID was used to make it easier to read the results from Snort. We have to assume this tool does not change any information from Snort.

The http inspection setting in the Snort configuration had to be disabled in order to get Snort to accept the configuration file. Several attempt were tried combined with intensive google'ing on the Internet, but was unable to make it accept a unicode.map character set file used with when inspecting traffic to Microsoft's IIS servers. This settings was, strangely enough, accepted on the original dedicated IDS machine using configuration scheme 1*.

4.4.1.2. Configuration scheme 1

Snort was here configured according to configuration scheme 1, described in 4.2.2, with the default configuration.

18 http://www.snort.org

4.4.1.3. Configuration scheme 2

All rules files in Snorts signature files were included in the configuration, using all available signatures.

4.4.1.4. Configuration scheme 3

All rules were disabled, and no signatures used in the Snort configuration.

4.4.1.5. Configuration scheme 4

In the IP fragmentation section in the Snort configuration, preprocessor frag2 the following parameters were changed :

- Default session timeout increased from 60 seconds to 120
- Default fragmentation buffer, memcap, increased from 4 MB to 8 MB

4.4.1.6. Configuration scheme 5

In the TCP stream reassembly section in the Snort configuration, preprocessor stream4, the following parameters were changed:

- The parameter for detecting stealth port scans were enabled, sending detect_scans as parameter to the preprocessor.
- Default session timeout increased from 30 seconds to 120.
- Default reassembly buffer, memcap, increased from 8 MB to 12 MB

4.4.2. Test results with configuration scheme 1*

The test results presented is preformed on a different machine than the results in 4.4.3-4.4.7. The machine used here was the original machine intended to host the IDS. It is an Intel Celeron 400 MHz, described in 3.3.3. The configuration of Snort using configuration scheme 1, is the same as in 4.4.1.2, with the exception that the http inspection worked on this machine. The reason for using an asterisk[19] in when referring to the configuration scheme, is to help making no mistake mixing up the results from

19 *

the to different machines, since the preferences for the results are different, based on different hardware and a slight difference in the configuration.

Since there are only one result with this computer, there are no point of showing them alone i diagrams. The idea is to compare the results from several different configurations, and this is unfeasible with only results from one configuration available. Instead it is more interesting to compare result with this machine and and the other machine used in the test. By this, we can see which impact hardware changes have on the testing results. The comparison charts will be presented after the other results in section 4.4.5.

4.4.3. Change of hardware hosting the IDS

4.4.3.1. Background for hardware change

Due to hardware failure in the beginning of the test with configuration scheme 2, this is the only test results that could be retrieved with use of that computer. With no extra money for spare parts[20], the machine acting as honeypot had to act as IDS as well.

4.4.3.2. Consequences for further testing and results

With the need of porting the IDS onto the machine running services, this implies some direct consequences for further testing.

- System services and the IDS service must now run on the same host, sharing even less resources than the machine previously used. This is especially problematic since when launching a tool from the other machine, the service tested will consume cpu time, as well as the IDS. This will affect the results by increased packet drop rate, especially with use of traffic intensive tools. This leads to less accurate results, since many of the packets dropped can contain attacks. E.g. a launched DoS attack against the web service, may render the IDS useless.

- Since further testing suffers from increased packet drops, the drop rate must be measured as well as number of alerts. As a consequence, some of the tests were expanded. Therefore not all results exist in the initial first test compared to the other.

- The IDS will be more exposed, since we cannot hide behind no IP address assignation. The IDS have to be on the same interface[21] as the service on the honeypot. A secondary NIC could have been used as an alternative, but this option was not available at the time.

20 Spare parts for laptops are expensive!
21 Network Interface Card (NIC)

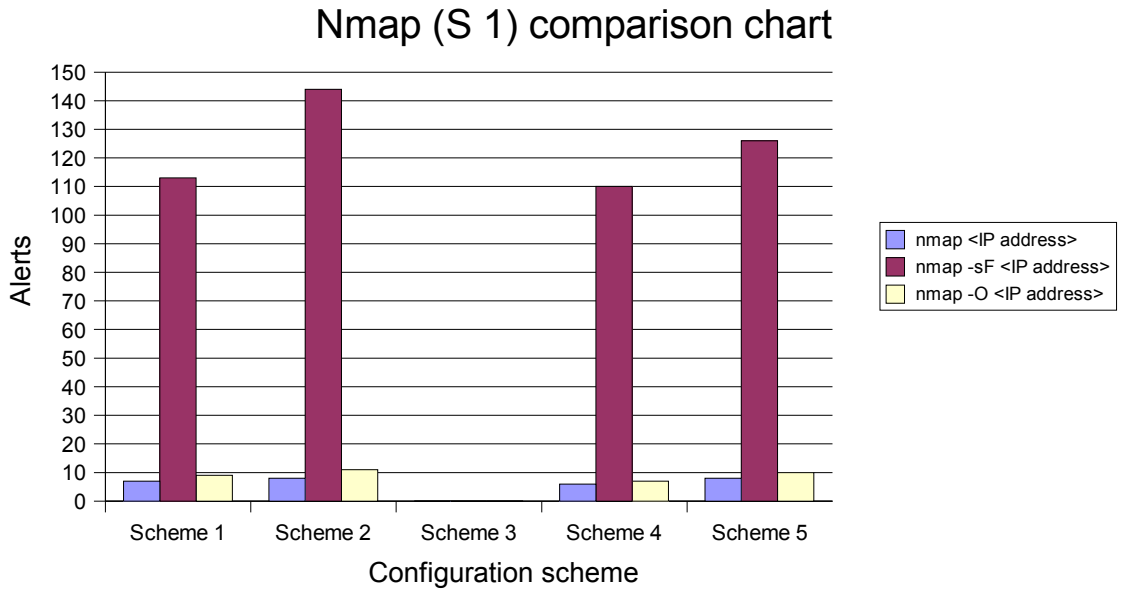
4.4.4. Presentation of results running IDS on honeypot

4.4.4.1. Introduction

The results will be comparison chart were the IDS testing results are displayed with data from configuration scheme 1 to 5, grouped after which software used in the test. By this, variations in the number of alerts generated from the IDS could be compared, and we cans see how these numbers changes based on change of IDS configuration. In addition, each chart of result will be followed by a chart displaying the drop rate from the tests. When reading the graphs it is important to have in mind what the different configuration schemes are. Refer to section 4.2.

4.4.4.2. Results with Nmap (S 1)

Figure 4.1: Comparison chart of the results using Nmap



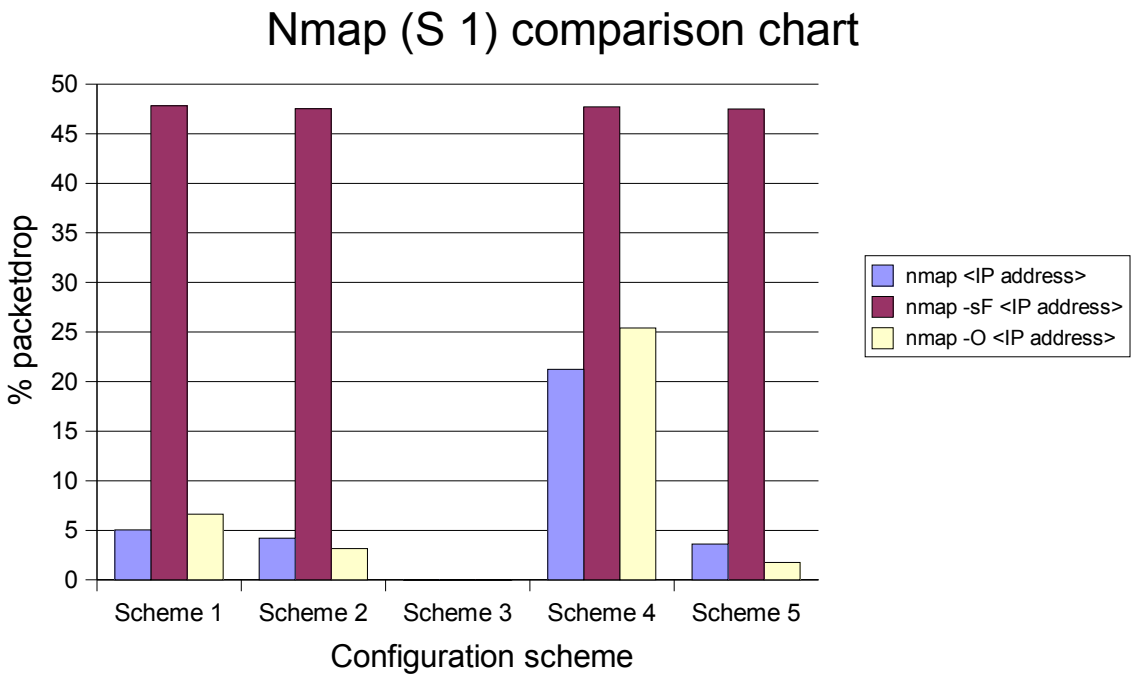
From Figure 4.1 we see that the TCP FIN portscan triggers most alerts. Using no signatures no alerts is raised. Enabling all signatures in scheme 2 seems to increase the number of alerts. Increasing IP fragmentation buffer and session timeout seems to slightly reduce the number of alerts compared to default configuration in scheme 1. In scheme 5 TCP reassembly buffer and session timeout actually increases the alerts compared to scheme 1. Note that scheme 5 uses default IP fragmentation parameters. To see any rough lines here, one can only say that increasing buffers in timeouts in IP fragmentation may reduce the alerts compared to a default configuration. Combining increases in buffer and timeouts in both TCP reassembly and IP fragmentation have not been done in this IDS test, but results could be interesting to evaluate in another test

situation to see if this combination might give better results.

The problem with port scans, is that they generate some alerts. How to deal with these, are a security policy issue. The security policy document should define to what extent port scanning is accepted. If port scans should would be legal, one could define it as portscan activity and make the IDS ignore it. In that case these alerts would be false-positives and thereby reduced or eliminated. But port scans may also be used by attackers to gain information about network and services, in a early step of an attack. Thereby one can lose the preventive effect, and we might have false-negatives. It is a challenge when port scans are not allowed, because of the amount of port scans that are not any part of an attack. It could just be someone doing it for fun or for testing purposes. This would then lead to many false-positives. This illustrates that configuring IDSs is difficult, and must be anchored in security policy. It is hard to tell whether one should characterize such alerts as false positives or false negatives, since a port scan can be a part of an attack or just innocent traffic. A solution might be that the IDS should check later traffic to see if something more suspicious would occur. But then again one might lose the ability to be preventive against the attacker.

In Figure 4.2 we see the packet drop rate from the nmap test. We see that with the -sF parameter the drop rate increases significantly. This is because of increased traffic. Since the host running the IDS has very limited resources, the IDS has to throw away more packets. We see a slightly decrease in drop rate by enabling all signatures in scheme 2, and this might be the cause of increased alerts using that configuration scheme. Scheme 4 has much higher drop rate than the others, and this is probably the why scheme 4 has the lowest alert level due to the high drop rate. This demonstrates that it is easy to be fooled by the numbers. Especially with slow hardware, drop rate must be considered to reduce the possibility of making the wrong decisions.

Figure 4.2: *Comparison chart of the drop rate using Nmap*



4.4.4.3. Results with Nessus (S 2)

In Figure 4.3 we see some of the same characteristics as for nmap. Nessus uses port scan techniques like nmap, so the discussion in the previous section is also valid here. When using safe checks we see that the alert level is quite the same, with the exception of scheme 3 where no signatures are enabled. Increasing buffers and session timeouts seem to reduce the alert level slightly. When even the harmful attacks are enabled[22], we see a rise in number of alerts, especially in scheme 2. It is interesting to see that the IDS used, Snort, gave a small number of alerts. This is the detection architecture that reports of erroneous packets, used by the Nessus' probes. If one, by mistake, has disabled all signatures this shows that the IDS in a limited manner, is able to report events even though it is a signature based IDS. This can be regarded as positive feature, but the usefulness might be left for discussion. Nessus has a large number of test and it is very hard to do any statistics of what is false negatives or false positives. The alerts is useful to indicate that something is happening, but the number of alerts should preferably be reduced

22 All plugins in the Figure 4.3 and 4.4

Figure 4.3: Comparison chart of the results using Nessus

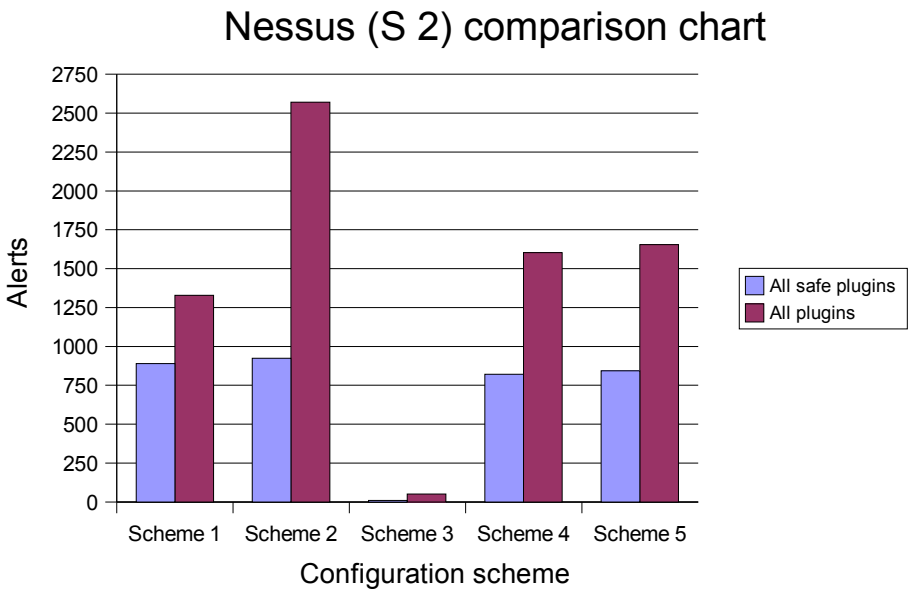
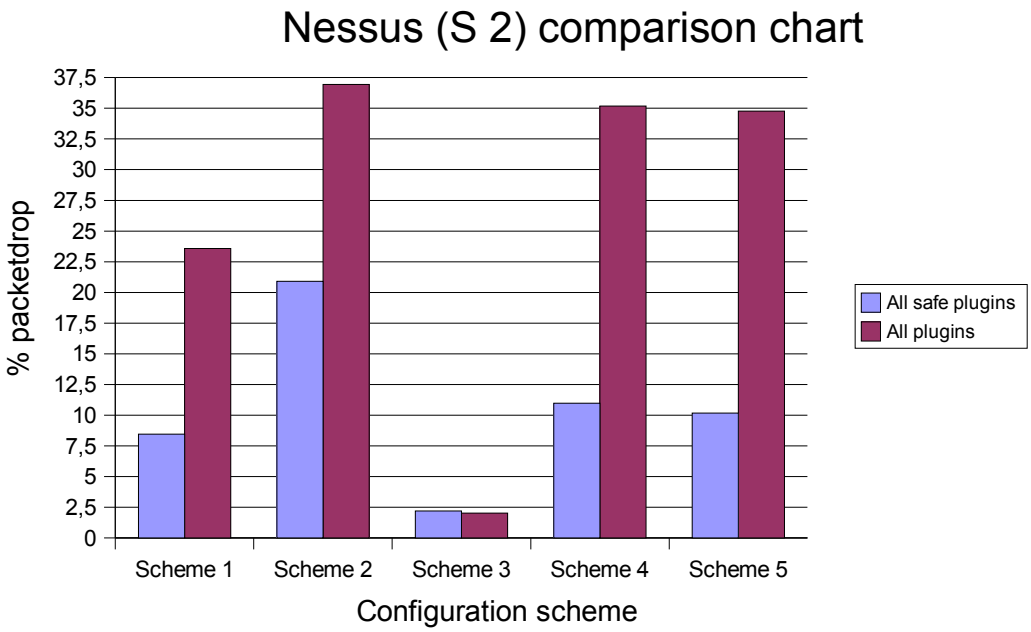


Figure 4.4 shows that the drop rate reflects the alert level. When all attacks are enabled we see an increase in drop rate, which indicates increased traffic. This we see especially where all signatures are enabled (scheme 2) and increased buffers and timeouts in scheme 3 and 4. The reason is most likely because those schemes need more computational power and memory

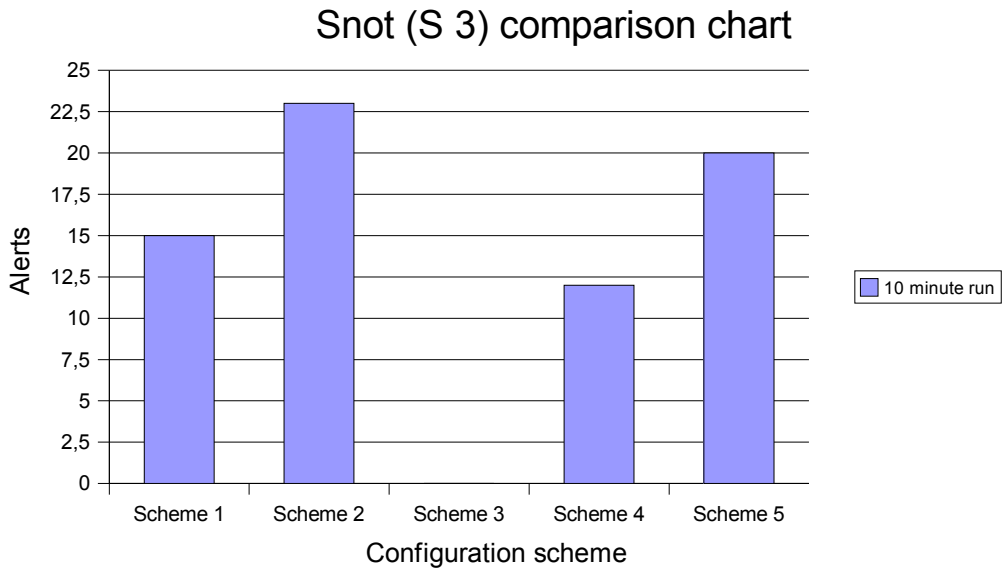
Figure 4.4: Comparison chart of the drop rate using Nessus



.

4.4.4.4. Results with Snot (S 3)

Figure 4.5: Comparison chart of the results using Snot

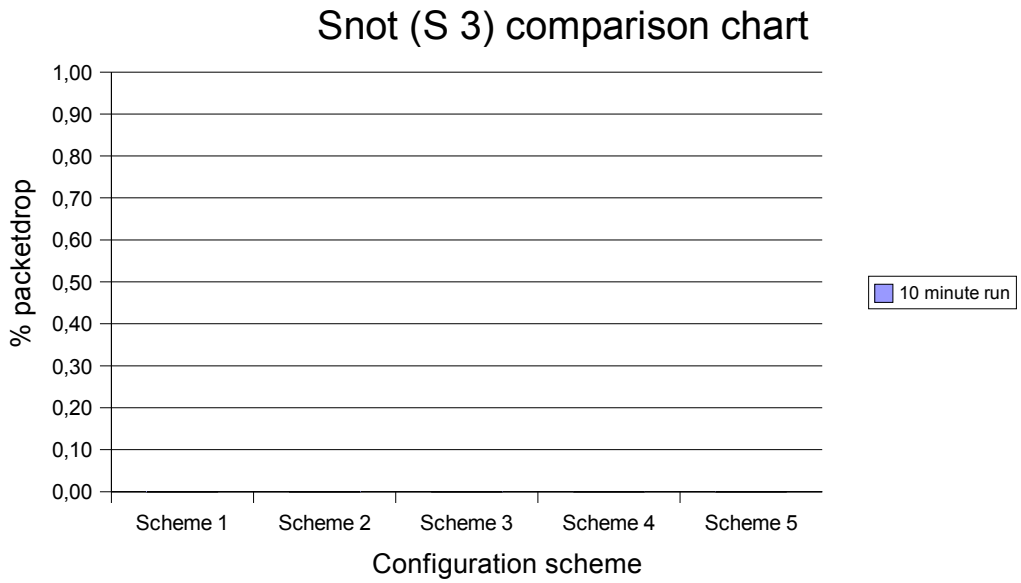


Snot uses the Snort signature database to launch a series of fake attacks. Snot was used with randomly up to 5 second delay between each attack, so therefor it is not guaranteed that the number of attacks was the same for each run. Therefore the results might not be

representable. But still we see, in Figure 4.5, the same trend as the previous results. Scheme 4 gives lowest number of attacks. Increasing TCP reassembly buffers and timeout seems not to bee the solution. Enabling all signatures naturally also increases the detection engine's ability to detect more of the attack. These attacks are characterized as false positives, since the Snot is called a “false positives generator”. In scheme 3 it is natural, that no attacks are detected, since Snort creates traffic based on the signature database.

In Figure 4.6 it is surprisingly to see that there are no packet drops. The explanation is due to the delays between the attacks, probably gives the IDS time to process all traffic.

