Smart Intrusion Detection Systems

Diploma Thesis
in Electrical Engineering

open systems

Rolf Sigg
Thomas Singer

March 2001
Preface

Our diploma thesis gave us the great opportunity to study IT security in general and particularly intrusion detection at Open Systems AG, a company focusing on Internet security. The challenging objective of our thesis was to improve existing intrusion detection solutions in order to reduce the number of false alerts.

We would like to thank Stefan Lampart and Lionel Gresse, our tutors at Open Systems, for their encouraging guidance as well as Florian Gutzwiller, CEO of Open Systems, for offering us this project. Many thanks also go to everybody else at Open Systems for their hospitality and the great working atmosphere.

We are also grateful to Marc Rennhard and Prof. Bernhard Plattner from the computer systems research group of the Swiss Federal Institute of Technology for making this project possible and giving us helpful suggestions.

Our work gave us interesting insight into the areas of IT security and computer networking, and we had the opportunity to get to know Open Systems, which is a great company to work at.

Zurich, March 2001

Rolf Sigg

Thomas Singer
Abstract

Our diploma thesis is about improving intrusion detection systems (IDS). Intrusion detection is the art of detecting inappropriate, incorrect, or anomalous activity on computers and computer networks. Today, the majority of intrusion detection systems try to accomplish this task by acting somehow like a virus scanner. They look at captured network packets or system logs in order to find occurrences of patterns they have stored in a database of known attacks and system vulnerabilities. Having found such a pattern, IDS try to take adequate reactions such as resetting the communication connection or alerting an emergency team.

One of the main problems with intrusion detection systems is that they generate many false alerts. The goal of our diploma thesis is to design and implement concepts that reduce the number of false alerts of IDS but still ensure their reliability.

We have designed and implemented what we call smart IDS extensions. These extensions capture alerts generated by conventional intrusion detection systems and take further investigations to find out whether or not these alerts are false ones. For example our extensions are able to find out whether or not an attacked host is vulnerable against an observed attack.

In a test-environment, we have been able to show that using our smart IDS extensions, the number of false alerts can be lowered substantially while the reliability of the underlying intrusion detection system is still guaranteed.
Kurzfassung


Eines der Hauptprobleme mit Intrusion Detection Systemen ist, dass sie viele Fehlalarms generieren. Das Ziel unserer Diplomarbeit ist, Konzepte zu entwerfen und zu implementieren, welche bei gleichbleibender Zuverlässigkeit eines IDS die Anzahl der Fehlalarms reduzieren.

Wir haben sogenannte smart IDS Erweiterungen entworfen und implementiert. Diese Erweiterungen fangen die Alarme von konventionellen Intrusion Detection Systemen ab und untersuchen, ob es sich dabei um Fehlalarms handelt oder nicht. Zum Beispiel sind diese Erweiterungen in der Lage herauszufinden, ob eine angegriffene Maschine gegen eine beobachtete Attacke wirklich ungeschützt ist.

In einer Test-Umgebung konnten wir zeigen, dass dank unseren smart IDS Erweiterungen die Anzahl von Fehlalarmen ohne Kompromisse bezüglich der Zuverlässigkeit des benutzten Intrusion Detection Systems beträchtlich verringert werden kann.
# Contents

Preface ........................................... i
Abstract ....................................... iii
Kurzfassung ..................................... v

1 Intrusion Detection Basics .................. 1
   1.1 Introduction .................................. 1
   1.2 Classification of Intrusion Detection Systems ..... 2
       1.2.1 Data Evaluation .......................... 2
       1.2.2 Data Collection .......................... 3
   1.3 False Alerts ................................... 3
   1.4 IDS Responses ............................... 4
       1.4.1 Reconfiguring Firewall .................... 4
       1.4.2 Terminating TCP Connection ............... 4
       1.4.3 Launch a Program ......................... 6
       1.4.4 Log Event ............................... 6
       1.4.5 Notification via E-Mail .................. 6
   1.5 Location of the IDS ........................ 7
   1.6 Problems and Limitations .................. 8
       1.6.1 Missed Packets .......................... 8
       1.6.2 Denial of Service ....................... 8
       1.6.3 IDS are Detectable ...................... 9
       1.6.4 Insertion and Evasion ................... 9
   1.7 Common Vulnerabilities and Exposures (CVE) .. 13
   1.8 Summary ..................................... 13