Figure 4.6: Comparison chart of the drop rate using Snot



4.4.4.5. Results with Nikto (S 5)

This tool examines web services, and has the ability to use IDS evasion techniques. The results are shown in Figure 4.7. Using no IDS evasion, it seems scheme 2 is slightly better than scheme 1 and 4. Using the evasion techniques, the results have some variations among the schemes, but seeing all tests together in the “big picture”, again scheme 4 seems to have on average lower alert rate than the rest, except for scheme 3. In general the results implies roughly the same trend we have seen before.

Figure 4.7: Comparison chart of the results using Nikto

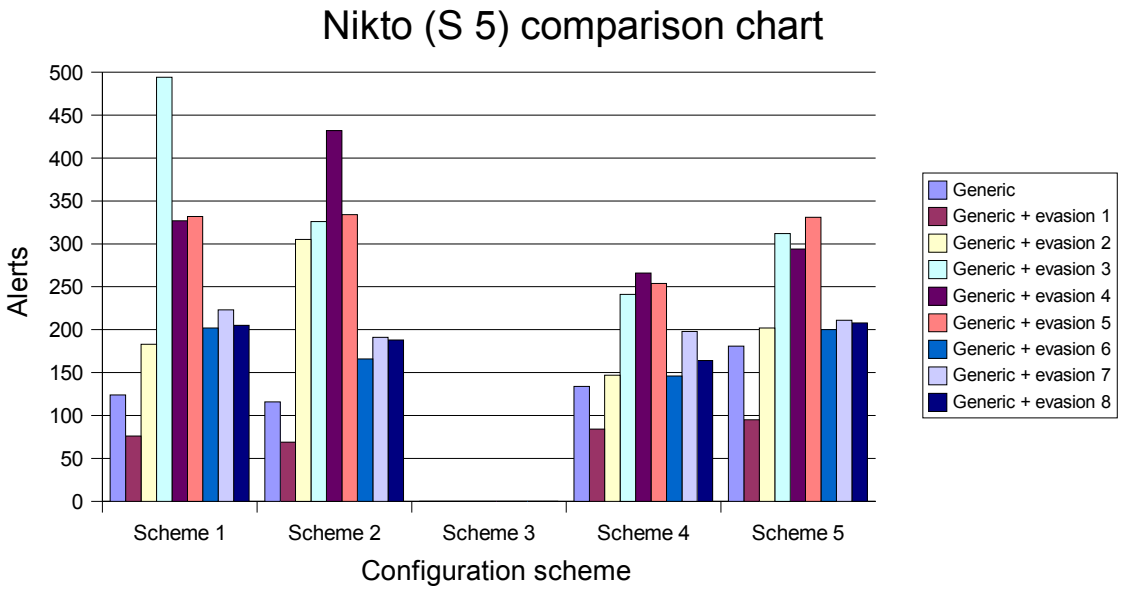
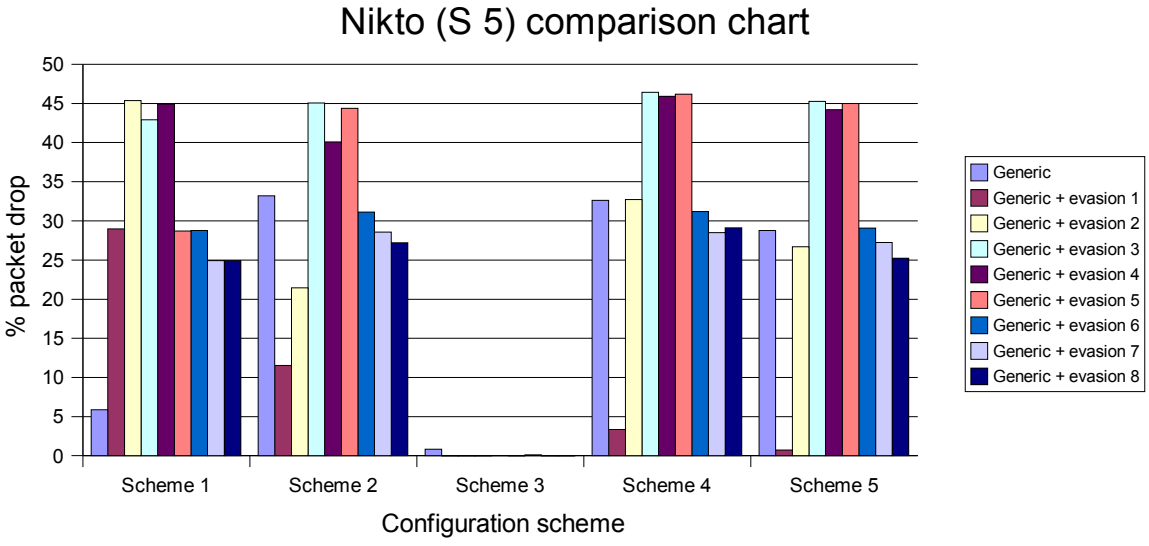


Figure 4.8: Comparison chart of the drop rate using Nikto



The drop rates in Figure 4.8 show that this test generates some traffi, but it is interesting that some test have low drop rate with default or all signatures enabled, but higher drop rates when increasing buffers and session timeouts. This are most likely due to computational power. But there are also tests that is opposite, where the drop rate falls. It is not clear why. Even with no signature used, we see here that even scheme 3 suffers

from packet drop. This is very small values, and could also be caused by suddenly high cpu usage of the host running the IDS.

4.4.4.6. Results with Fragroute (S 6)

Fragroute rewrites traffic from other tools, and the results with Snot are shown i Figure 4.9. We see that the number of alerts are significantly higher than Snot without Fragroute. Here it is interesting to see that using configuration from scheme 3, also generates alerts. This is because Fragroute generates new traffic with errors in TCP headers an IP header. These are detected by Snort even without any signatures. It is also worth noticing that scheme 4 now has higher amounts of alerts than both scheme 1 and 5. Again, scheme 2 is in its own class, like with the previous use of snort.

Figure 4.9: Comparison chart of the results using Fragroute with Snot

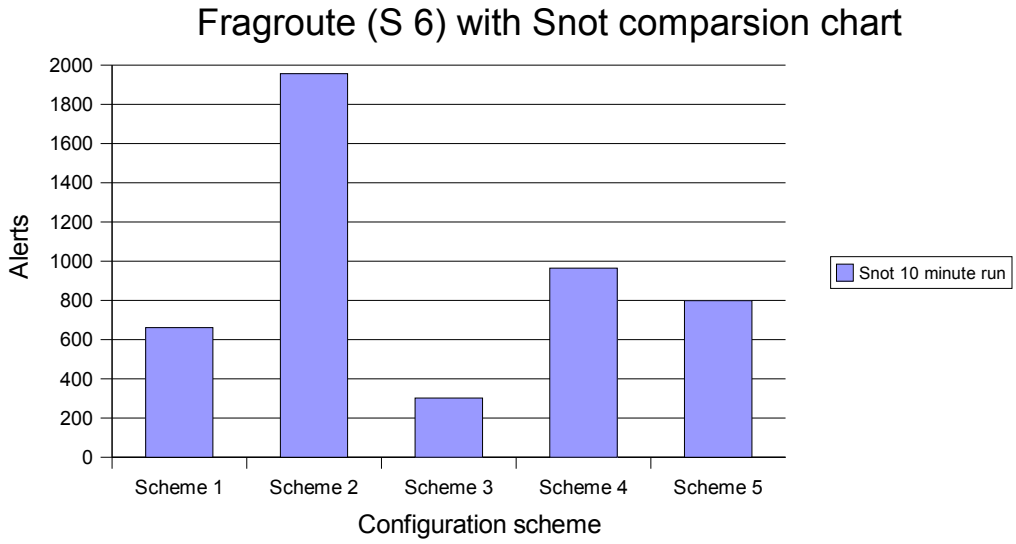
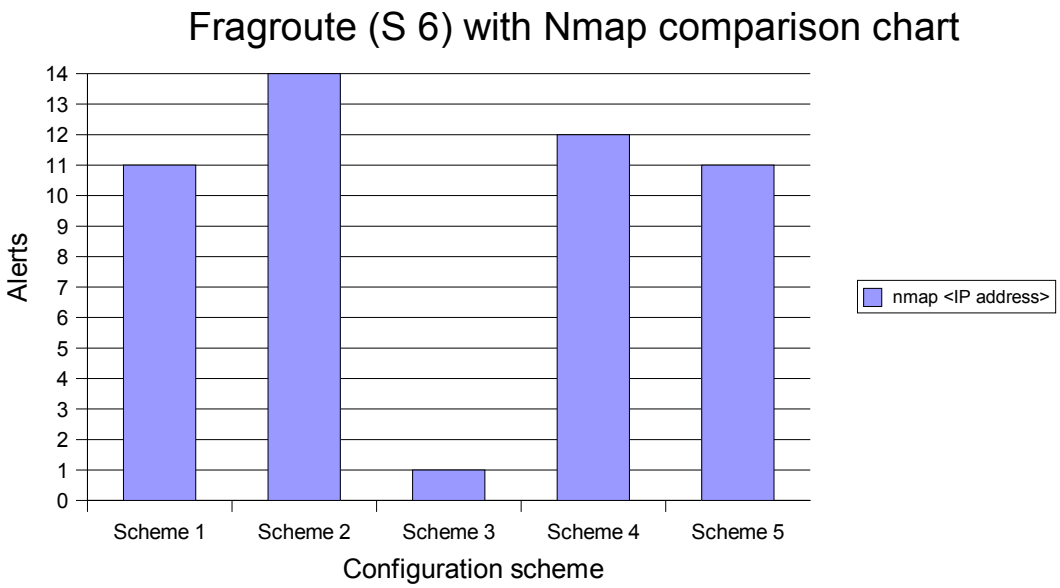


Figure 4.10: Comparison chart of the results using Fragroute with Nmap



In Figure 4.10, we see Fragroute using nmap traffic. The trend is like the previous chart, only somewhat more leveled. Again, scheme 3 generates alerts, due to header errors.

Using Fragroute with Nikto traffic, the pattern repeats it selves in Figure 4.11, though even more leveled, with the exception of that scheme 4 here generates slightly less alerts than scheme 5.

Figure 4.11: Comparison chart of the results using Fragroute with Nikto

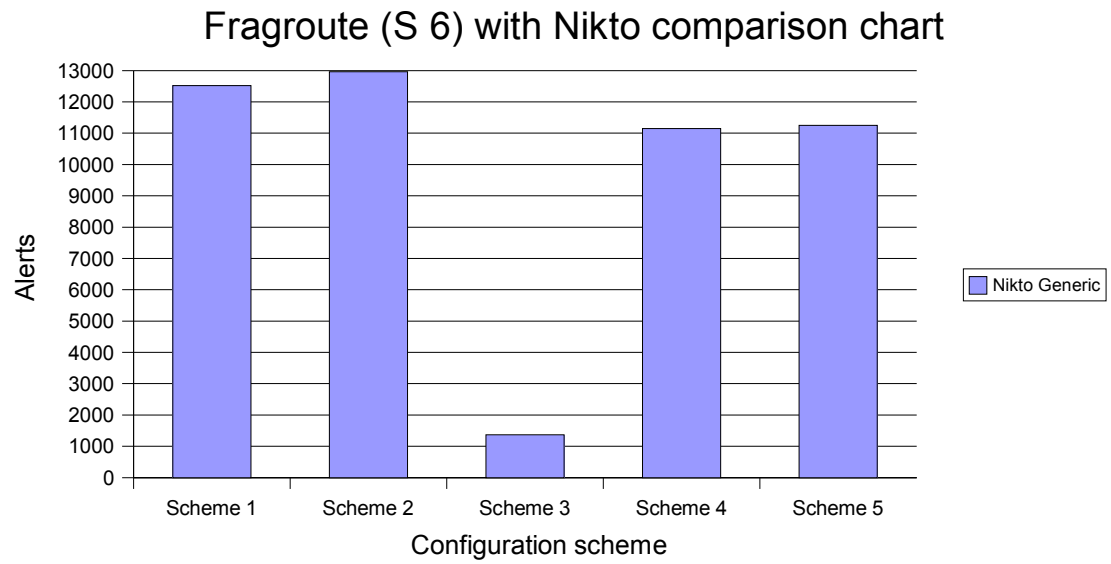
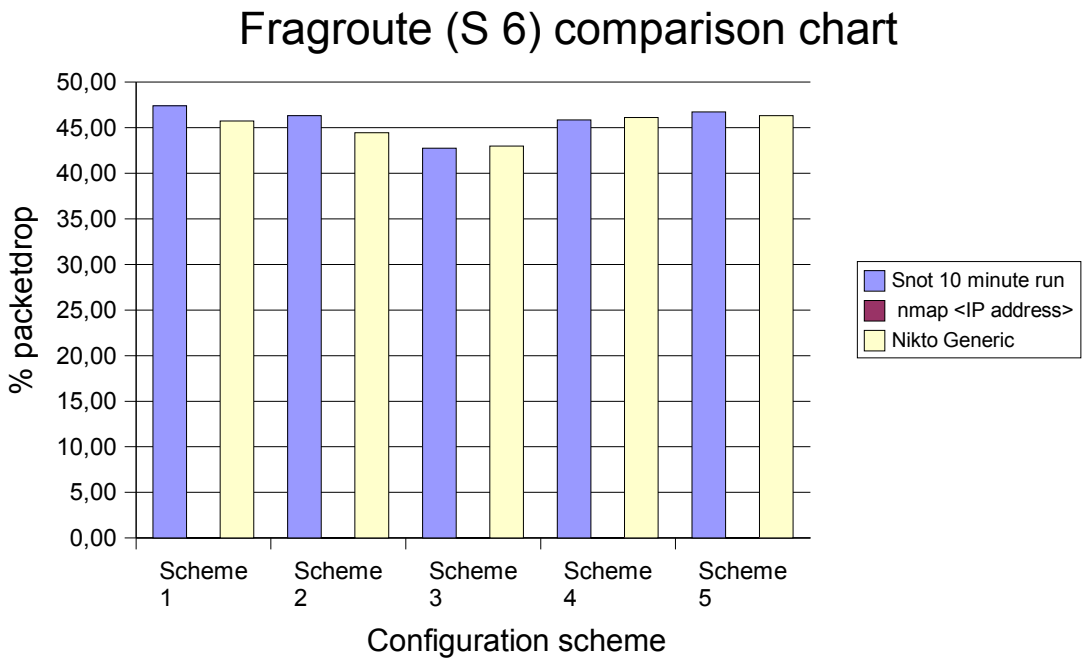


Figure 4.12: Comparison chart of the drop rate using Fragroute with its tools



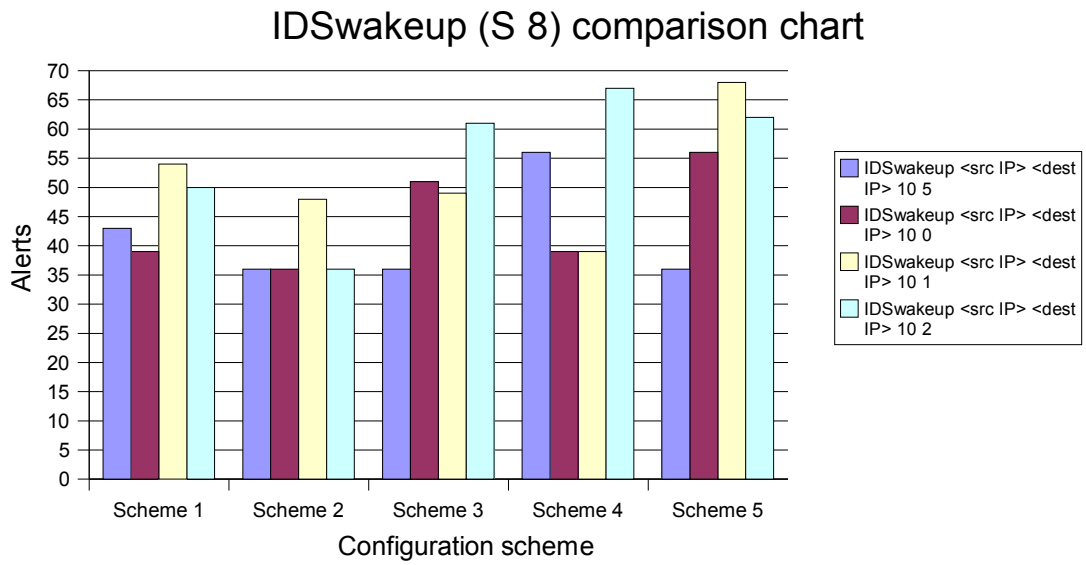
When analyzing the drop rate for the three previous alerts in Figure 4.11, we see that drop rate is high for all schemes. The rate is fairly equal, with the exception of scheme 3, which does not have to concentrate on pattern matching. It is clear that this is Fragroute that expands the traffic volume, making it harder for the IDS with limited hardware resources. If we compares drop rates with the tool's previous tests, this confirms the traffic increase.

4.4.4.7. Results with IDSwakeup (S 8)

Figure 4.13 shows the results from IDSwakeup. These results appear somewhat different than we have seen before. Scheme 3 produces alerts, being one of the schemes with the highest alert level. It is interesting to see that in this test, scheme 2 seems to have the lowest results. In this test both scheme 4 and 5 is higher, with a couple of exception of the parameters used. It is not clear why the results here differs from the others, but this is a tool written for testing IDSs, so therefore this might test better the abilities of the IDS. The parameter responsible for the variations is TTL[23] with different values. This seems to have effect on the results.

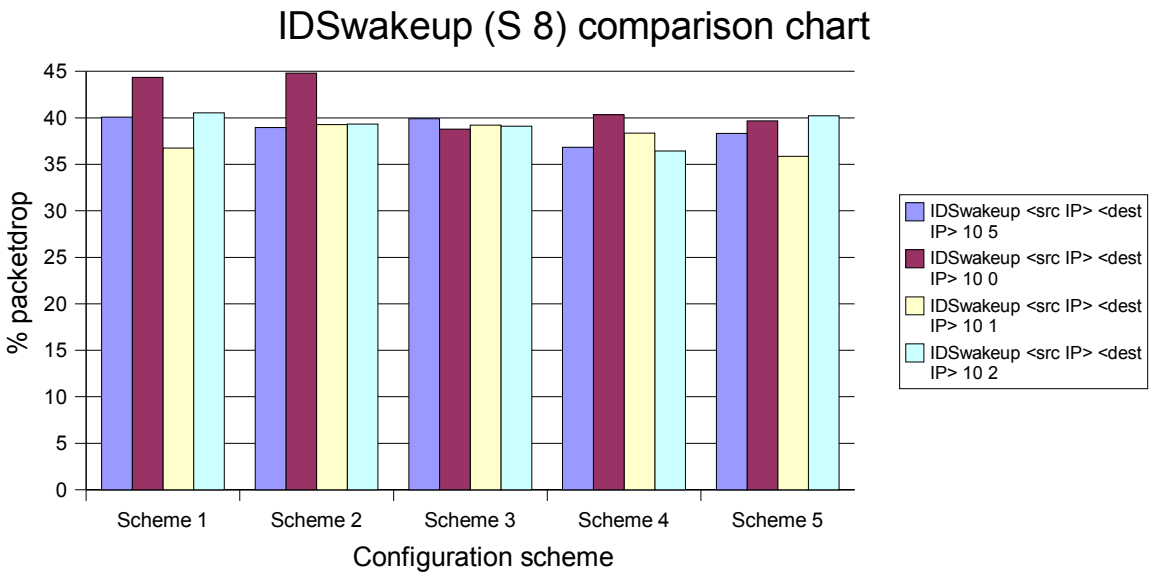
23 Time To Live parameter in the IP header

Figure 4.13: Comparison chart of the results using IDSwakeup



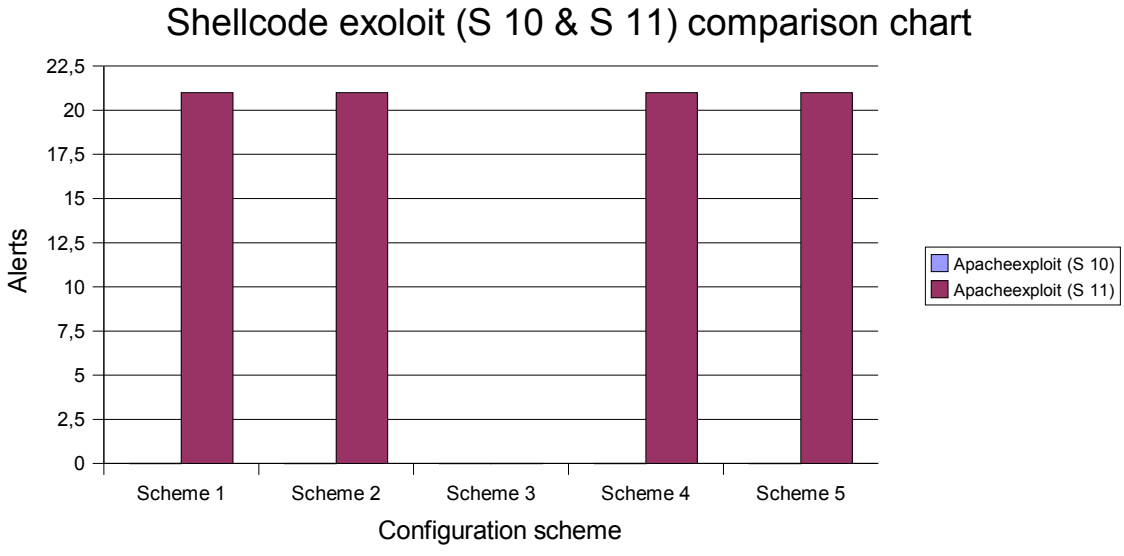
If we see the drop rates from Figure 4.14, we can see that this tool generates some traffic. The drop rate is fairly even among the schemes. It is worth noticing that here the drop rate is not any higher with increased buffers and session timeouts than with the other schemes. This might indicate that the tool uses long session delays during the test.

Figure 4.14: Comparison chart of the drop rate using IDSwakeup



4.4.4.8. Results with Shellcode exploits (S 10 and S 11)

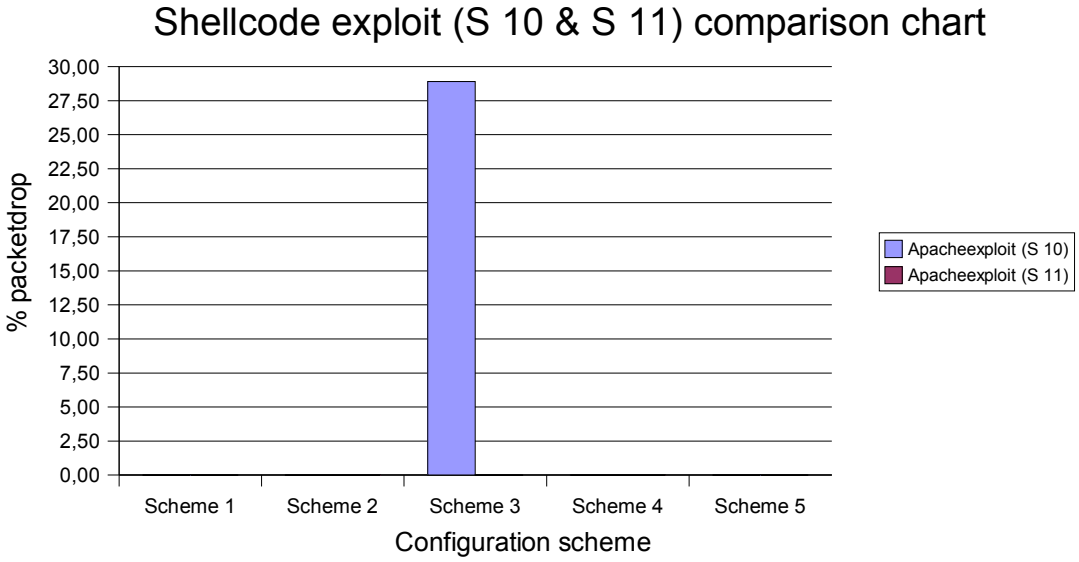
Figure 4.15: Comparison chart of the results using Shellcode exploits



Using shellcode exploits, the results are shown in figure 4.15. Two different exploits for the Apache Web server were used. The first exploit was not detected by any configuration. The second was equally detected by all, except for scheme 3. Again, the difficulties classifying the alert appears. The apache version used, was not vulnerable for these two exploits. So the question is if whether this misdetection is a false negative, or

the detected were a false positives. To make any conclusions like this or similar, the IDS actually have to know what services and versions that runs inside the network. To make this possible, one have to have information flow to the decision module of the IDS, or have a decision system based on information from the IDS and other tools and information sources on the network.

Figure 4.16: Comparison chart of the drop rate using Shellcode exploits



Reading the drop rates in Figure 4.16, we see a very strange result. These exploits generates a little amount of traffic. We see that the schemes using the signature detection does not have packet drops. But scheme 3 has. This must be an error that most likely is due to a sudden use of cpu time on the host.

4.4.4.9.Results with ISIC (S 14)

Figure 4.17 shows the results from using the stress test, and TCP/IP stack tool ISIC. We see that 5 400 000 packets naturally generates more alerts than 1 000 000. Here we see some of the commonalities in the charts from previous tests, but with the exception of scheme 3. Since the traffic relies on testing the TCP/IP stack, scheme 3 also generates errors because of errors in the TCP / UDP and IP packets. But we can see that the configuration here has consequences of the number of alerts reported. As mentioned, seeing the results in general, we see the same tendencies as before, but with default configuration maybe slightly better.

Figure 4.17: Comparison chart of the results using ISIC

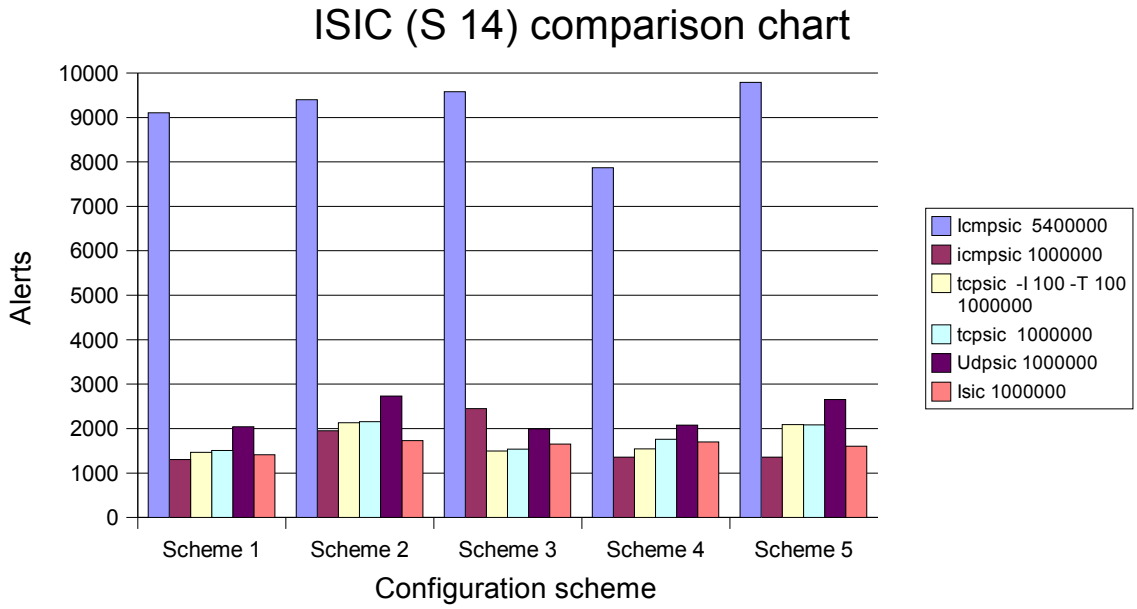
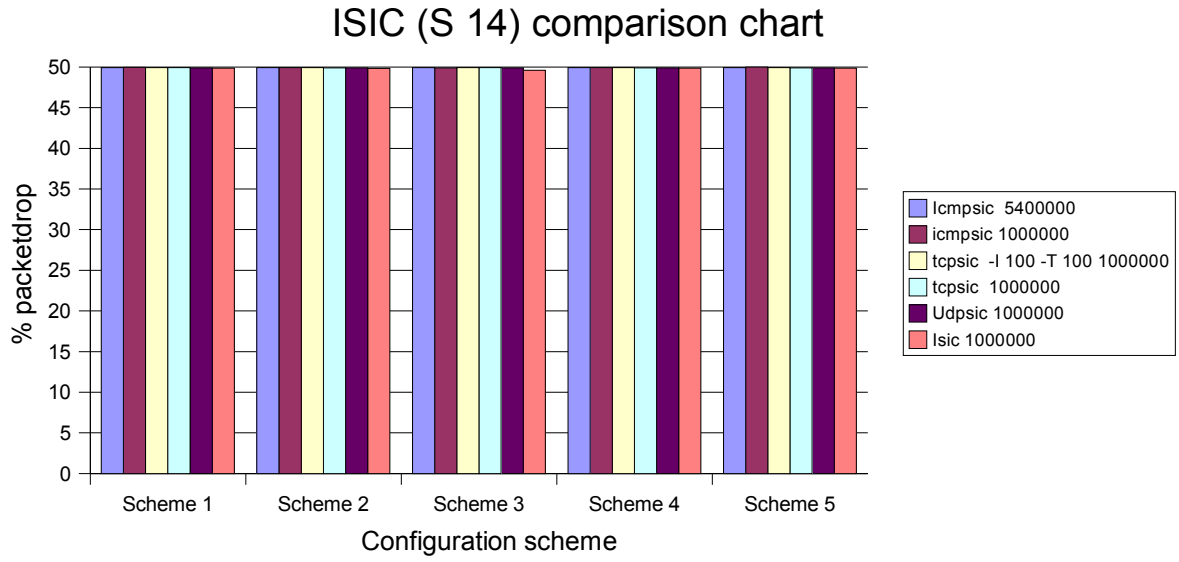


Figure 4.18: Comparison chart of the drop rate using ISIC



From the drop rates presented in Figure 4.18, we see that this really is a stress test, that also could be used as a DoS attack. With drop rates at just under 50 %, this tools almost makes the IDS unusable. This implies the limitation using older hardware with relatively low capacity. Here it is even more important not to interpret the results of the alerts directly. With this drop rate we cannot say other than it is a dead race between the configurations, at best. Seen on the bright side, the one can used the results to see that something unusual is happening, and take further actions from there.

4.4.5. Presentation of results; consequences of hardware change

4.4.5.1. Introduction

Refer to the previous 4.4.2 and 4.4.3 sections.
There exist no data on drop rate to the first test using dedicated machine running the IDS.

4.4.5.2. Results with Nmap (S 1)

Figure 4.19: Comparison chart of the results using Nmap

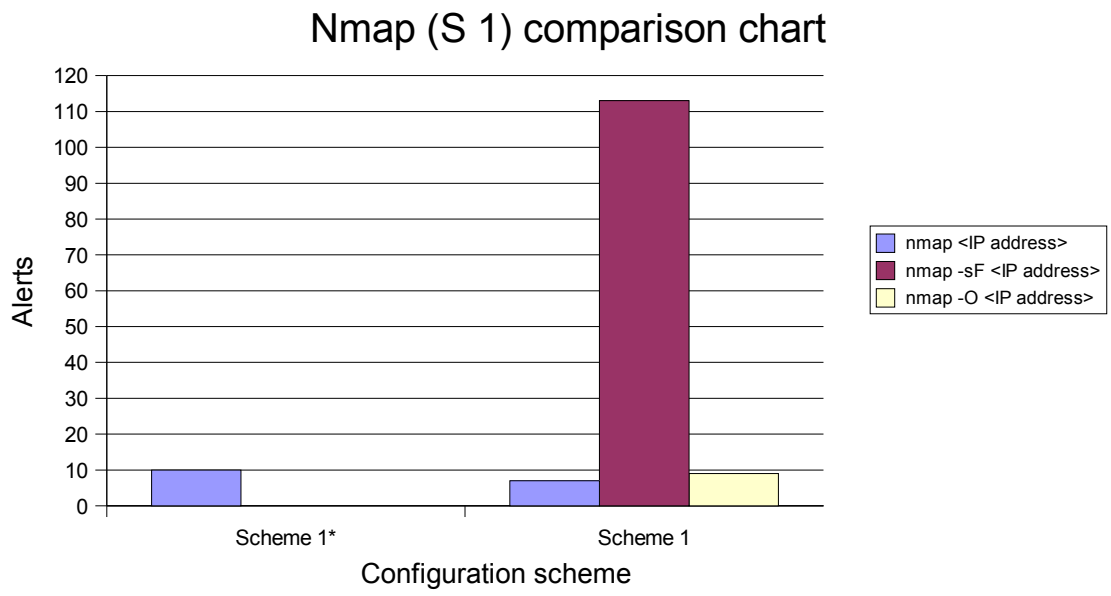
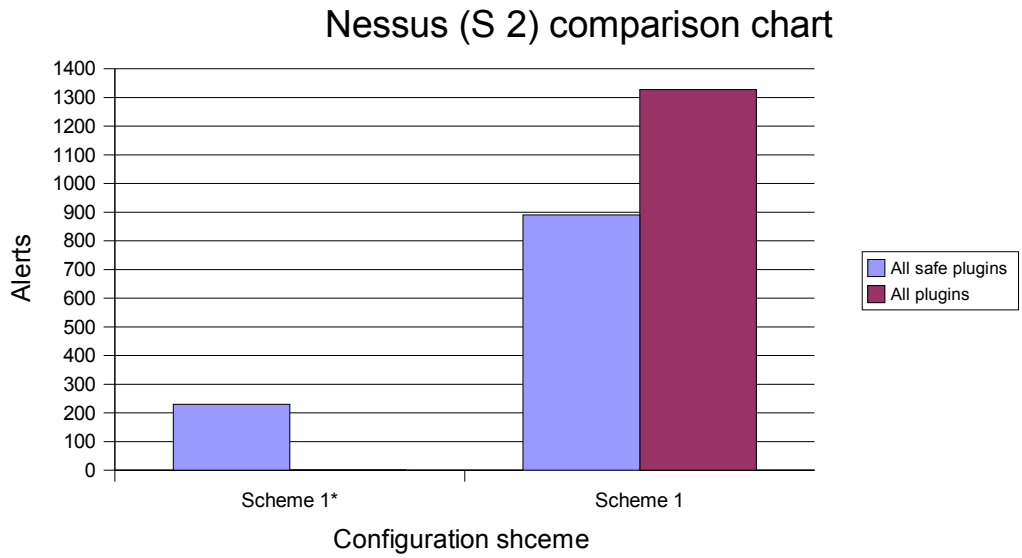


Figure 4.19 presents the nmap results. The initial nmap test there was only one result available. Scheme 1* on the dedicated machine has more alerts than the IDS on the honeypot machine. Since the packet drop rate is unknown we cannot directly conclude anything from this chart, but apparently more alerts were generated from scheme 1*.

4.4.5.3. Results with Nessus (S 2)

Figure 4.20: Comparison chart of the results using Nessus



The results from Nessus are displayed in Figure 4.20. Due to expanding the test after the dedicated machine died, the "All plugins" results are not available in scheme 1*. But we see that the number of alerts in scheme 1* are much less than in scheme 1. There must be some kind of error in the scheme 1* result here. Either that not all plugins really was enabled, or sudden increase in cpu usage on the dedicated machine.

4.4.5.4. Results with Snot (S 3)

Figure 4.21 shows the results using Snot. Initial, Snot ran in scheme 1* in 300 minutes against 10 minutes on scheme 1. Therefore we have to calculate an estimate for a 10 minute run in scheme 1* :

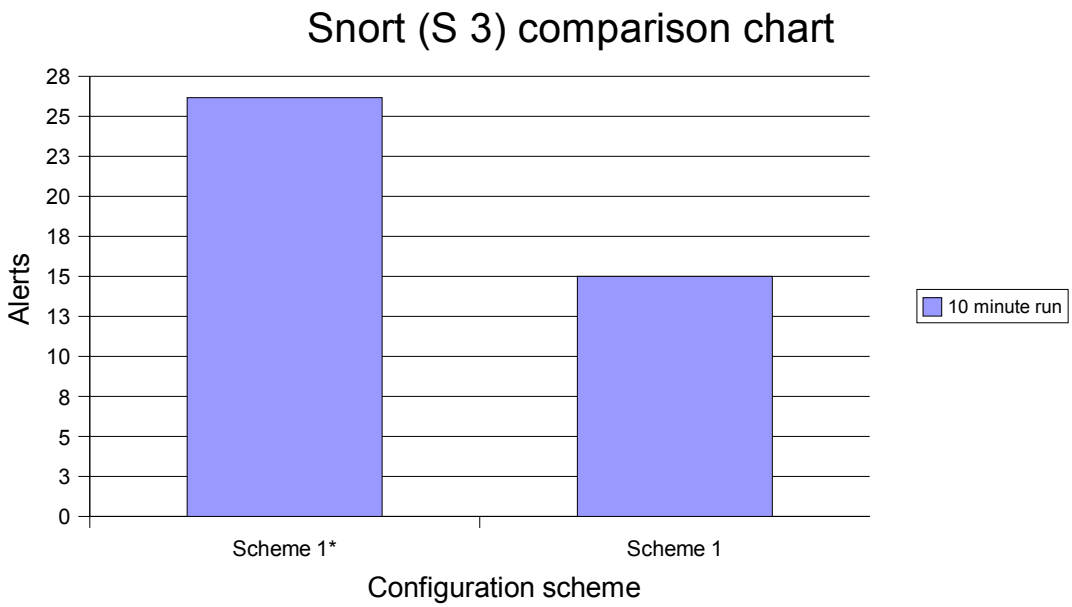
785 alerts / 300 min. * 10 min. = 26 alerts

This result is more comparable since we use the same time period. Note that this is an estimate. Snot's randomly delays from 0-5 seconds between each attack may influence the number of alerts.

So based on the estimate and the results from scheme 1, we see that the number of alerts is significantly higher than for scheme 1. It is difficult to say why, since the configuration is the same, except for http inspection in scheme 1-5[24], which may have had impact on the results. Since http inspection was disabled in scheme 1-5, this has no consequences when comparing the results.

24http inspection would not work whatsoever because of the unicode.map file. It had to be disabled on the honeypot machine. Refer to Snort manual [28].

Figure 4.21: Comparison chart of the results using Snot



4.4.5.5. Results with Nikto (S 5)

Figure 4.22: Comparison chart of the results using Nikto

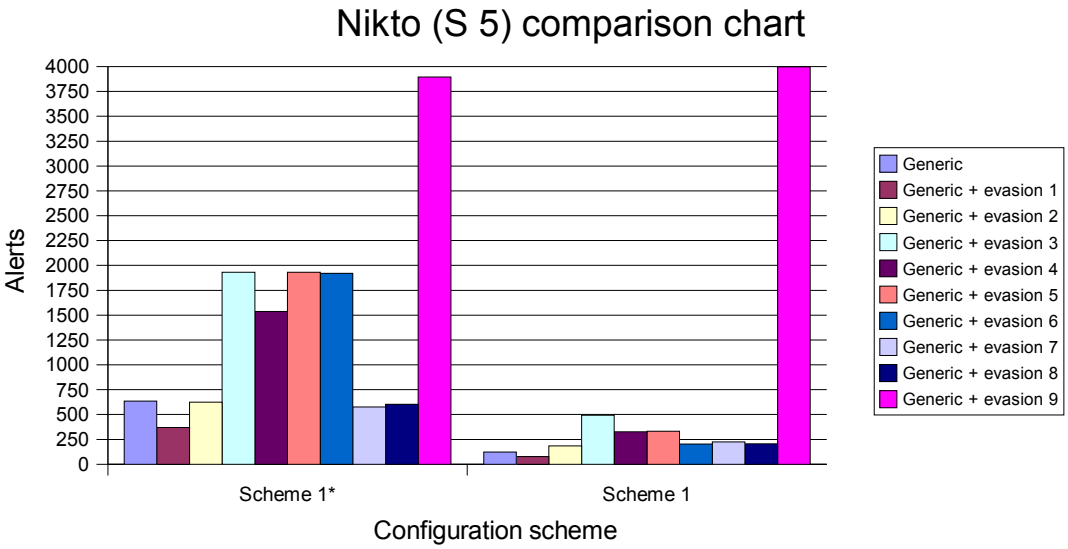
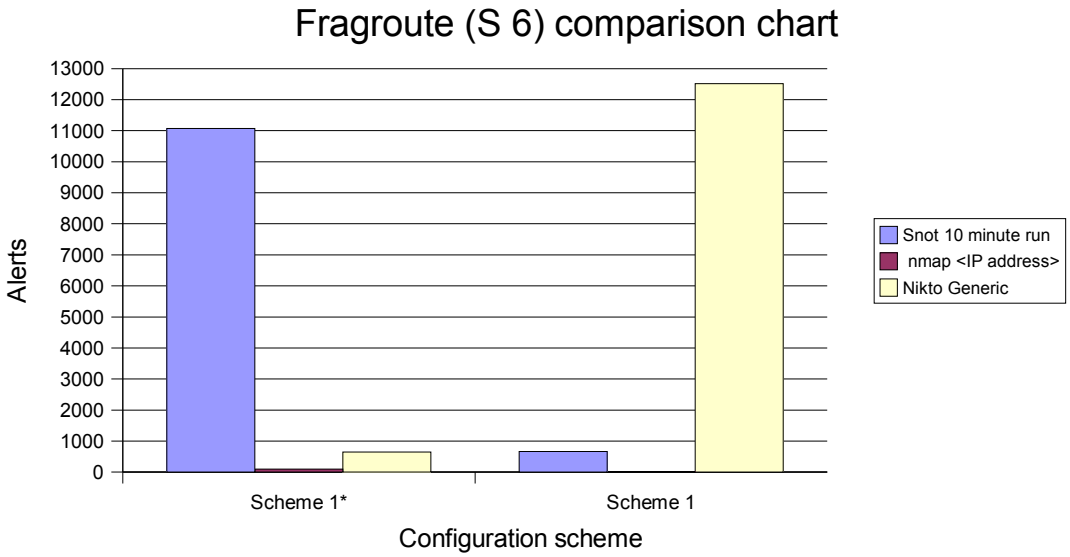


Figure 4.22 shows the results of Nikto, including IDS evasion 9. This was so time consuming that it, due to little time, had to be ruled out in scheme 3-5. If we make an assumption that since scheme 1* were tested on a faster and dedicated IDS machine it should have less packet drop rate than for scheme 1. Following that assumption, the

results here seem more logical. Lower drop rate should increase the number of detected alerts. This we saw in the results in 4.4.4, where higher drop rate gave lower alert frequency. One thing to notice here is the evasion 9[25] parameter. This is somewhat higher in scheme 1. If we check the Appendix A, the drop rate for scheme 1 is 0 %, so packet drops is not the issue here. What causes this is hard to tell for certain. But if we see the trend that in general higher computational powers here gives higher detection rates.