2 Intrusion Detection Products ............... 15
   2.1 Overview .................................... 15
   2.2 Abstract Structure .......................... 15
   2.3 The Performance vs. Accuracy Compromise ..... 16
   2.4 Products .................................... 16
       2.4.1 Network Security Wizards Dragon ....... 16
       2.4.2 Internet Security Systems RealSecure .. 20
       2.4.3 Network Flight Recorder ................. 21
       2.4.4 Snort .................................... 23
   2.5 Summary .................................... 25
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Smart IDS Extensions</td>
<td>27</td>
</tr>
<tr>
<td>3.1</td>
<td>The Need for Extensions</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>Problems Addressed by Our Extensions</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>Interaction of Extensions with IDS</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Proposed Investigation Extensions</td>
<td>29</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Availability Tester</td>
<td>29</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Triggered Scanning</td>
<td>30</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Configuration Database</td>
<td>30</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Severity Levels</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>Related Work</td>
<td>35</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Theoretical Work</td>
<td>35</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Event Analysis</td>
<td>35</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Architectural Improvements</td>
<td>36</td>
</tr>
<tr>
<td>3.5.4</td>
<td>IDS Software Evaluation</td>
<td>36</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Reactions</td>
<td>36</td>
</tr>
<tr>
<td>3.6</td>
<td>Summary</td>
<td>36</td>
</tr>
<tr>
<td>4</td>
<td>Classification of Attacks</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>Investigations</td>
<td>39</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Pre-Attack Investigations</td>
<td>39</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Post-Attack Investigations</td>
<td>40</td>
</tr>
<tr>
<td>4.2</td>
<td>Classification</td>
<td>40</td>
</tr>
<tr>
<td>4.3</td>
<td>Attack-Class Dependent Use of Investigations</td>
<td>41</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Ideal Configuration Database</td>
<td>41</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Real World Configuration Database</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>SIDS Implementation</td>
<td>45</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Abstracting IDS Interface — The Dragon Mail-PARSER</td>
<td>45</td>
</tr>
<tr>
<td>5.3</td>
<td>The SIDS-Core</td>
<td>49</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Separated Policies</td>
<td>52</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Macros in the Policy-Files</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Module-Caller</td>
<td>53</td>
</tr>
<tr>
<td>5.5</td>
<td>Modules</td>
<td>54</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Host Tester</td>
<td>54</td>
</tr>
<tr>
<td>5.5.2</td>
<td>TCP Port Tester</td>
<td>55</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Vulnerability Scanner</td>
<td>55</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Configuration Database</td>
<td>59</td>
</tr>
<tr>
<td>5.6</td>
<td>Summary</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>Accuracy Comparison</td>
<td>61</td>
</tr>
<tr>
<td>6.1</td>
<td>Test Environment</td>
<td>61</td>
</tr>
<tr>
<td>6.1.1</td>
<td>The Attacker</td>
<td>62</td>
</tr>
<tr>
<td>6.1.2</td>
<td>The Victim</td>
<td>62</td>
</tr>
<tr>
<td>6.1.3</td>
<td>The SIDS Configuration</td>
<td>63</td>
</tr>
<tr>
<td>6.2</td>
<td>Results</td>
<td>66</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Linux Victim</td>
<td>66</td>
</tr>
<tr>
<td>6.2.2</td>
<td>NT Victim</td>
<td>68</td>
</tr>
<tr>
<td>6.3</td>
<td>Summary</td>
<td>70</td>
</tr>
</tbody>
</table>
CONTENTS

7 Conclusions 71

A Official Project Description 75

B Timeplan 79

C Important Tools and Manuals 81
   C.1 Tcapdump .......................... 81
   C.2 Tcapreplay .......................... 88
   C.3 Fagrouter .......................... 90
   C.4 Ethereal ............................. 92
   C.5 Nemesis .............................. 92
   C.6 Nessus ............................... 94
   C.7 Nmap ................................. 94