4.4.5.6. Results with Fragroute (S 6)

Figure 4.23: Comparison chart of the results using Fragroute

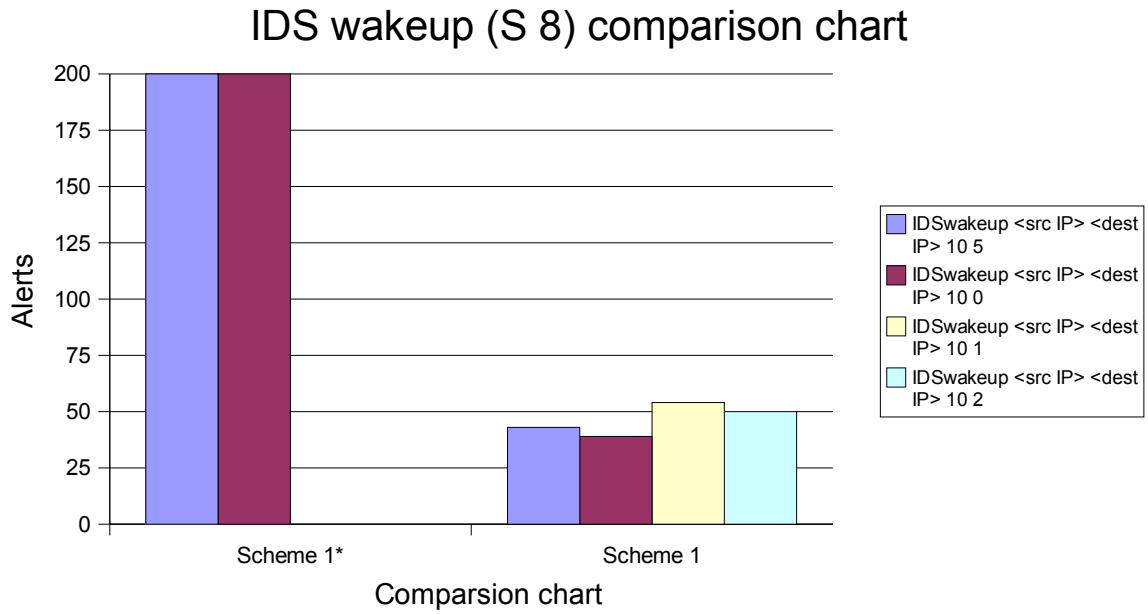


The Fragroute results are shown in Figure 4.23. From 4.4.4.6 we saw that Fragroute, in general, generates more traffic using other tools than they would if they were running standalone. We see that number of alerts running Snot gives extraordinary amounts of alerts. This results seems very strange and is in a whole different class of numbers than the previous test. Note that this test was executed with Snot running in exactly 10 minutes. There are no obvious reasons for this number to be so high. Http inspection is probably not, as the only reason, cause of this high number. Also, during the run of Nikto in scheme 1*, the lights blinks indicating traffic on the hub, were suspiciously "silent". What causing this, is also very hard to tell, but we see in the results, that something must be wrong here. Scheme 1 has been compared with the other schemes, and nothing directly appears wrong. Therefore it is more likely that something happened with the scheme 1* test. These results are too suspicious to be used in any conclusive line.

25 Session splicing

4.4.5.7. Results with IDSwakeup (S 8)

Figure 4.24: Comparison chart of the results using IDSwakeup

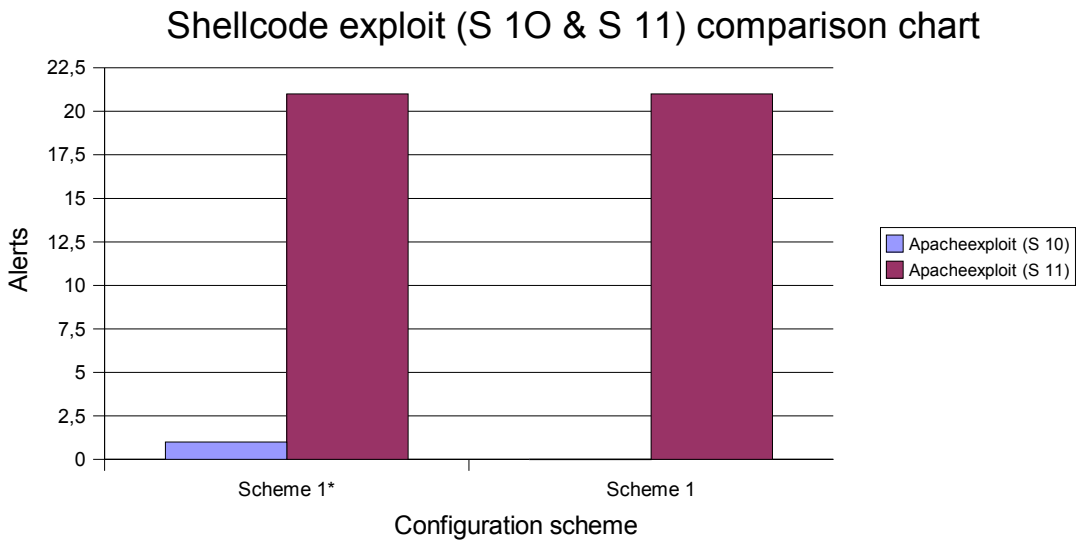


The results of the IDSwakeup test are displayed in Figure 4.24. In scheme 1* only two parameters were tested with. But we see that the number of alerts is much higher here using scheme 1* . Drop rates in scheme 1, is high, ranging from around 35 % to 45 %. This will probably be the reason for much lower alert rate in scheme 1.

4.4.5.8. Results with Shellcode exploits (S 10 & S 11)

Figure 4.25 shows the results from the shellcode exploits. Here we see something interesting. The S 10 exploit was in fact detected in scheme 1*. This was mot detected by any schemes on the honeypot machine. Since there were literally no packet drops, the http inspection setting used in scheme seems to be the most likely cause of this detection. As we can see, the S 11 exploits are detected on both with equal amount of alerts.

Figure 4.25: Comparison chart of the results using Shellcode exploits



4.4.5.9. Results with ISIC

Figure 4.26: Comparison chart of the results using ISIC

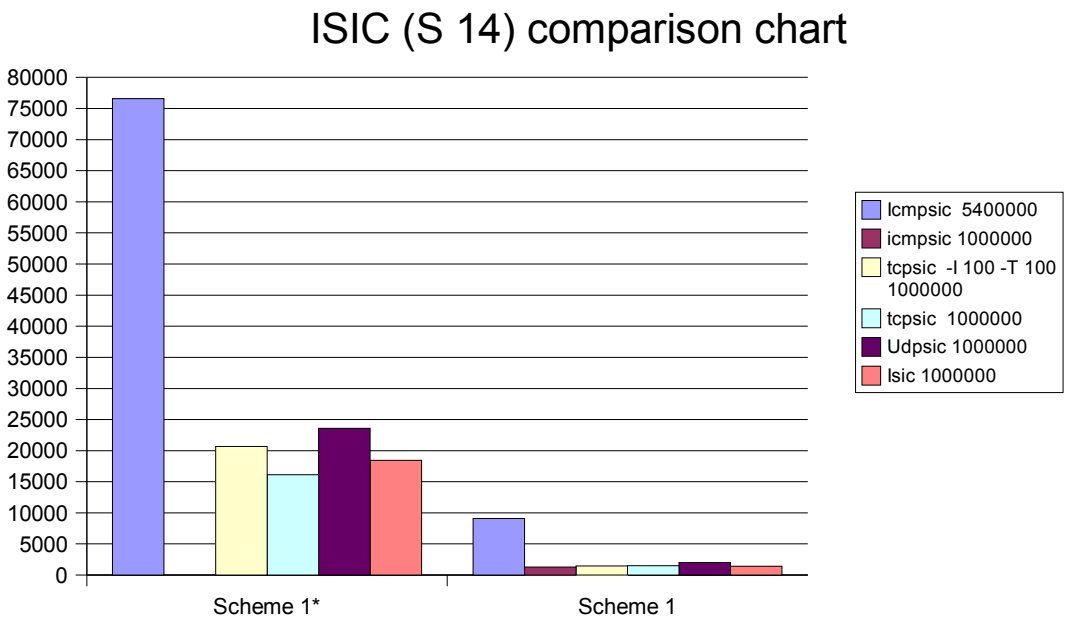


Figure 4.26 shows the results using ISIC. In this traffic intensive test, we really see how important the hardware is. Scheme 1* have an alert level in its own class. We saw that every second packet were thrown using the honeypot machine. It would really have been interesting knowing the drop rate from the dedicated machine, but these are not available. The icmpsic test using 1000000 packets was not committed with scheme 1*.

4.4.6. Summary and conclusions on the test results

4.4.6.1. Scheme 1 – scheme 5

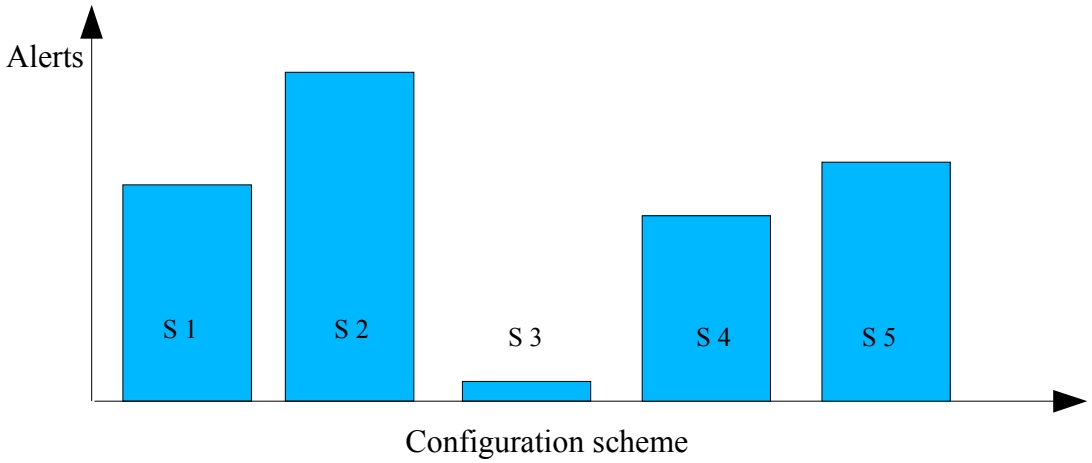
Section 4.4.4 has described the results from the benchmarking of an IDS. In this case Snort was used. We have defined some configuration schemes, a general approach on what to focus on when configuring the IDS, to be used when testing an IDS configuration. 5 different configurations has been tested, and we have seen the results in charts making it easier to see the bigger lines. Due to very limited hardware, we have seen that the IDS tends to throw packets when it is challenged with heavy load from large amounts of traffic, or when other services on the machine running the IDS occupies resources needed by the IDS. The drop rates have shown to be important, and are very often the reason for decreased alert levels.

To use these tests to improve IDS configurations, one must be aware of the limitations in network and hardware. Drop rates have shown to have impact on the results. Event if the results tells us about alerts, the drop rate can be used as a measurement of how much we can trust those numbers. Of course, many other factors, like quality of tools, IDS capabilities, methodology, may also have impact on the results. Refer to Appendix B for some words about reliability of testing.

In this test we there mostly too high drop rate to be able to directly say if one configuration is better than the other. If we had more reliable data, there might have been possible to use the numbers more directly when fine tuning and testing the configurations. Here we have to see bigger lines. We have seen a common pattern of the charts, see Figure 4.27. Scheme 1 has some alerts, scheme 2 has higher level of alerts, scheme 3 with no or little, scheme 4 with lesser than scheme 1 and finally scheme 5 with little more than scheme 1

.

Figure 4.27: Conclusive lines based on the results from the IDS benchmarking



If the numbers in the test had been more reliable, we could have concluded with that applying attributes from configuration scheme 4 would have a positive effect. But as we know from the drop rates, scheme 4 had typically relative higher drop rate than the others, so we cannot conclude directly with that. Another thing we can see, is that applying every signature available, might not be the the best way of reducing the alert level. This confirms the Snort manual [28], were dynamic and adapted rules and signatures are encouraged to use. This means only adapting signatures that matches the traffic available on a network. For instance, if one does not have any telnet servers on the network, why should on search for inbound telnet traffic? The only reason would be to monitor attempts, if that is interesting to gain knowledge of. We also see that the IDS will miss very much of the attacks, if by some reason all signatures have been disabled. This might be trivial, but if so happened it would be preferable that an IDS would report little, instead of nothing. But then, it might be more suspicious if it didn't report anything. The point is that eliminating signatures that should be activated, important information might be lost that could in worse case be a false positive, a missed attack.

4.4.6.2. Scheme 1 - scheme 1*

Here we compared the test results from to different computers, with different preferences. This we saw the consequences of. The test parameters was not 100 % equal, with the result of comparison problems. A slightly difference in the configuration scheme resulted in deviations. This shows that in order to compare a scheme from a computer to another, or even among different kinds of IDSs, the configuration based on the scheme has to be as identical as possible, or else one have to take the differences into account.

We saw that differences in hardware resources made quite big deviations in the alert level. Due to lack of drop rate information from the fastest machine, we cannot directly conclude anything about whether the drop rate decreases, but logics say that it ought to.

Using these results directly is no good. There are to much uncertainties related to one or the other result. But what is apparent, is that using adequate hardware is necessary. If one is uncertain, this benchmarking methodology and its tool can be used, with focusing on the drop rate.

4.5 The other scenarios

Executing the IDS benchmark is a very time consuming process. Especially when applying many configuration schemes that should be tested, often several times when fine tuning. In this thesis, one of the goals was to try to test the configurations in different environments, described in chapter 3, to see what variations the change of environment makes. There has not been time for this, but we have seen that what signatures that have been enabled, has effects on the results. Adapting the signatures to the actual environment will therefore be an important issue when benchmarking the IDS configurations. If one test an configuration in one environment, this might not be just as good on another network with other preferences. This is also mentioned by Ranum in his experiences with IDS benchmarking [1].

4.6 Guideline to IDS configurations

Clearly, due to the fact of high packet drop rates, it would be too ambiguous, if not erroneous, to try to point out how an IDS should be configured. To do this, we had to increase the number of configuration schemes, by making each scheme more specialized and narrow. But this has a drawback. If the configuration schemes are to narrow, they might be harder to use with other kinds of IDSs. But if comparison among IDSs are irrelevant for a cause, more specialized schemes can be made, since it then might only be one IDS benchmarked. One have to check the reliability of the tests to get more accurate results, see Appendix B. But validity is very important as well. One have to be sure that the tools tests what the properties of the IDS that is intended.

To give a simple example of how pointing out improvements of an IDS configuration can be done, we can use the results described in 4.4.4 and the conclusions in 4.4.6.
If the drop rates had been much lower, and the results as they are, we could have concluded with that since configuration scheme 4 had general lower alert rates[26]. In addition, it seems like adapting signatures to its working environment also would have a positive effect. By this, make such adaptions in the configuration. After this, a new round of testing has to be applied to verify the effects of the changes. The more detailed configuration schemes to compare with, the better and more specific elements that improves the IDS configuration can be revealed. But it is a drawback of heavily expanded time consume, because of higher number of test rounds. This must in that case be a trade off, regarding granularity of configuration schemes and usage of time. All in all, it is a question of how much resources one is willing to sacrifice, i.e. what gives most effect of the money.

26 Meaning lower false positives. False negatives also has to be encounted for in the evaluation

5. General discussion of results and experiences

5.1 Software

5.1.1. IDS

It is clearly that more than just one IDS has to be tested to increase the validity of the test results. Snort is representative for the NIDSs that are freely available, and has rich functionality with lots of configuration options. However the results from the tests are characterized by the lack of computing resources. This can clearly bee seen after the IDS were moved to a slow machine, where packet drops so severe that nearly almost every second packet were thrown away. This decreases the value of the results. One cannot be sure if the percentage of packet drop is constant. Since it is the same host that runs the services often attacked, and the IDS. They will claim the same resources, that are definitely limited, leading to that at least one of them loses. This is typical when both are under heavy load. Therefore we see an increase in packet drops with intensive and exhaustive test that require a lot from both the service and the IDS. A DoS attack against a service will probably be an DoS against the IDS as well. This is a big drawback in the reliability of the results. This is an unwanted situation, but the hardware situation during the work with this thesis is the reason for this. By the projects final weeks it was to late to replace any hardware.

5.1.2. Testing tools

5.1.2.1. General issues

Much of the testing tools used in this thesis are quite old, with no longer development. 4-5 year old software does not always work on todays operating systems. This project has proved that fact. The lack of good and automated testing software leads to use of large amounts of time trying to get the software to work, and work as intended. Some of the software are quite useful, and performs very well of what they are intended to do. After searching the web for open source / BSD licensed / freeware tools, most of the developers says that their tool tests IDSs, but I haven't found any yet, that tests a wide aspect of the IDS. Most tools have a narrow field they tend to operate. By relating to many small and specialized tools, the testing becomes more time consuming and complicated than it ideally could be. Nessus impressed me during the benchmarking process. Not only is it one of the most advanced vulnerability scanners, but it has also implemented IDS evasion techniques, based on Whisker. It would not surprise me, that this tool may evolve towards a IDS testing tool as well in a relative short time period. It

is my opinion that the results from the testing would have been easier to draw conclusions from if we had a more unified tool set available.

Some of the tools aren't very usable in attack statistics. When a tool is used for testing an IDS's signature capabilities, one cannot count on that the tool gives information of how many attacks that are launched. This makes using the results in accurate calculations. The tools may never have been intended to be used in statistical analysis, but this is surely something to think about when developing IDS testing tools in the future.

The next chapter are some words on each tool describing experiences with them in this work, and weaknesses discovered during the test..

5.1.2.2. Nmap (S 1)

Nmap is a great tool for testing the IDS's port scan activity abilities. It is the most common port scanning tool available, and is rich of advanced functionality. This come in hand if one will tune the IDS to only detect sophisticated port scans, like stealth scans.

5.1.2.3. Nessus (S 2)

Nessus was originally a port scanning tool, using nmap, and a vulnerability scanner. Lately it also has gotten the Whisker IDS evasion abilities, that makes it even more actual as a IDS testing tool candidate. It is rich on features, but the interest in IDS context will be to test the IDS's ability to detect port scans and vulnerability scans. The IDS evasion functionality increases its value if one does not use Whisker or Whisker based software. In the test used here, none of the IDS evasion techniques were used, since Nikto (S 5) uses those features. Nessus has may advanced features for its vulnerability scans. The plug in based system makes it easy to customize the scans, that also can be useful for IDS testing purposes.

5.1.2.4. Snot (S 3)

Snot is a false positive generator. It was originally designed to use with the Snort IDS. It uses the Snort's signature database as input to generate traffic that contains the patterns found in the snort signatures. It can though be used with other IDSs, but is then limited to generating attacks that Snort detects. This has not been tested here. The problem with Snot, is that it is old, and has not been updated for a while. It is designed to use Snort version 1.8 signatures, but in these test version 2.1 signatures were used. The consequence of this, was that many of the newer signature were rejected by Snot

because of changes in structure in the signature database[27]. As of lack of features, it would give much more value, when testing reliability, that Snot would give some statistics of number of attacks generated, time consume. This would indeed aid in measuring performance. The last comment on this tool, is about making it work. It is distributed as source code, so it has to be compiled on the computer. Since it is so old, there were some problems to get it to compile. Necessary libraries on the Linux system was to new, and not backward compatible with older versions. The solution was to install old libraries. These problems is relatively easy to solve, but takes some time to figure out what is wrong and how to solve it. This increases the level of usage, and might be a negative side for many.

5.1.2.5. Sneeze (S 4)

This software is also a false positives generator. But it was impossible to get it work on the systems used in the tests. Several attempts were done. It finally compiled, but would not work as intended. None of the efforts in trying to make it work failed. It might be possible to make it work, maybe on other Linux distributions, or platforms, but one have to have in mind that there is a possibility that it might not work. No results were possible to produce, and that is why it is not in the test results.

5.1.2.6. Nikto (S 5)

This tool scans a web server for vulnerabilities. This is based on the Whisker libraries for IDS evasion techniques. In the tests used here, the apache web server contained very little features to test. This tool will probably be more fruitful in when testing an in-real-life use web server, maybe running a web shop or a log-in site. All the IDS evasion techniques, 1-9 were tested. IDS evasion number 9 session splicing, was too time consuming, and was only executed on configuration scheme 1* and 1-2. The conclusion after using the IDS evasion capabilities, is that in most cases the number of alerts increased, compared to using Nikto without these features. This indicates that the IDS in this case, managed to detect these evasion techniques, but generated much more alarms in addition. The question is whether the IDS should report that it has detected evasion techniques, or it just should report the events triggered by the underlying software, here Nikto's web tests.

5.1.2.7. Fragroute (S 6)

Fragroute uses other tool's traffic, and rewrites it and sends it to a destination host. In this benchmark, Nmap, Nikto and Snot were used to see how the IDS dealed with this modified traffic. This tool performs fragmentation, delays, drops, overlapping,

27 The “database” consists of several text files, joined to one file with the Unix/Linux command 'cat'

reordering, duplicating on the ingress traffic. As a consequence, the IDS generates much more alerts by sending the traffic through this tool, than the tools does standalone. The vast majority if the alerts are due to TCP / UDP and IP header errors.

5.1.2.8. Fragrouter (S 7)

This tool performs much of the same as Fragroute, though they have differences. Fragrouter is intended to run on a dedicated host. It accepts incoming traffic, preferably traffic by other tools and rewrites the IP traffic. The IP packets will be reordered, fragmented in different sizes, or else altered in many ways. The goal is that the traffic should evade the IDS. Unfortunately, there just were not enough machines available in this test setting to make use of this tool. It would indeed be interesting to see how the IDS would deal with this kind of traffic. The OSEC methodology [3] has defined several test with this tool, but this has to be done in later experiments with more machines available.

5.1.2.9. IDSwakeup (S 8)

This is as tool designed for testing IDSs. It generates a series of false attacks to see if the IDS can detect it. There are no data on how good this tool is, but we saw the variation of alerts based on the configuration schemes, and thereby usable. Though, it has no update functionality, like for example Nikto has.

5.1.2.10. ADMmutate (S 9)

ADMmutate is an API[28] for generating polymorphic shellcode exploits. Ordinary exploits have a typical network traffic pattern making it easy to create a signature for it. Making the exploit polymorphic means that the characteristics of the exploit are being changed by this API, so that the attack pattern changes compared to the original. This requires that the source code for an exploit is available, and calling some API functions before it is compiled. Though the manual explained that it was quite trivial to implement, it was not possible to compile the exploits using the ADMmutate API. Threes different exploits were tried, but no one would compile. With computer engineering and programming background, this ought to be trivial, but days of trial and error combined with searching the Internet gave no results. It would really have been interesting to see if the IDS would have managed to detect polymorphic shellcode exploits. The API has not been updated for a while, and this might be the cause of why the code would not compile with the API's functions[29].

28 Application Programing Interface
29 The exploits compiled fine without using the API

5.1.2.11. Apache 1.3.x – 2.0.48 remote users disclosure exploit (S 10)

As an example of exploits, this was used to see how the IDS dealt with shellcode exploits. Though the web server was not vulnerable for this exploit, it was still interesting to see if the IDS detected it. As mentioned, registering false negatives is not trivial, but using exploits and combine it with the polymorphic API, it is easier to register these false negatives when an actual attack is executed. This exploit did not produce any alerts, but since the web server was not vulnerable in this case we might say that it was no false negative. But the IDS cannot know for sure which version of software that runs or not, so it actually was a false negative.

5.1.2.12. Apache 1.3.X Remote Exploit (S 11)

The web server this exploit attacked, were not vulnerable either, but here we got alerts from the IDS. The exploit generated 21 alerts in the tests. This should be more than enough to alert of an exploit attack. It is tempting to say that 20 of the alerts were false positives, but it relies on how the exploit is made. It could easily be built abusing several exploits at once.

5.1.2.13. 7Plagues (S 12)

This is a Perl script launching huge amounts of traffic, a DoS tool. But again, trouble appeared. The script was made using the old threading model in Pearl, so even with downloading old Perl modules of different kinds, it would not run. Though the author is familiar with Perl programming, after many, many attempts, this script could not be used. ISIC therefore had to be the stress tool used. This script needs to be updated in order to work with the newer operating systems and Perl versions.

5.1.2.14. ISIC (S 13)

This stress testing tool testing the TCP/IP stack implementation, could also be used as a form of DoS. This came clear when we saw the drop rate from the IDS. The tool has the ability to test the IP, TCP and UDP protocols. The intense traffic generated by the ISIC tools really shot the alert levels to the sky. It is worth noticing that this tool can crash operating systems with poor TCP/IP implementation, which makes it a quite dangerous tool, so it might be important to detect such activities. But the downside is the high level of alerts. It is tempting to classify most of them as false positives, but security policy and IDS implementation have to be the parameters for such classification. This tool is is anchored in the OSEC methodology [3], but the throughput at 8 Mbit/s and 5 400 000 packets was to heavy both for the IDS and the machine running ISIC. The real throughput reported by ISIC was between 3,5 and 4,0 Mbit/s, regarding which protocol

tested. The amount of transferred packets were due to practical reasons reduces to 1 000 000. The hardware limitations therefore was responsible that this tool could not be executed at “full speed”.

5.1.2.15. T 1 Network Traffic Generator (T 1)

This tool is intended to create network traffic. It was planned that this should be used in scenario 2, where the IDS tests were to be executed with background traffic. The tool were tried out, but even if it complied well, it didn't seem to work. It is not clear what was wrong, but there were not enough time to find it out. Finding a good traffic generator is hard. Ranum [1] also points out that most of them are flawed in IDS benchmarking situations. It depends on the intentions. Here it was originally planned to make some extra noise and traffic on the network to challenge the IDS. This could have also be done with ISIC, but it would have triggered many alerts, making it harder to see differences among the configurations.

5.2 Benchmarking methodology

The methodology used as base for the testing have so far been adequate. There were never any goals to find the “best methodology in the world”, but rather to find a methodology that were sufficient enough to cover the most important aspects of benchmarking IDSs. Focusing on the IDS configuration has not been any problem using the methodologies in [2] and [3]. One of the greatest challenges has been to find suitable software of good quality, that tests the topics described in the methodology. As mentioned before, the hardest part has been insufficient hardware resources, which also has consequences for compliance of the methodology. Some requirements have not been achieved because of limited computing power. Stress testing tool as ISIC is a very good example of this. Fragrouter is another example on a tool that hasn't been used as according to the methodology. OSEC has several tests based on Fragrouter, but it couldn't be executed due to lack of computers.

5.3 Benchmarking IDS configurations

The idea of purposing a methodology to use for non-experts in benchmarking IDS, has shown to work, based on the experiment. Though this experiment has been troubled with insufficient hardware, this can be used to underline how important it is to know the limitations in the IDS testing, this is also confirmed by Ranum [1]. Using slow hardware in this experiment, revealed high packet drop rate, making the results less reliable. Considering drop rates and combining results, we were able to see common pattern that some configurations tended to have generally higher or lower alert levels than others.

The trend was apparent with almost every tool used. This might be a good indication. But due to the fact of poor reliability, we cannot conclude anything directly, but to say that this seems to work and be a way of performing IDS benchmarks with focus on configurations. Its strength lies in the foundation of the methodology, so with further work based on this thesis, more conclusive information might be gained.

6. Future work

6.1 Introduction

Working with this thesis, it has come clearly that IDS benchmarking is very time consuming and a complexed topic. Very much of the initiate planned work, have have to be ruled out due to time consuming tests, and technical difficulties. During the tests, it came also clear that more work has to be done with the software, so based on the experiences in the IDS testing the next sections proposals for improvements and expansions of this work

6.2 Methodology

- Evaluate the OSEC and OSSTM methodologies with a deeper an more thorough analysis with focus on how well they cover all aspects of an IDS.
- See if there are any method to "benchmark" how good the methodologies are.
  - If so, find out how good open methodologies compared to those not available for public.

6.3 IDS and configuration benchmarking

- Execute the tests using the two last scenarios defined in chapter 3, preferably with scenario 1 since hardware platform might be different, with the goal to see how static and dynamic[30] background traffic affect the results.
- Test other IDSs.
  - Compare several IDSs.
  - Compare one IDS using different hardware to see the affects.
  - Compare free of charge IDSs against commercial using this methodology or an extended edition of it.
- Use this methodology on a real-life network, or bigger test network
- See if there are possibilities to port this experiment to other types of IDSs, e.g. Host based IDSs.
- Create more configuration schemes with higher granularity.
- Create more configuration schemes and test them among several IDSs and see how configuration schemes makes sense when comparing performance across different IDSs.
  - See how granular these can be before encounting problems with different configuration architecture.

30 And authentic, refer to scenario 3

- Deeper study of the connection between configuration scheme and the configuration elements.
- Deeper study in reliability and validity of an IDS and how it deals with tests.
  - How does this affect results in a benchmarking situation.
- Study the effects of using analyzing tools with intrusion databases.
  - Usability, user friendliness, error rates are important topics.

6.4 Testing software

- Further develop some of the older tools not working, or difficult making it work.
- Improve user friendliness of many of the tools.
- Try to combine some of the tools and write it as one. Dealing with fewer tools might male the testing less complicated.
- Create new tools specialized in IDS testing
- Find replacements for tools not working in this experiment, and perform benchmarks
  - Evaluate against the methodology
- Experiment with usage of several traffic load generators and see if those has effects on the results.
- Enhance the software test suite
  - Reveal weak coverage according to benchmarking methodology.

7. Conclusions

7.1 Methodology

In this thesis, The OSEC and OSSTM methodologies were used as a base for the IDS benchmarking. Both the OSEC and OSSTM methodologies had specialized sections for benchmarking IDSs. The OSEC methodology's NIDS testing criteria, were far more detailed than the OSSTM, therefore this methodology influenced more than the OSSTM. The OSEC methodology had a series of tests with detailed description of what to test, and what results were acceptable or not. It seems that these methodologies together have been adequate for this usage in the experiment of this thesis. There were no indications that there were any limitations in the methodologies that might have impacted or decreased the value of the results. On the other hand it is hard to measure if the methodology is implemented right. Therefore the question of the validity in this is not possible to say anything conclusive about. There are too many black boxes in such testing environment, which one does not see in details what happens with the data flow. This is for IDS, testing tools and even how the operating systems and network devices deal with the network traffic. We just have to trust that the results are somewhat correct, but have in mind that errors may appear, since we do not have the entire picture of haw data is processed[31]. Measuring methodology compliance, the question if we actually follows and use the methodology as intended, is a thesis in it selves, and it is not guaranteed that the work is 100 % compliant to the methodology here either. Reasons for this are because of software tools that did not work or had some lack of feature. In addition, the test network was too little and simple to implement all checks defined in the methodology.

7.2 IDS and benchmarking configurations

In the experiment, we used Snort as an example of using a free IDS. The introduction of configuration schemes was essential to be able to classify and test the different configurations. The schemes made it also easier to pinpoint what improvements that could be gained in a configuration, and it has a potential to be a framework when benchmarking configurations among different IDSs. This has not been tested, but the though behind these schemes are to make it possible to have a cross IDS comparison framework. Further testing have to be done, to prove that it is feasible. Tested with one IDS they the concept of such configuration schemes seems useful, but due to lack of time, only five general schemes were designed and tested. To make the benchmarking and comparison more fruitful, several more schemes must be defined with higher granularity that identifies more specialized parts of the configuration.

It was possible to detect variations among the results of the different schemes. Though, due to slow hardware, the reliability of the results was not good. This Appendix B

31 At least not in this project

shows. Therefore one has to be very careful with drawing any conclusions. Still, we were able to see some common characteristics when comparing all schemes with the results from all tools. Thereby it is demonstrated that the changes in configuration can be detected by benchmarking. But it is important to be aware of the limitations of both hardware and the tools used. The tests of this thesis showed that slow hardware were responsible for high packet drop rates, especially at high traffic volumes.

Further testing has to be done to verify the effects of the changes. The work has shown that this is a time consuming affair, when using many tools with many configuration. So a lot of time have to be invested when using such method. Not all tests planned in this thesis were carried out due to optimistic plans and some technical difficulties. Such errors are easy to make, and one have to remember that trouble can occur a, and have in mind that these tests takes more time than one would think. In light of the experiences from this thesis, on can say the original plans was a bit too optimistic.

7.3 Software tools

The tools used in the tests when benchmarking the IDS were anchored in the methodology as good a it was possible to get it. They were freely available using the licenses defined earlier in this thesis. The experiences with these tools showed that there are a lack of unified IDS testing tools. If following the methodology, one has to deal with several tools doing special tasks. Many of these these were useful, bot sadly some of them would not work, due to the fact of being old and incompatible with newer systems. Some had not been updated for some years, and it is questionable if they are valid today as they were at the time they were released. Therefor a lot of work have to be done on this front of IDS testing, New and updated tools may aid gaining more valid results. In addition, some of them were both complicated to get to work properly, as well as complicated usage and user interface. Performing tests with current software has a relatively high user level. One have to have deeper knowledge in operating systems and software development, as well as detailed network communication.

7.4 Summary

In this thesis, it has been focused on finding a way to use IDS benchmarking to improve the configuration of IDSs. Using public available methodologies based on best practices and experience, the thesis has founded a base for further work in developing a benchmarking suite for novices in the IDS benchmarking area. The suite of testing software has shown that there are software available for use in IDS testing, though it is not perfect, and a lot of work must be done improving existing software. Much of the existing software are too buggy and outdated. There is a need for unified tools. The ideal might be one tool testing all aspects of the methodology, but this might be to ambitious today.

We have used the experiment to show that the idea of benchmarking IDS configurations

makes results. The variations are detectable. But due to poor reliability in the test results, we cannot dram any direct conclusion of important elements that improve the IDS configuration. Much more work has to be done with faster hardware and more testing of different and more detailed configurations in order to get steps further in the process of identifying important configuration elements.

The usability of the approach described and tested in this thesis, is a bit impaired by the fact that testing these configurations takes very long time. Configurations have to be tested an compared with other configurations again and again. The use of time and complexity might be frightening factors due to economy and used of resources. But with further research and development might reduce the efforts in a more standardized manner. This work is though a foundation to further studies with the goal to finding a more standardized and open methodologies of benchmarking IDSs and their configuration, with goals of improving the IDS performance.

All research questions have been addressed, but is not easy to make a conclusion with two lines below the answer on every question asked. There are many variables to be aware of and conditions that impacts on the results. Even though, we have addressed the problems and considerations were there has been to thin foundation to anchor conclusions.

Acknowledgments

I'd like to thank the following that contributed to the work with this thesis:

- My teaching supervisor Professor Einar Snekkenes.
- Fellow students in my class, 02MAISA, for helpful feedback during our presentations.
- Erik Hjelmås for providing hardware and helpful support and hints.
- IT Service at Gjøvik University College for providing hardware, and a separated subnet with unlimited Internet access.
- Openoffice.org (http://www.openoffice.org) as a splendid alternative to MS Word. No comment about Latex mentioned.

And finally I would like to thank my girlfriend and roommate Iselin Hillersøy for patience, support and care, as well as my inspiration and motivation during the work with this thesis.

References

[1]
Markus J. Ranum,
Experiences benchmarking Intrusion Detections Systems,
NFR Security Press,
2001

[2]
Pete Herzog,
OSSTM 2.1 Open Source Security Testing Methodology manual,
The institute for Security and Open Methodology, ISECOM,
2003

[3]
Neohapsis OSEC Open Security Evaluation Criteria,
Open Security Evaluation Criteria (OSEC) website,
http://osec.neohapsis.com/

[4]
Douglas C. Schmidt, Adam Porter,
Leveraging Open-Source Communities To Improve the Quality & Performance of Open-Source Software,
In First Workshop on Open-Source Software Engineering, International Conference on Software Engineering, May 2001,
2001

[5]
N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee and R. A. Olsson,
A Methodology for Testing Intrusion Detection Systems,
IEEE Transactions on Software Engineering,
1996

[6]
R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham and M. Zissman,
Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation,
IEEE Computer Society Press,
2000

[7]
Distrowatch.com,
How popular is your distribution?,
http://www.distrowatch.com/stats.php?section=popularity,
April 2004

[8]
W. Stallings,
Network Security Essentials Application and Standards, Second Edition,
Prentice Hall, pp 297-309,
2003
[9]
D. Song, G. Shaffer and M. Undy,
Nidsbench – A network intrusion detection system test suite,
Anzen Computing RAID99,
1999

[10]
T.H. Ptacek, T.M. Newsham,
Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection,
Secure Networks Inc.,
1998

[11]
W. Stallings,
Data & Computer Communications, Sixth Edition,
Prentice Hall, pp 528-529,
2000

[12]
W. R. Stevens,
TCP/IP Illustrated, Volume 1, The protocols,
Addison Wesley, pp 54-60,
2000

[13]
B. Laing,
How To Guide-Implementing a Network Based Intrusion Detection System,
Internet Security Systems,
2000

[14]
R. Bace, P. Mell,
Intrusion Detection Systems,
NIST Special Publication on Intrusion Detection Systems,
2000

[15]
P. Mell,V. Hu, R. Lippmann, J. Haines, M. Zissman,
An Overview of Issues in Testing Intrusion Detection Systems,
National Institute of Standards and Technology and Massachusetts Institute of Technology Lincoln Laboratory,
2003

[16]
M.J. Ranum,
False Positives: A User's Guide to Making Sense of IDS Alarms,
ICSA Labs IDSC,
2003

[17]
S. Axelsson,
On A difficulty in Intrusion Detection,
Recent Advances in Intrusion Detection
1999,
http://citeseer.nj.nec.com/axelsson99difficulty.html

[18]
C. Del Carlo,
Intrusion detection evasion: How Attacker get past the burglar alarm,
SANS Institute,
2003

[19]
S. Axelsson,
Intrusion Detection Systems: A Survey and Taxonomy,
Chalmers University 99-15,
2000,
http://citeseer.nj.nec.com/axelsson00intrusion.html

[20]
S. Axelsson,
Research in Intrusion-Detection Systems: A survey,
Chalmers University of Technology,
1998, revisited 1999

[21]
N. Athanasiades, R. Abler, J. Levine, H. Owen and G. Riley
Intrusion Detection Testing and Benchmarking Methodologies,
First IEEE International Information Assurance Workshop,
2003

[22]
R. Anderson,
Why Information Security is Hard,
University of Cambridge Computer Laboratory,
2001,
http://www.acsac.org/2001/papers/110.pdf

[23]
M. Vandenwauver, J. Claessens, W. Moreau, C. Vaduva and R. Maier,
Why Enterprises Need More than Firewalls and Intrusion Detection Systems,
IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'99), 16-18 June 1999, Stanford, CA, USA,
1999

[24]
S. Lodin,
Intrusion Detection Product Evaluation Criteria,
Lodin, S. (1998). Intrusion Detection Product Evaluation Criteria,
1998,
http://citeseer.nj.nec.com/lodin98intrusion.html

[25]
CC – Common Criteria for Information Security Evaluation,
Common Criteria homepage,
http://csrc.nist.gov/cc/

[26]
Morten Srene,
Vitenskapelig forfatterskap - Hvordan lykkes med skriftlige studentoppgaver,
Kolle forlag, ISBN 82-463-0016-4, pp 71-112,
1999

[27]
R. Maxion and K. Tan,
Benchmarking Anomaly-Based Detection Systems,
In Proceedings of the 1st International Conference on Dependable Systems & Networks,
2000,
http://citeseer.nj.nec.com/maxion00benchmarking.html

[28]
M. Roesch, C. Green,
Snort Users Manual,
2004,
http://www.snort.org/docs/snort_manual.pdf

[29]
ACID,
Analysis Console for Intrusion Databases (ACID) Website,
http://www.andrew.cmu.edu/user/rdanyliw/snort/snortacid.html

[30]
Einar Snekkenes,
Lecture notes for a course on Security metrics,
NISlab,
2003

[31]
Neohapsis IDS Archives for IDS mailing lists
http://archives.neohapsis.com/archives/ids/

Definitions and commonly used words

This is a list of words used in this thesis, which need further explanations of what they mean in this context. Some of the descriptions has been found on http://www.google.com using its "define:keyword" search string.

Keyword	***Description***
Attacker	A person that performs or try to perform a an attack to gain unauthorized privileges or information on a remote system, based on Confidentiality, Integrity and Availability
Benchmark	A method of standardized quantitative measurement, often performance of some object. Test is also used here as a synonym, enhancing language variation.
DMZ	DeMilitarized Zone. Network of part of a network were the security level is different than from other networks. This may be due to that there are servers running on this network, with different access control than for the rest of the network. Here we might find web servers, mail servers, ftp servers, etc, were inbound traffic is allowed.
DoS	Denial of Service. Attack from one or many hosts with the goal to exhaust resources on a remote system with intention to bring it out of service.
False positive	An intrusion detection that indicates a security event, here an attack, which really is legitimate traffic
False negative	A missed intrusion detection, where an actual attack gets past the IDS without any notice.
Honeypot	A host specially configured with services as a decoy for potential attackers. Often used to inspect attack patterns.
Host	A computer or system often assigned with an Internet address (IP address)
HIDS	Host Intrusion Detection System. IDS running on a host to detect intrusions on that specific machine.
IDS	Intrusion Detection System. A general term used in context with burglar alarms used on computer systems that alerts of any detected attack. In this thesis the term is often used where NIDS or the IDS name would be more correct to use.
Lag	Latency. In network context; Time from a data packet is sent, to it is received by the receiver.
Methodology	A description of steps and procedures to collect and analyze information.

Keyword	***Description***
NIC	Network Interface Card, often Network card. Necessary for network communication (here: Ethernet) among computers.
NIDS	Network Intrusion Detection System
Polymorphic	The ability to change or mutate its appearance so that the characteristics appear different, but its functionality is the same. Used to fool detection engines, like anti virus or IDS.
Sensor	The detecting engine of an IDS. It is located where the it can have access to the physical network. The sensor is often located with the rest of the IDS software or hardware, but there are distributed IDSs with many sensors spread out on the network, with a central host that processes the alerts, the decision module.
System administrator	Person or role which has higher or the highest privileges on a computer or network, responsible for running the computer or network.
Victim	Here defined as the machine running services that a possible attacker will use or abuse to get unauthorized resources. Used here with the term honeypot.
VPN	Virtual Private Network. A logical network of computers talking to each other across physical networks, using encryption to ensure authentication, confidentiality and integrity.

Appendix A: Test results

A.1 Introduction

This chapter presents the results from the IDS tests from the experiment. The results are organized in tables for increasing readability. Each result from a configuration scheme is placed in its own table. If needed, strange or unclear results will be commented. The "Type" field has intentionally been left blank, to underline the difficulties of classifying the the alerts as attack or false positive, or even false negatives.

A.2 Configuration scheme 1

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Nmap, 3.48 (S 1)			
nmap <IP address>	5,05	7	
nmap -sF <IP address>	47,83	113	
nmap -O <IP address>	6,65	9	
Notes:			
Nessus, 2.0.10 (S 2)			
All plugins enabled except the ones that could be harmful to the service on the victim	8,54	890	
All plugins enabled, even the harmful ones	23,57	1328	
Notes: The second test crashed the remote machine at first try.			
Snot, 0.92a (S 3)			
`snot -r rules/all.rules -s <source IP> - <destination IP> -l 5`	0,00	15	
Notes: all.rules is all the signatures used by snort (`cat rules/*.rules > rules/all.rules`). Not all signatures where approved by snot, probably because of that snot is quite old and not 100% compatible with todays version of Snort.10 minute run.			
Sneeze, 1.0 (S 4)			

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`sneeze.pl -f rules/all.rules -s <source IP> - d <destination IP>`	-	-	
Notes: This software was impossible to make it work. When starting it, it used all CPU cycles but sent no packets. It is a Perl script and several attempts to fix it didn't succeed.			
Nikto, 1.32 (S 5)			
`nikto.pl -Cgidirs -generic -host <IP> -output output1.txt -verbose`	33,51	124	
Next commands using Whiskers IDS evasion			
`nikto.pl -Cgidirs -generic -host <IP> -output output1e1.txt -verbose -evasion 1`	5,86	76	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e2.txt -verbose -evasion 2`	28,96	183	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e3.txt -verbose -evasion 3`	45,35	494	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e4.txt -verbose -evasion 4`	42,93	327	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e5.txt -verbose -evasion 5`	44,93	332	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e6.txt -verbose -evasion 6`	28,72	202	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e7.txt -verbose -evasion 7`	28,77	223	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e8.txt -verbose -evasion 8`	24,91	205	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e9.txt -verbose -evasion 9`	0	3998	
Notes:			
Fragroute, 1.2 (S 6)			
`fragroute <destination IP>`			
`snot -r rules/all.rules -s <source IP> - d <destination IP> -l 5`	47,40	661	
`nmap <IP address>`	0,00	11	
`nikto.pl -Cgidirs -generic -host <IP> -output output1f.txt -verbose`	45,75	12518	
Notes: Fragroute rewrites traffic and is dependent on other tools output.			
Fragrouter, 1.2 (S 7)		-	
`fragrouter -B1 -host <IP address>`		-	
`fragrouter -F1 -host <IP address>`		-	

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`fragrouter -F2 -host <IP address>`		-	
`fragrouter -F3 -host <IP address>`		-	
`fragrouter -F4 -host <IP address>`		-	
`fragrouter -F5 -host <IP address>`		-	
`fragrouter -F6 -host <IP address>`		-	
`fragrouter -F7 -host <IP address>`		-	
Notes: Fragrouter is routing and mixing network traffic to a host, so that the attacks must run from a host, through the host running Fragrouter. Fragrouter mixes the attacks in 8 different ways to evade IDSs. In my test setting a became short of a machine, and therefore it was not possible to run this software in a reasonable setting. A proposal for use, would be to set up Fragrouter in 8 modes, running one ore more of the tools for each mode. According to OSEC [3], the Fragrouter tests are vital and should be accomplished.			
IDSWakeup, 1.0 (S 8)			
`IDSwakeup <source IP> <destination IP> 10 5`	40,07	43	
`IDSwakeup <source IP> <destination IP> 10 0`	44,35	39	
`IDSwakeup <source IP> <destination IP> 10 1`	36,76	54	
`IDSwakeup <source IP> <destination IP> 10 2`	40,54	50	
Notes: IDSwakep with altered TTL (Time-To-Live) field. The software package also contains iwu that can send a buffer as a datagram. This is useful when testing for attack signatures. Did not find this useful here, since other software does this in easier ways. Se documentation for usage.			
ADMmutate, 0.84 (S 9)			

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***

API for creating polymorphic shellcode. Apply header and call functions from the original shellcode:
`#include "ADMmutapi.h"`
`struct morphctl *mctlp;`
`struct morphctl mut;`
`mut.upper = 0; mut.lower = 0;`
`mut.banned = 0; mctlp = &mut;`
`mut.arch = IA32; // Intel x86`
`...`
`init_mutate(mctlp);`
`apply_key(buff, strlen(shellcode), nops-1, mctlp);`
`apply_jnops(buff, nops-1, mut);`
`apply_engine(buff, strlen(shellcode), nops-1, mut);`
`...`			

Intended use with Apache 1.3.X Remote Exploit		-	
Notes: Though this should be relatively straight-forward to implement in the shellcode, according to the documentation, this was not the case. It was impossible to compile the exploit with the polymorphic code lines. The errors originates from the API, and several attempts on fixing the problem(s) failed.			
Shellcode exploit, (S 10) (Apache 1.3.x – 2.0.48 remote users disclosure)			
`apacheexploit -t <destination IP> -p 80 -u userlist.txt -l log1.txt -b`	0,00	0	
Notes: Compile: `gcc m00-apache-w00t.c -o apacheexploit` Target was not vulnerable, but the exploit was detected. Userlist.txt is a list of users to remotely disclose. Here the exploit ran 3 times without any detection!			
Shellcode exploit, (S 11) (Apache 1.3.x remote exploit)			
`apacheexploit 3 <destination IP>:<destination port>`	0,00	21	
Notes: Compile: `gcc 01-apache-scalp.c -o apacherexploit` First parameter indicates operating system / web server. Intended for OpenBSD but works on Linux			
7plagues, (S 12)			

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`7plagues.pl <destination IP> <protocol>`		-	
Notes: Protocol is icmp, udp, tcp, igmp, misc. This is a Perl script that was impossible to get to work. It is written with the old threading system of Perl, and was after many, many attempts with different library packages impossible to get it to work.			
ISIC, 0.05 (S 12)			
`icmpsic -s rand -d <destination IP> -I 100 -p 5400000 -m 8192 -r 342536` 1)	49,94	9106	
`icmpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 2)	49,95	1304	
`tcpsic -s rand -d <destination IP>,80 -I 100 -T 100 -p 1000000 -m 8192 -r 342536` 3)	49,94	1468	
`tcpsic -s rand -d <destination IP>,80 -p 1000000 -m 8192 -r 342536` 4)	49,92	1506	
`udpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 5)	49,91	2042	
`isic -s rand -d <destination IP> -F50 -I10 -p 1000000 -m 8192 -r 342536` 6)	49,87	1411	
Notes: 1-2) 5 400 000 packet are to exhausting for the low capacity machines. Therefore 1 000 000 packets are chosen. 4 Mbit/s throughput. The requirements are described in OSEC [3]. Machines with low capacity is the reason for this. 3) 3,7 Mbit/s throughput. 4) 3,7 Mbit/s 5) 3,4 Mbit/s 6) 3,8 Mbit/s with 50% fragmentation, 10% random IP header length			

A.3 Configuration scheme 2

Software and command options (Config 2)	*Packet drop rate (%)*	*Results (Alert)*	*Type*
Nmap, 3.48 (S 1)			
nmap <IP address>	4,21	8	
nmap -sF <IP address>	47,52	144	
nmap -O <IP address>	3,18	11	
Notes:			
Nessus, 2.0.10 (S 2)			
All plugins enabled except the ones that could be harmful to the service on the victim	20,91	924	
All plugins enabled, even the harmful ones	36,93	2571	
Notes:			
Snot, 0.92a (S 3)			
`snot -r rules/all.rules -s <source IP> - <destination IP> -l 5`	0,00	23	
Notes: all.rules is all the signatures used by snort (`cat rules/*.rules > rules/all.rules`). Not all signatures where approved by snot, probably because of that snot is quite old and not 100% compatible with todays version of Snort.10 minute run.			
Sneeze, 1.0 (S 4)			
`sneeze.pl -f rules/all.rules -s <source IP> - d <destination IP>`	-	-	
Notes: This software was impossible to make it work. When starting it, it used all CPU cycles but sent no packets. It is a Perl script and several attempts to fix it didn't succeed.			
Nikto, 1.32 (S 5)			
`nikto.pl -Cgidirs -generic -host <IP> -output output2.txt -verbose`	33,21	116	
Next commands using Whiskers IDS evasion			
`nikto.pl -Cgidirs -generic -host <IP> -output output2e1.txt -verbose -evasion 1`	11,54	69	
`nikto.pl -Cgidirs -generic -host <IP> -output output2e2.txt -verbose -evasion 2`	21,46	305	

Software and command options (Config 2)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`nikto.pl -Cgidirs -generic -host <IP> -output output2e3.txt -verbose -evasion 3`	45,06	326	
`nikto.pl -Cgidirs -generic -host <IP> -output output2e4.txt -verbose -evasion 4`	40,07	432	
`nikto.pl -Cgidirs -generic -host <IP> -output output2e5.txt -verbose -evasion 5`	44,38	334	
`nikto.pl -Cgidirs -generic -host <IP> -output output2e6.txt -verbose -evasion 6`	31,11	166	
`nikto.pl -Cgidirs -generic -host <IP> -output output2e7.txt -verbose -evasion 7`	28,57	191	
`nikto.pl -Cgidirs -generic -host <IP> -output output2e8.txt -verbose -evasion 8`	27,21	188	
`nikto.pl -Cgidirs -generic -host <IP> -output output2e9.txt -verbose -evasion 9`	0,00	3897	
Notes:			
Fragroute, 1.2 (S 6)			
`fragroute <destination IP>`			
`snot -r rules/all.rules -s <source IP> - d <destination IP> -l 5`	46,32	1956	
`nmap <IP address>`	0,00	14	
`nikto.pl -Cgidirs -generic -host <IP> -output output2f.txt -verbose`	44,44	12968	
Notes: Fragroute rewrites traffic and is dependent on other tools output.			
Fragrouter, 1.2 (S 7)		-	
`fragrouter -B1 -host <IP address>`		-	
`fragrouter -F1 -host <IP address>`		-	
`fragrouter -F2 -host <IP address>`		-	
`fragrouter -F3 -host <IP address>`		-	
`fragrouter -F4 -host <IP address>`		-	
`fragrouter -F5 -host <IP address>`		-	
`fragrouter -F6 -host <IP address>`		-	
`fragrouter -F7 -host <IP address>`		-	

Software and command options (Config 2)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Fragrouter is routing and mixing network traffic to a host, so that the attacks must run from a host, through the host running Fragrouter. Fragrouter mixes the attacks in 8 different ways to evade IDSs. In my test setting a became short of a machine, and therefore it was not possible to run this software in a reasonable setting. A proposal for use, would be to set up Fragrouter in 8 modes, running one ore more of the tools for each mode. According to OSEC [3], the Fragrouter tests are vital and should be accomplished.			
IDSWakeup, 1.0 (S 8)			
`IDSwakeup <source IP> <destination IP> 10 5`	38,96	36	
`IDSwakeup <source IP> <destination IP> 10 0`	44,81	36	
`IDSwakeup <source IP> <destination IP> 10 1`	39,28	48	
`IDSwakeup <source IP> <destination IP> 10 2`	39,33	36	
Notes: IDSwakep with altered TTL (Time-To-Live) field. The software package also contains iwu that can send a buffer as a datagram. This is useful when testing for attack signatures. Did not find this useful here, since other software does this in easier ways. Se documentation for usage.			
ADMmutate, 0.84 (S 9)			

API for creating polymorphic shellcode. Apply header and call functions from the original shellcode:
`#include "ADMmutapi.h"`
`struct morphctl *mctlp;`
`struct morphctl mut;`
`mut.upper = 0; mut.lower = 0;`
`mut.banned = 0; mctlp = &mut;`
`mut.arch = IA32; // Intel x86`
`...`
`init_mutate(mctlp);`
`apply_key(buff, strlen(shellcode), nops-1, mctlp);`
`apply_jnops(buff, nops-1, mut);`
`apply_engine(buff, strlen(shellcode), nops-1, mut);`
`...`			

Intended use with Apache 1.3.X Remote Exploit		-	

Software and command options (Config 2)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Though this should be relatively straight-forward to implement in the shellcode, according to the documentation, this was not the case. It was impossible to compile the exploit with the polymorphic code lines. The errors originates from the API, and several attempts on fixing the problem(s) failed.			
Shellcode exploit, (S 10) (Apache 1.3.x – 2.0.48 remote users disclosure)			
`apacheexploit -t <destination IP> -p 80 -u userlist.txt -l log2.txt -b`	0,00	0	
Notes: Compile: `gcc m00-apache-w00t.c -o apacheexploit` Target was not vulnerable, but the exploit was detected. Userlist.txt is a list of users to remotely disclose. Here the exploit ran 3 times without any detection!			
Shellcode exploit, (S 11) (Apache 1.3.x remote exploit)			
`apacheexploit 3 <destination IP>:<destination port>`	0,00	21	
Notes: Compile: `gcc 01-apache-scalp.c -o apacherexploit` First parameter indicates operating system / web server. Intended for OpenBSD but works on Linux			
7plagues, (S 12)			
`7plagues.pl <destination IP> <protocol>`		-	
Notes: Protocol is icmp, udp, tcp, igmp, misc. This is a Perl script that was impossible to get to work. It is written with the old threading system of Perl, and was after many, many attempts with different library packages impossible to get it to work.			
ISIC, 0.05 (S 12)			
`icmpsic -s rand -d <destination IP> -I 100 -p 5400000 -m 8192 -r 342536` 1)	49,93	9402	
`icmpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 2)	49,92	1950	
`tcpsic -s rand -d <destination IP>,80 -I 100 -T 100 -p 1000000 -m 8192 -r 342536` 3)	49,92	2135	

Software and command options (Config 2)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`tcpsic -s rand -d <destination IP>,80 -p 1000000 -m 8192 -r 342536` 4)	49,90	2158	
`udpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 5)	49,88	2730	
`isic -s rand -d <destination IP> -F50 -I10 -p 1000000 -m 8192 -r 342536` 6)	49,85	1729	
Notes: 1-2) 5 400 000 packet are to exhausting for the low capacity machines. Therefore 1 000 000 packets are chosen. 4 Mbit/s throughput. The requirements are described in OSEC [3]. Machines with low capacity is the reason for this. 3) 3,4 Mbit/s throughput. 4) 3,4 Mbit/s 5) 3,4 Mbit/s 6) 3,8 Mbit/s with 50% fragmentation, 10% random IP header length			

A.4 Configuration scheme 3

Software and command options (Config 3)	*Packet drop rate (%)*	*Results (Alert)*	*Type*
Nmap, 3.48 (S 1)			
nmap <IP address>	0,00	0	
nmap -sF <IP address>	0,00	0	
nmap -O <IP address>	0,00	0	
Notes:			
Nessus, 2.0.10 (S 2)			
All plugins enabled except the ones that could be harmful to the service on the victim	2,20	10	
All plugins enabled, even the harmful ones	2,03	52	
Notes: The preprocessors in Snort [28] are responsible for detections, since signatures are disabled			
Snot, 0.92a (S 3)			
`snot -r rules/all.rules -s <source IP> - <destination IP> -l 5`	0,00	0	
Notes: all.rules is all the signatures used by snort (`cat rules/*.rules > rules/all.rules`). Not all signatures where approved by snot, probably because of that snot is quite old and not 100% compatible with todays version of Snort.10 minute run. Nothing detected since Snot uses the Snort signatures to generate false positives			
Sneeze, 1.0 (S 4)			
`sneeze.pl -f rules/all.rules -s <source IP> - d <destination IP>`	-	-	
Notes: This software was impossible to make it work. When starting it, it used all CPU cycles but sent no packets. It is a Perl script and several attempts to fix it didn't succeed.			
Nikto, 1.32 (S 5)			
`nikto.pl -Cgidirs -generic -host <IP> -output output3.txt -verbose`	0,82	0	
Next commands using Whiskers IDS evasion			

Software and command options (Config 3)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`nikto.pl -Cgidirs -generic -host <IP> -output output3e1.txt -verbose -evasion 1`	0,00	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e2.txt -verbose -evasion 2`	0,00	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e3.txt -verbose -evasion 3`	0,00	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e4.txt -verbose -evasion 4`	0,02	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e5.txt -verbose -evasion 5`	0,00	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e6.txt -verbose -evasion 6`	0,10	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e7.txt -verbose -evasion 7`	0,00	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e8.txt -verbose -evasion 8`	0,00	0	
`nikto.pl -Cgidirs -generic -host <IP> -output output3e9.txt -verbose -evasion 9`	0,00	0	
Fragroute, 1.2 (S 6)			
`fragroute <destination IP>`			
`snot -r rules/all.rules -s <source IP> - d <destination IP> -l 5`	42,75	302	
`nmap <IP address>`	0,00	1	
`nikto.pl -Cgidirs -generic -host <IP> -output output3f.txt -verbose`	42,99	1372	
Notes: Fragroute rewrites traffic and is dependent on other tools output. Results here are due to header errors in TCP and IP protocols, not a signature detection			
Fragrouter, 1.2 (S 7)		-	
`fragrouter -B1 -host <IP address>`		-	
`fragrouter -F1 -host <IP address>`		-	
`fragrouter -F2 -host <IP address>`		-	
`fragrouter -F3 -host <IP address>`		-	
`fragrouter -F4 -host <IP address>`		-	
`fragrouter -F5 -host <IP address>`		-	
`fragrouter -F6 -host <IP address>`		-	
`fragrouter -F7 -host <IP address>`		-	

Software and command options (Config 3)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Fragrouter is routing and mixing network traffic to a host, so that the attacks must run from a host, through the host running Fragrouter. Fragrouter mixes the attacks in 8 different ways to evade IDSs. In my test setting a became short of a machine, and therefore it was not possible to run this software in a reasonable setting. A proposal for use, would be to set up Fragrouter in 8 modes, running one ore more of the tools for each mode. According to OSEC [3], the Fragrouter tests are vital and should be accomplished.			
IDSWakeup, 1.0 (S 8)			
`IDSwakeup <source IP> <destination IP> 10 5`	39,90	36	
`IDSwakeup <source IP> <destination IP> 10 0`	38,80	51	
`IDSwakeup <source IP> <destination IP> 10 1`	39,22	49	
`IDSwakeup <source IP> <destination IP> 10 2`	39,10	61	
Notes: IDSwakep with altered TTL (Time-To-Live) field. The software package also contains iwu that can send a buffer as a datagram. This is useful when testing for attack signatures. Did not find this useful here, since other software does this in easier ways. Se documentation for usage.			
ADMmutate, 0.84 (S 9)			

API for creating polymorphic shellcode. Apply header and call functions from the original shellcode:
`#include "ADMmutapi.h"`
`struct morphctl *mctlp;`
`struct morphctl mut;`
`mut.upper = 0; mut.lower = 0;`
`mut.banned = 0; mctlp = &mut;`
`mut.arch = IA32; // Intel x86`
`...`
`init_mutate(mctlp);`
`apply_key(buff, strlen(shellcode), nops-1, mctlp);`
`apply_jnops(buff, nops-1, mut);`
`apply_engine(buff, strlen(shellcode), nops-1, mut);`
`...`			

Intended use with Apache 1.3.X Remote Exploit		-	

Software and command options (Config 3)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Though this should be relatively straight-forward to implement in the shellcode, according to the documentation, this was not the case. It was impossible to compile the exploit with the polymorphic code lines. The errors originates from the API, and several attempts on fixing the problem(s) failed.			
Shellcode exploit, (S 10) (Apache 1.3.x – 2.0.48 remote users disclosure)			
`apacheexploit -t <destination IP> -p 80 -u userlist.txt -l log3.txt -b`	28,89	0	
Notes: Compile: `gcc m00-apache-w00t.c -o apacheexploit` Target was not vulnerable, but the exploit was detected. Userlist.txt is a list of users to remotely disclose.			
Shellcode exploit, (S 11) (Apache 1.3.x remote exploit)			
`apacheexploit 3 <destination IP>:<destination port>`	0,00	0	
Notes: Compile: `gcc 01-apache-scalp.c -o apacherexploit` First parameter indicates operating system / web server. Intended for OpenBSD but works on Linux			
7plagues, (S 12)			
`7plagues.pl <destination IP> <protocol>`		-	
Notes: Protocol is icmp, udp, tcp, igmp, misc. This is a Perl script that was impossible to get to work. It is written with the old threading system of Perl, and was after many, many attempts with different library packages impossible to get it to work.			
ISIC, 0.05 (S 12)			
`icmpsic -s rand -d <destination IP> -I 100 -p 5400000 -m 8192 -r 342536` 1)	49,92	9581	
`icmpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 2)	49,89	2450	
`tcpsic -s rand -d <destination IP>,80 -I 100 -T 100 -p 1000000 -m 8192 -r 342536` 3)	49,94	1493	
`tcpsic -s rand -d <destination IP>,80 -p 1000000 -m 8192 -r 342536` 4)	49,92	1536	

Software and command options (Config 3)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`udpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 5)	49,91	1996	
`isic -s rand -d <destination IP> -F50 -I10 -p 1000000 -m 8192 -r 342536` 6)	49,59	1651	
Notes: 1-2) 5 400 000 packet are to exhausting for the low capacity machines. Therefore 1 000 000 packets are chosen. 4 Mbit/s throughput. The requirements are described in OSEC [3]. Machines with low capacity is the reason for this. 3) 3,7 Mbit/s throughput. 4) 3,7 Mbit/s 5) 3,4 Mbit/s 6) 3,8 Mbit/s with 50% fragmentation, 10% random IP header length			

A.5 Configuration scheme 4

Software and command options (Config 4)	*Packet drop rate (%)*	*Results (Alert)*	*Type*
Nmap, 3.48 (S 1)			
nmap <IP address>	21,23	6	
nmap -sF <IP address>	47,72	110	
nmap -O <IP address>	25,40	7	
Notes:			
Nessus, 2.0.10 (S 2)			
All plugins enabled except the ones that could be harmful to the service on the victim	10,98	821	
All plugins enabled, even the harmful ones	35,19	1603	
Notes:			
Snot, 0.92a (S 3)			
`snot -r rules/all.rules -s <source IP> - <destination IP> -l 5`	0,00	12	
Notes: all.rules is all the signatures used by snort (`cat rules/*.rules > rules/all.rules`). Not all signatures where approved by snot, probably because of that snot is quite old and not 100% compatible with todays version of Snort.10 minute run. Nothing detected since Snot uses the Snort signatures to generate false positives.			
Sneeze, 1.0 (S 4)			
`sneeze.pl -f rules/all.rules -s <source IP> - d <destination IP>`	-	-	
Notes: This software was impossible to make it work. When starting it, it used all CPU cycles but sent no packets. It is a Perl script and several attempts to fix it didn't succeed.			
Nikto, 1.32 (S 5)			
`nikto.pl -Cgidirs -generic -host <IP> -output output4.txt -verbose`	32,62	134	
Next commands using Whiskers IDS evasion			
`nikto.pl -Cgidirs -generic -host <IP> -output output4e1.txt -verbose -evasion 1`	3,35	84	

Software and command options (Config 4)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`nikto.pl -Cgidirs -generic -host <IP> -output output4e2.txt -verbose -evasion 2`	32,74	147	
`nikto.pl -Cgidirs -generic -host <IP> -output output4e3.txt -verbose -evasion 3`	46,43	241	
`nikto.pl -Cgidirs -generic -host <IP> -output output4e4.txt -verbose -evasion 4`	45,93	266	
`nikto.pl -Cgidirs -generic -host <IP> -output output4e5.txt -verbose -evasion 5`	46,20	254	
`nikto.pl -Cgidirs -generic -host <IP> -output output4e6.txt -verbose -evasion 6`	31,18	146	
`nikto.pl -Cgidirs -generic -host <IP> -output output4e7.txt -verbose -evasion 7`	28,51	198	
`nikto.pl -Cgidirs -generic -host <IP> -output output4e8.txt -verbose -evasion 8`	29,12	164	
`nikto.pl -Cgidirs -generic -host <IP> -output output4e9.txt -verbose -evasion 9`		-	
`Notes: Evasion 9 skipped due to extensive time use.`			
Fragroute, 1.2 (S 6)			
`fragroute <destination IP>`			
`snot -r rules/all.rules -s <source IP> - d <destination IP> -l 5`	45,86	964	
`nmap <IP address>`	0,00	12	
`nikto.pl -Cgidirs -generic -host <IP> -output output4f.txt -verbose`	46,11	11149	
Notes: Fragroute rewrites traffic and is dependent on other tools output.			
Fragrouter, 1.2 (S 7)		-	
`fragrouter -B1 -host <IP address>`		-	
`fragrouter -F1 -host <IP address>`		-	
`fragrouter -F2 -host <IP address>`		-	
`fragrouter -F3 -host <IP address>`		-	
`fragrouter -F4 -host <IP address>`		-	
`fragrouter -F5 -host <IP address>`		-	
`fragrouter -F6 -host <IP address>`		-	
`fragrouter -F7 -host <IP address>`		-	

Software and command options (Config 4)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Fragrouter is routing and mixing network traffic to a host, so that the attacks must run from a host, through the host running Fragrouter. Fragrouter mixes the attacks in 8 different ways to evade IDSs. In my test setting a became short of a machine, and therefore it was not possible to run this software in a reasonable setting. A proposal for use, would be to set up Fragrouter in 8 modes, running one ore more of the tools for each mode. According to OSEC [3], the Fragrouter tests are vital and should be accomplished.			
IDSWakeup, 1.0 (S 8)			
`IDSwakeup <source IP> <destination IP> 10 5`	36,83	56	
`IDSwakeup <source IP> <destination IP> 10 0`	40,35	39	
`IDSwakeup <source IP> <destination IP> 10 1`	38,35	39	
`IDSwakeup <source IP> <destination IP> 10 2`	36,44	67	
Notes: IDSwakep with altered TTL (Time-To-Live) field. The software package also contains iwu that can send a buffer as a datagram. This is useful when testing for attack signatures. Did not find this useful here, since other software does this in easier ways. Se documentation for usage.			
ADMmutate, 0.84 (S 9)			

API for creating polymorphic shellcode. Apply header and call functions from the original shellcode:
`#include "ADMmutapi.h"`
`struct morphctl *mctlp;`
`struct morphctl mut;`
`mut.upper = 0; mut.lower = 0;`
`mut.banned = 0; mctlp = &mut;`
`mut.arch = IA32; // Intel x86`
`...`
`init_mutate(mctlp);`
`apply_key(buff, strlen(shellcode), nops-1, mctlp);`
`apply_jnops(buff, nops-1, mut);`
`apply_engine(buff, strlen(shellcode), nops-1, mut);`
`...`			

Intended use with Apache 1.3.X Remote Exploit		-	

Software and command options (Config 4)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Though this should be relatively straight-forward to implement in the shellcode, according to the documentation, this was not the case. It was impossible to compile the exploit with the polymorphic code lines. The errors originates from the API, and several attempts on fixing the problem(s) failed.			
Shellcode exploit, (S 10) (Apache 1.3.x – 2.0.48 remote users disclosure)			
`apacheexploit -t <destination IP> -p 80 -u userlist.txt -l log4.txt -b`	0,00	0	
Notes: Compile: `gcc m00-apache-w00t.c -o apacheexploit` Target was not vulnerable, but the exploit was detected. Userlist.txt is a list of users to remotely disclose. Here the exploit ran 3 times without any detection!			
Shellcode exploit, (S 11) (Apache 1.3.x remote exploit)			
`apacheexploit 3 <destination IP>:<destination port>`	0,00	21	
Notes: Compile: `gcc 01-apache-scalp.c -o apacherexploit` First parameter indicates operating system / web server. Intended for OpenBSD but works on Linux			
7plagues, (S 12)			
`7plagues.pl <destination IP> <protocol>`		-	
Notes: Protocol is icmp, udp, tcp, igmp, misc. This is a Perl script that was impossible to get to work. It is written with the old threading system of Perl, and was after many, many attempts with different library packages impossible to get it to work.			
ISIC, 0.05 (S 12)			
`icmpsic -s rand -d <destination IP> -I 100 -p 5400000 -m 8192 -r 342536` 1)	49,94	7870	
`icmpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 2)	49,94	1358	
`tcpsic -s rand -d <destination IP>,80 -I 100 -T 100 -p 1000000 -m 8192 -r 342536` 3)	49,94	1542	

Software and command options (Config 4)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`tcpsic -s rand -d <destination IP>,80 -p 1000000 -m 8192 -r 342536` 4)	49,91	1759	
`udpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 5)	49,91	2078	
`isic -s rand -d <destination IP> -F50 -I10 -p 1000000 -m 8192 -r 342536` 6)	49,86	1702	
Notes: 1-2) 5 400 000 packet are to exhausting for the low capacity machines. Therefore 1 000 000 packets are chosen. 4 Mbit/s throughput. The requirements are described in OSEC [3]. Machines with low capacity is the reason for this. 3) 3,4 Mbit/s throughput. 4) 3,4 Mbit/s 5) 3,4 Mbit/s 6) 3,8 Mbit/s with 50% fragmentation, 10% random IP header length			

A.6 Configuration scheme 5

Software and command options (Config 5)	*Packet drop rate (%)*	*Results (Alert)*	*Type*
Nmap, 3.48 (S 1)			
nmap <IP address>	3,61	8	
nmap -sF <IP address>	47,49	126	
nmap -O <IP address>	1,77	10	
Notes:			
Nessus, 2.0.10 (S 2)			
All plugins enabled except the ones that could be harmful to the service on the victim	10,18	844	
All plugins enabled, even the harmful ones	34,77	1654	
Notes: The second test crashed the remote machine at first try.			
Snot, 0.92a (S 3)			
`snot -r rules/all.rules -s <source IP> - <destination IP> -l 5`	0,00	20	
Notes: all.rules is all the signatures used by snort (`cat rules/*.rules > rules/all.rules`). Not all signatures where approved by snot, probably because of that snot is quite old and not 100% compatible with todays version of Snort.10 minute run.			
Sneeze, 1.0 (S 4)			
`sneeze.pl -f rules/all.rules -s <source IP> - d <destination IP>`	-	-	
Notes: This software was impossible to make it work. When starting it, it used all CPU cycles but sent no packets. It is a Perl script and several attempts to fix it didn't succeed.			
Nikto, 1.32 (S 5)			
`nikto.pl -Cgidirs -generic -host <IP> -output output5.txt -verbose`	28,77	181	
Next commands using Whiskers IDS evasion			
`nikto.pl -Cgidirs -generic -host <IP> -output output5e1.txt -verbose -evasion 1`	0,73	95	

Software and command options (Config 5)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`nikto.pl -Cgidirs -generic -host <IP> -output output5e2.txt -verbose -evasion 2`	26,70	202	
`nikto.pl -Cgidirs -generic -host <IP> -output output5e3.txt -verbose -evasion 3`	45,28	312	
`nikto.pl -Cgidirs -generic -host <IP> -output output5e4.txt -verbose -evasion 4`	44,20	294	
`nikto.pl -Cgidirs -generic -host <IP> -output output5e5.txt -verbose -evasion 5`	45,00	331	
`nikto.pl -Cgidirs -generic -host <IP> -output output5e6.txt -verbose -evasion 6`	29,08	200	
`nikto.pl -Cgidirs -generic -host <IP> -output output5e7.txt -verbose -evasion 7`	27,24	211	
`nikto.pl -Cgidirs -generic -host <IP> -output output5e8.txt -verbose -evasion 8`	25,25	208	
`nikto.pl -Cgidirs -generic -host <IP> -output output5e9.txt -verbose -evasion 9`	-	-	
Notes: Evasion 9 skipped due to extensive time use.			
Fragroute, 1.2 (S 6)			
`fragroute <destination IP>`			
`snot -r rules/all.rules -s <source IP> - d <destination IP> -l 5`	46,74	799	
`nmap <IP address>`	0,00	11	
`nikto.pl -Cgidirs -generic -host <IP> -output output5f.txt -verbose`	46,32	11255	
Notes: Fragroute rewrites traffic and is dependent on other tools output.			
Fragrouter, 1.2 (S 7)		-	
`fragrouter -B1 -host <IP address>`		-	
`fragrouter -F1 -host <IP address>`		-	
`fragrouter -F2 -host <IP address>`		-	
`fragrouter -F3 -host <IP address>`		-	
`fragrouter -F4 -host <IP address>`		-	
`fragrouter -F5 -host <IP address>`		-	
`fragrouter -F6 -host <IP address>`		-	
`fragrouter -F7 -host <IP address>`		-	

Software and command options (Config 5)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Fragrouter is routing and mixing network traffic to a host, so that the attacks must run from a host, through the host running Fragrouter. Fragrouter mixes the attacks in 8 different ways to evade IDSs. In my test setting a became short of a machine, and therefore it was not possible to run this software in a reasonable setting. A proposal for use, would be to set up Fragrouter in 8 modes, running one ore more of the tools for each mode. According to OSEC [3], the Fragrouter tests are vital and should be accomplished.			
IDSWakeup, 1.0 (S 8)			
`IDSwakeup <source IP> <destination IP> 10 5`	38,32	36	
`IDSwakeup <source IP> <destination IP> 10 0`	39,67	56	
`IDSwakeup <source IP> <destination IP> 10 1`	35,87	68	
`IDSwakeup <source IP> <destination IP> 10 2`	40,23	62	
Notes: IDSwakep with altered TTL (Time-To-Live) field. The software package also contains iwu that can send a buffer as a datagram. This is useful when testing for attack signatures. Did not find this useful here, since other software does this in easier ways. Se documentation for usage.			
ADMmutate, 0.84 (S 9)			

API for creating polymorphic shellcode. Apply header and call functions from the original shellcode:
`#include "ADMmutapi.h"`
`struct morphctl *mctlp;`
`struct morphctl mut;`
`mut.upper = 0; mut.lower = 0;`
`mut.banned = 0; mctlp = &mut;`
`mut.arch = IA32; // Intel x86`
`...`
`init_mutate(mctlp);`
`apply_key(buff, strlen(shellcode), nops-1, mctlp);`
`apply_jnops(buff, nops-1, mut);`
`apply_engine(buff, strlen(shellcode), nops-1, mut);`
`...`			

Intended use with Apache 1.3.X Remote Exploit		-	

Software and command options (Config 5)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: Though this should be relatively straight-forward to implement in the shellcode, according to the documentation, this was not the case. It was impossible to compile the exploit with the polymorphic code lines. The errors originates from the API, and several attempts on fixing the problem(s) failed.			
Shellcode exploit, (S 10) (Apache 1.3.x – 2.0.48 remote users disclosure)			
`apacheexploit -t <destination IP> -p 80 -u userlist.txt -l log5.txt -b`	0,00	0	
Notes: Compile: `gcc m00-apache-w00t.c -o apacheexploit` Target was not vulnerable, but the exploit was detected. Userlist.txt is a list of users to remotely disclose. Here the exploit ran 3 times without any detection!			
Shellcode exploit, (S 11) (Apache 1.3.x remote exploit)			
`apacheexploit 3 <destination IP>:<destination port>`	0,00	21	
Notes: Compile: `gcc 01-apache-scalp.c -o apacherexploit` First parameter indicates operating system / web server. Intended for OpenBSD but works on Linux			
7plagues, (S 12)			
`7plagues.pl <destination IP> <protocol>`		-	
Notes: Protocol is icmp, udp, tcp, igmp, misc. This is a Perl script that was impossible to get to work. It is written with the old threading system of Perl, and was after many, many attempts with different library packages impossible to get it to work.			
ISIC, 0.05 (S 12)			
`icmpsic -s rand -d <destination IP> -I 100 -p 5400000 -m 8192 -r 342536` 1)	49,93	9792	
`icmpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 2)	50,00	1358	
`tcpsic -s rand -d <destination IP>,80 -I 100 -T 100 -p 1000000 -m 8192 -r 342536` 3)	49,92	2088	

Software and command options (Config 5)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`tcpsic -s rand -d <destination IP>,80 -p 1000000 -m 8192 -r 342536` 4)	49,90	2085	
`udpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 5)	49,88	2655	
`isic -s rand -d <destination IP> -F50 -I10 -p 1000000 -m 8192 -r 342536` 6)	49,86	1604	
Notes: 1-2) 5 400 000 packet are to exhausting for the low capacity machines. Therefore 1 000 000 packets are chosen. 4 Mbit/s throughput. The requirements are described in OSEC [3]. Machines with low capacity is the reason for this. 3) 3,4 Mbit/s throughput. 4) 3,4 Mbit/s 5) 3,4 Mbit/s 6) 3,8 Mbit/s with 50% fragmentation, 10% random IP header length			

A.7 Configuration scheme 1 *

A.7.1 Introduction

These results have been performed on a faster machine, see chapter 3.3.3. This machine was originally intended to host the IDS, but broke down early in the testing phase. Therefore there exists only results from this machine. The results may be compared with the other results to see the impact of using a low capacity machine.

A.7.2 Results

Software and command options (Config 1)	*Packet drop rate (%)*	*Results (Alert)*	*Type*
Nmap, 3.48 (S 1)			
nmap <IP address>	-	10	
nmap -sF <IP address>	-	-	
nmap -O <IP address>	-	-	
Notes:			
Nessus, 2.0.10 (S 2)			
All plugins enabled except the ones that could be harmful to the service on the victim	-	230	
All plugins enabled, even the harmful ones	-	0	
Notes: Last result must be erroneous			
Snot, 0.92a (S 3)			
`snot -r rules/all.rules -s <source IP> - <destination IP> -l 5`	-	785	
Notes: all.rules is all the signatures used by snort (`cat rules/*.rules > rules/all.rules`). Not all signatures where approved by snot, probably because of that snot is quite old and not 100% compatible with todays version of Snort. 5 hours and 15 minutes run. Results seem very little.			
Sneeze, 1.0 (S 4)			
`sneeze.pl -f rules/all.rules -s <source IP> - d <destination IP>`	-	-	

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
Notes: This software was impossible to make it work. When starting it, it used all CPU cycles but sent no packets. It is a Perl script and several attempts to fix it didn't succeed.			
Nikto, 1.32 (S 5)			
`nikto.pl -Cgidirs -generic -host <IP> -output output2.txt -verbose`	-	635	
Next commands using Whiskers IDS evasion			
`nikto.pl -Cgidirs -generic -host <IP> -output output1e1.txt -verbose -evasion 1`	-	370	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e2.txt -verbose -evasion 2`	-	625	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e3.txt -verbose -evasion 3`	-	1930	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e4.txt -verbose -evasion 4`	-	1537	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e5.txt -verbose -evasion 5`	-	1930	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e6.txt -verbose -evasion 6`	-	1921	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e7.txt -verbose -evasion 7`	-	575	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e8.txt -verbose -evasion 8`	-	604	
`nikto.pl -Cgidirs -generic -host <IP> -output output1e9.txt -verbose -evasion 9`	-	3896	
Notes:			
Fragroute, 1.2 (S 6)			
`fragroute <destination IP>`			
`snot -r rules/all.rules -s <source IP> - d <destination IP> -l 5`	-	11072	
`nmap -sF<IP address>`	-	101	
`nikto.pl -Cgidirs -generic -host <IP> -output output1f.txt -verbose`	-	652	
Notes: Fragroute rewrites traffic and is dependent on other tools output. Suspicious results from Nikto. Seems very low.			
Fragrouter, 1.2 (S 7)		-	
`fragrouter -B1 -host <IP address>`		-	
`fragrouter -F1 -host <IP address>`		-	

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`fragrouter -F2 -host <IP address>`		-	
`fragrouter -F3 -host <IP address>`		-	
`fragrouter -F4 -host <IP address>`		-	
`fragrouter -F5 -host <IP address>`		-	
`fragrouter -F6 -host <IP address>`		-	
`fragrouter -F7 -host <IP address>`		-	
Notes: Fragrouter is routing and mixing network traffic to a host, so that the attacks must run from a host, through the host running Fragrouter. Fragrouter mixes the attacks in 8 different ways to evade IDSs. In my test setting a became short of a machine, and therefore it was not possible to run this software in a reasonable setting. A proposal for use, would be to set up Fragrouter in 8 modes, running one ore more of the tools for each mode. According to OSEC [3], the Fragrouter tests are vital and should be accomplished.			
IDSWakeup, 1.0 (S 8)			
`IDSwakeup <source IP> <destination IP> 10 5`	-	200	
`IDSwakeup <source IP> <destination IP> 10 0`	-	200	
`IDSwakeup <source IP> <destination IP> 10 1`	-	-	
`IDSwakeup <source IP> <destination IP> 10 2`	-	-	
Notes: IDSwakep with altered TTL (Time-To-Live) field. The software package also contains iwu that can send a buffer as a datagram. This is useful when testing for attack signatures. Did not find this useful here, since other software does this in easier ways. Se documentation for usage.			
ADMmutate, 0.84 (S 9)			

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***

API for creating polymorphic shellcode. Apply header and call functions from the original shellcode:
`#include "ADMmutapi.h"`
`struct morphctl *mctlp;`
`struct morphctl mut;`
`mut.upper = 0; mut.lower = 0;`
`mut.banned = 0; mctlp = &mut;`
`mut.arch = IA32; // Intel x86`
`...`
`init_mutate(mctlp);`
`apply_key(buff, strlen(shellcode), nops-1, mctlp);`
`apply_jnops(buff, nops-1, mut);`
`apply_engine(buff, strlen(shellcode), nops-1, mut);`
`...`			

Intended use with Apache 1.3.X Remote Exploit		-	
Notes: Though this should be relatively straight-forward to implement in the shellcode, according to the documentation, this was not the case. It was impossible to compile the exploit with the polymorphic code lines. The errors originates from the API, and several attempts on fixing the problem(s) failed.			
Shellcode exploit, (S 10) (Apache 1.3.x – 2.0.48 remote users disclosure)			
`apacheexploit -t <destination IP> -p 80 -u userlist.txt -l log1.txt -b`	-	1	
Notes: Compile: `gcc m00-apache-w00t.c -o apacheexploit` Target was not vulnerable, but the exploit was detected. Userlist.txt is a list of users to remotely disclose.			
Shellcode exploit, (S 11) (Apache 1.3.x remote exploit)			
`apacheexploit 3 <destination IP>:<destination port>`	-	21	
Notes: Compile: `gcc 01-apache-scalp.c -o apacherexploit` First parameter indicates operating system / web server. Intended for OpenBSD but works on Linux			
7plagues, (S 12)			

Software and command options (Config 1)	***Packet drop rate (%)***	***Results (Alert)***	***Type***
`7plagues.pl <destination IP> <protocol>`		-	
Notes: Protocol is icmp, udp, tcp, igmp, misc. This is a Perl script that was impossible to get to work. It is written with the old threading system of Perl, and was after many, many attempts with different library packages impossible to get it to work.			
ISIC, 0.05 (S 12)			
`icmpsic -s rand -d <destination IP> -I 100 -p 5400000 -m 8192 -r 342536` 1)	-	76581	
`icmpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 2)	-	-	
`tcpsic -s rand -d <destination IP>,80 -I 100 -T 100 -p 1000000 -m 8192 -r 342536` 3)	-	20669	
`tcpsic -s rand -d <destination IP>,80 -p 1000000 -m 8192 -r 342536` 4)	-	16121	
`udpsic -s rand -d <destination IP> -I 100 -p 1000000 -m 8192 -r 342536` 5)	-	23602	
`isic -s rand -d <destination IP> -F50 -I10 -p 1000000 -m 8192 -r 342536` 6)	-	18424	
Notes: 1) The IDS seemed to stop at 2 000 000 packets. 4 Mbit/s throughput. The requirements are described in OSEC [3]. Machines with low capacity is the reason for this. 2) Not evaluated 3) 3,6 Mbit/s throughput. 4) 3,6 Mbit/s 5) 3,4 Mbit/s 6) 3,8 Mbit/s with 50% fragmentation, 10% random IP header length			

Appendix B: Reliability test

B.1 Introduction

This appendix contains a reliability test of the results from the IDS benchmarking. The goal is to give an idea of how accurate the IDS test results are. Reliability is the extent of how a test gives the same result when running the same test several times[1]. To show the reliability of the results form the IDS testing, a tool used in the testing were picked out. It was not a random choice. The set of tools generates different amounts of traffic with different types of attacks witch the IDS triggers on. Some generates large amounts of traffic that stresses the IDS. Other have smaller amounts of traffic. Some tools use more advanced attack patterns often with IDS evasion techniques, while others use more straight-forward attack types. Therefore the tool have to be representable as an average of all tools used in the IDS testing. This is not easy, if possible, to find what tool that generate "average traffic", so some in between was the approach. A tool that didn't generate too much traffic or too little according to the hardware used. Nor should it generate to complex, or to plain attacks. Time consume pr. test was also an important criteria, since the reliability test was to be repeated 40 times to get some statistical value of the numbers.

The choice fell on nmap (S 1). Nmap without any special parameters is to straight forward with fairly few alerts and small amounts of traffic. But by applying an extra attribute, the result was different. It might be generating too much traffic, but relatively short execution time was preferable. The command executed in the reliability test was

```
namp -sF <IP address>
```

where the -sF parameter performs a TCP FIN portscan[2].

The reliability test were done in the same system setting as the IDS testing with Snort as IDS. Default configuration according to configuration scheme 1 were used. The results are presented in the next section. Discussion of the results are presented in the end of this appendix.

1 Based on definition found at http://nces.ed.gov/nceskids/glossary.asp using google's "define:reliability" search string

2 For more information about this parameter and Nmap, see http://www.insecure.org/nmap/

B.2 Reliability test results

Table B.1 : Results using `namp -sF <IP address>`

Test nr	*Packets examined*	*Packets dropped*	*Drop rate in %*	*Nr of Alerts*
1	3479	6742	48,398	90
2	3451	6669	48,253	103
3	3488	6681	47,792	108
4	3446	6674	48,367	96
5	3471	6667	47,938	112
6	3456	6670	48,186	108
7	3477	6680	47,949	116
8	3452	6645	48,051	109
9	3484	6675	47,805	115
10	3460	6668	48,110	108
11	3448	6657	48,205	111
12	3484	6694	47,953	109
13	3463	6669	48,073	104
14	3481	6685	47,928	109
15	3458	6653	48,023	108
16	3480	6716	48,183	99
17	3479	6691	48,005	108
18	3460	6676	48,173	100
19	3491	6713	47,996	103
20	3480	6713	48,160	101
21	3462	6676	48,143	106
22	3455	6649	48,037	113
23	3483	6719	48,162	98
24	3474	6703	48,172	98
25	3477	6707	48,159	98
26	3457	6656	48,002	100
27	3479	6711	48,160	97
28	3465	6632	47,753	122
29	3453	6661	48,161	100

Test nr	*Packets examined*	*Packets dropped*	*Drop rate in %*	*Nr of Alerts*
30	3477	6715	48,220	95
31	3478	6714	48,198	105
32	3474	6699	48,142	102
33	3458	6672	48,171	97
34	3465	6659	47,965	110
35	3451	6637	48,004	111
36	3478	6712	48,182	103
37	3477	6711	48,190	109
38	3473	6654	47,806	118
39	3468	6677	48,061	105
40	3470	6678	48,038	102

Figure B.1: Graphical view of Examined packets by the IDS compared to the average value.

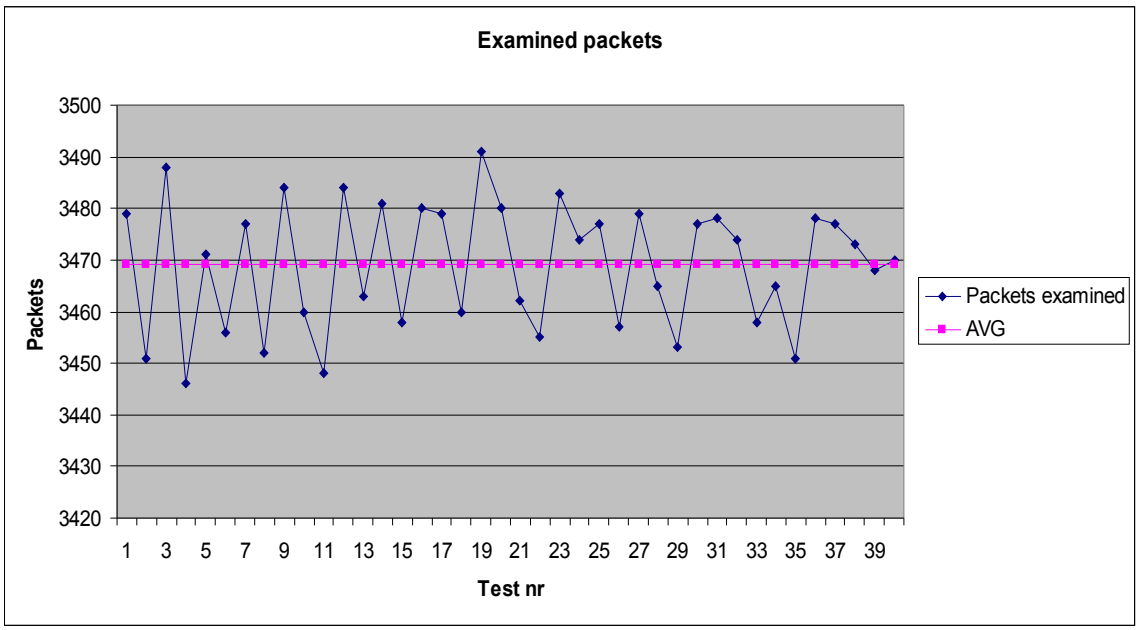


Figure B.2: Graphical view of Packets dropped by the IDS compared to the average value.

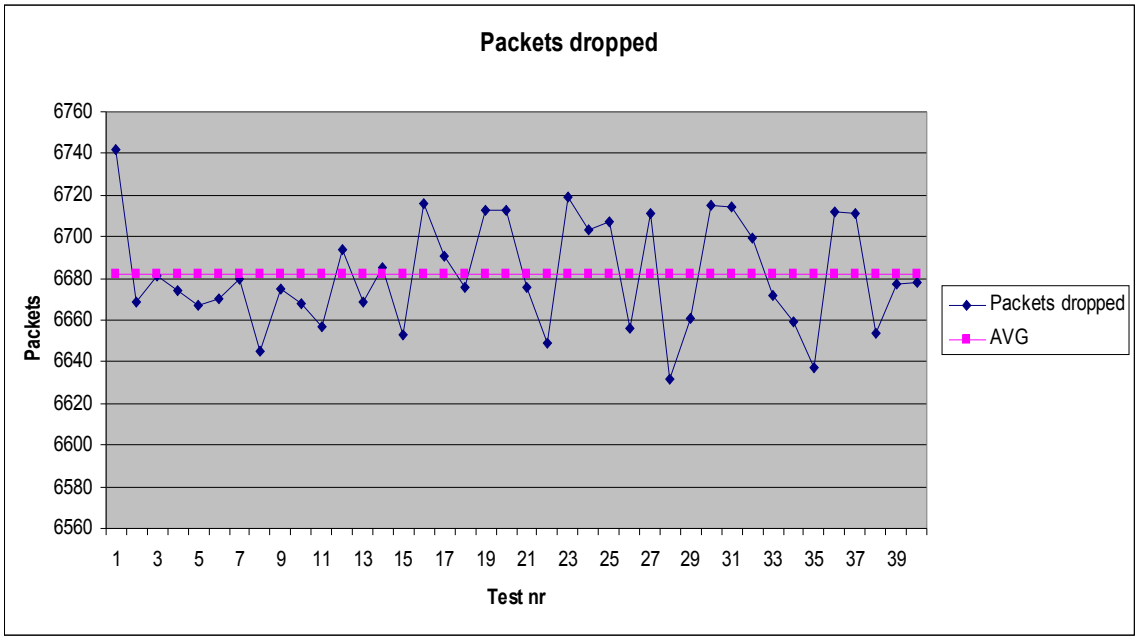


Figure B.3: *Graphical view of Drop rate of packets in % by the IDS compared to the average value.*

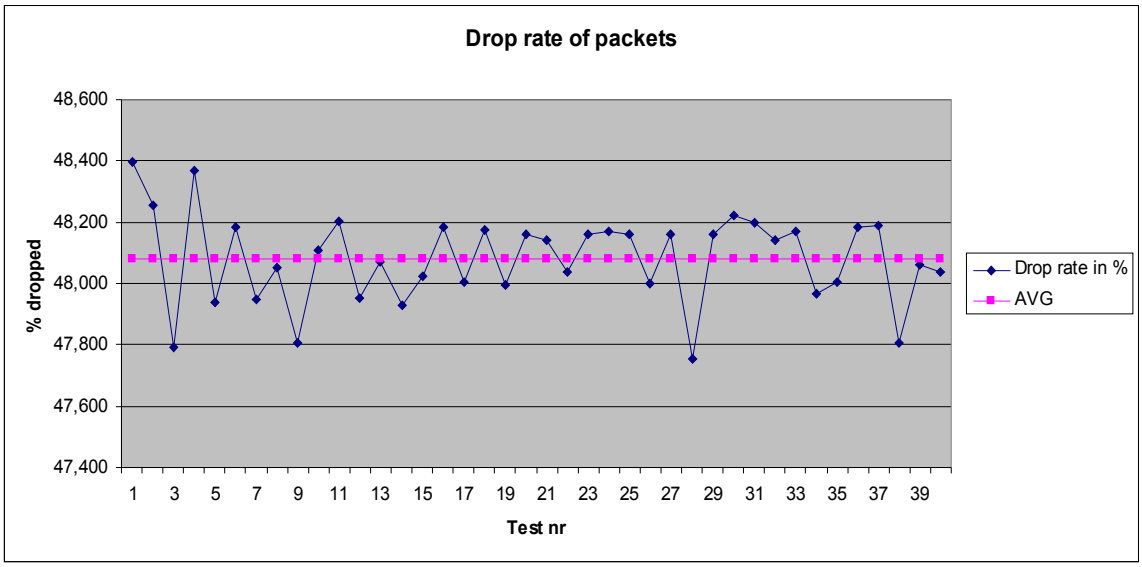


Figure B.4: *Graphical view of Nr of Alerts detected by the IDS compared to the average value.*

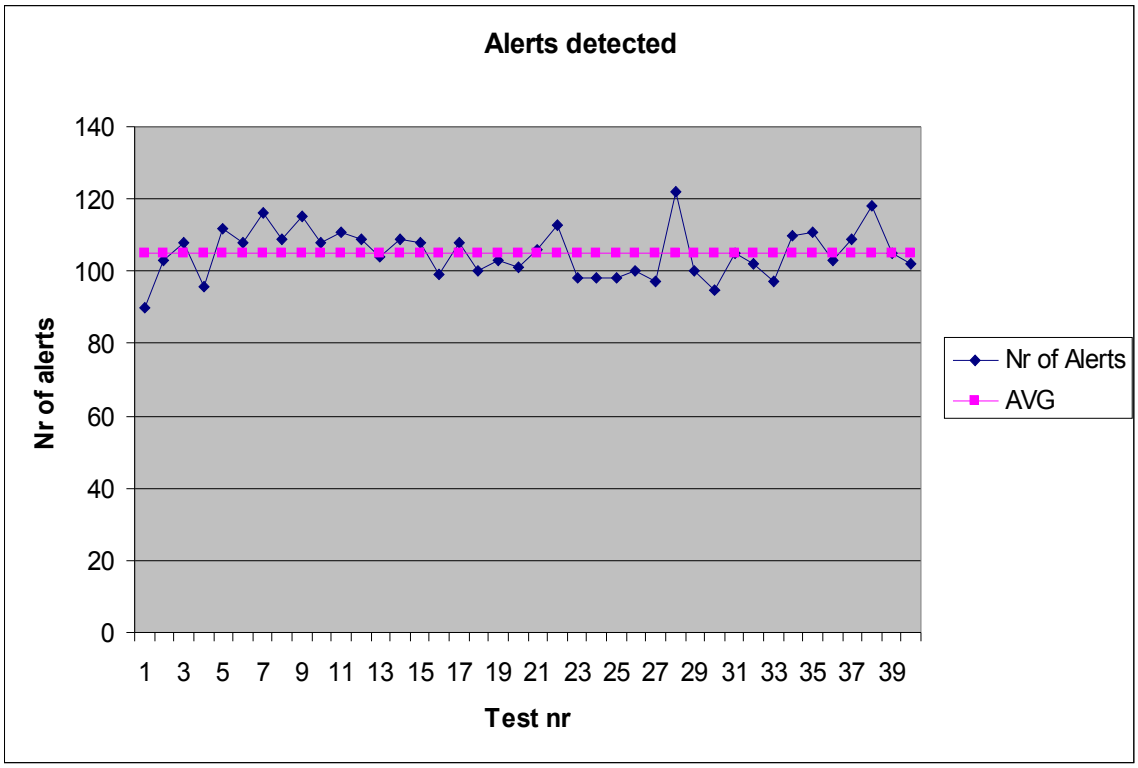


Table B.2: Statistics from the reliability test

Statistical variables	***Packets examined***	***Packets dropped***	***Drop rate in %***	***Nr of Alerts***
Average	3469	6682	48,082	105
Median	3472	6677	48,126	105
Standard deviation	12	26	0,146	7
IV[3]	0,003530475	0,00390016	0,003042636	0,065364929

3 Explained in section B.3

B.3 Discussion

IV is Index of variation and is a measurement of reliability [30], where
IV= Standard deviation / Median. The smaller value it has, the better is the reliability. As we can see from Table B.2, Packets examined, Packets dropped and Drop rate in % are relatively low. The drop rate are naturally derived from the other Packets examined and the Packets dropped values. But as we also can see from the results in Table B.1, Figure B.1-B.4 and the IV of Nr of Alerts, the reliability is not so high as for the other. We can see that even though the packet drop rate is fairly stable, the number of detected alerts varies much more related to the other values.

Since the packet drop rate is very high at around 48 % it is clear this is a big weakness i the IDS test results. But this is also due to limited hardware resources. The variation in alert detection, most likely is a result of the high drop rate. Therefore, this number is not so reliable as wanted in the IDS test results. It is important not to use the numbers directly to draw any final conclusions, but rather see the results all together an look for more general patterns from the results. Using the drop rate together with the results in every test, will therefore become more important to be considered in this setting, than with more powerful hardware.

It is also a question if the IDS reports the right number of packets received, examined and dropped. The reason for this is the possibility that packets may be discarded before snort can detect them. This could be measured with a stress test software where we would have to know exactly the amount of transmitted packets, and compare this to the number of packets detected by the IDS. Such test was not committed during the IDS testing in thesis, due to lack of software where this functionality is present. But it is clearly worth noticing, especially if one is able to produce very "accurate" test results, or testing detection engine or other parts of the operating system dealing with network communication and the TCP/IP stack implementation.

As a last comment on reliability, is the validity of this reliability test. Validity is if we are measuring what we have intended to measure[4]. As explained in the previous introduction section B.1, the tool used in this test is assumed to be somewhat representable for all the test tools used in the IDS tests. Preferably, every tool ought to have a reliability test like in B.2, since variations in both traffic amounts and attack types varies. This has consequences for drop rates and nr of detected alerts. But to achieve this, would be too time consuming, so this approach in this appendix is more an estimation of the reliability with an example of how it can be done.

4 Based on definition found at http://www.synergyaids.com/lacriaids/glossary.asp using google's "define:validity" search string

Appendix C: Technical problems

C.1 Introduction

With every project dealing with computers, network and software there is always a risk of things not going as planned, this is no exception. Various issues appeared during the work with this thesis that caused delays and problems with reaching both time limits and with the consequence of some originally planned activities had to be dropped. The technical problems were time consuming which did not help in an else time consuming project. These difficulties had direct impact of the results achieved in this thesis. Much of the difficulties have been already been discussed, and this appendix is intended as a summary.

C.2 IDS (Snort)

There were problems making Snort, version 2.1.2-2, accept the configuration file snort.conf with the http inspection setting enabled. The error was that Snort was unable to find a file called unicode.map, used when examining Microsoft IIS traffic. Though full path to this file were given, Snort still would not accept the configuration file, unless this setting were commented out. This error was fixed on the first dedicated machine by adding the full path, but on the honeypot machine, it was not possible to fix this.

C.3 Testing tools

A lot of the tools used when testing the IDS, was either problematic to get to work, or they did not work at all. Compiling was one of the greatest challenges due to old source code, and use of a fairly new Linux distribution. These showed to be incompatible, and some of the tools could not be used. This caused less data to work with. Substitutes were tried to be found, but either there were no-existent, or even the substitute would not work.

C.4 Hardware

Lost the dedicated IDS machine during the start of configuration scheme 2, so the victim machine, the honeypot, had to host the IDS as well as the services. The old hardware was the reason for high packet drop rates, an less reliable results. Lack of resources on both computer and network devices affects the results.