D How To . . . 99
   D.1 Install the Smart IDS Add-On and the Mail-Parser ............... 99
       D.1.1 Nessus .............................. 99
       D.1.2 The Smart IDS Add-On ................... 100
   D.2 Add New Vulnerability Checks ............................ 100
   D.3 Add New Signatures ............................. 101
   D.4 Clean the Signatures File ............................. 101
   D.5 Update the Database Module ............................ 102

E Contents of Enclosed CD 103

   Bibliography 105

   List of Figures 112

   List of Tables 113
Chapter 1

Intrusion Detection Basics

1.1 Introduction

Intrusion detection systems (IDS) are computer systems that try to detect inappropriate, incorrect, anomalous, or suspicious activity on computers or computer networks. Besides firewalls, integrity verifiers, virus scanners, or log-file monitors, they are a key element when it comes to securing computers and networks.

IDS detect intrusions based on collected data. There are two ways to collect data: network-based data collection and host-based data collection. Both are explained below. Furthermore, there are two complementary approaches to detect intrusions based on the previously collected data: knowledge-based intrusion detection and behaviour-based intrusion detection. Today the majority of commercial products are knowledge-based but behaviour-based approaches are a topic of current IDS research. Our work focuses on knowledge-based IDS. Below we will shortly explain the difference between the two. Figure 1.1 illustrates the different approaches of collecting data and detecting intrusions.

![Figure 1.1: A classification of intrusion detection systems. We are focusing on network- and knowledge-based IDS.](image)

In general, intrusion detection systems respond actively or passively to detected attacks as shown in figure 1.2. Active responses block or otherwise affect the progress of the attack. Passive responses simply report or record the incident. Active and passive responses are not exclusive. IDS should always log detection results, regardless of whether or not active responses are enabled. The set of possible countermeasures an IDS can take and the different ways it can trigger
an alert or do other kinds of reporting depends on the specific IDS product. The configuration\(^1\) of the IDS specifies how the intrusion detection system reports an attack and what kind of countermeasure it takes.

![Diagram of intrusion detection system]

**Figure 1.2:** An intrusion detection system comparing collected data with a set of patterns of known attacks. Based on the configuration and the results of the comparison, alerts or appropriate countermeasures are generated.

### 1.2 Classification of Intrusion Detection Systems

This section explains how intrusion detection systems are classified regarding how they collect and evaluate data. Tables 1.1 and 1.2 summarize the key points.

#### 1.2.1 Data Evaluation

**Knowledge-Based IDS**

Knowledge-based IDS work somehow like virus scanners. They look for known patterns of specific attacks and system vulnerabilities in previously collected data. The accuracy of knowledge-based intrusion detection systems is considered good. However, their completeness depends on the regular update of the database containing patterns of known attacks. Such a knowledge-based intrusion detection system is shown in figure 1.2.

**Behaviour-Based IDS**

Behaviour-based intrusion detection techniques assume that an intrusion can be detected by observing a deviation from normal or expected behaviour of the system or the users. The model of normal or valid behaviour is extracted from reference information collected by various means. The intrusion detection system later compares this model with the current activity. When a deviation is observed, an alarm is generated. In other words, anything that does not correspond to a previously learned behaviour is considered intrusive. Compared to a knowledge-based IDS a behaviour-based IDS has advantages regarding its completeness and disadvantages regarding its accuracy. This means that probably most intrusions will be

---

\(^1\) How an IDS will be configured not only is a technical issue but also a matter of management policy.
1.3 False Alerts

detected but chances of false positives\(^2\) are quite high. As mentioned above, our work focuses on knowledge-based intrusion detection systems.

<table>
<thead>
<tr>
<th></th>
<th>Knowledge-Based</th>
<th>Behaviour-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Must Be Trained</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Regular Updates Needed</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td>Completeness</td>
<td>Depends on Updates</td>
<td>Good</td>
</tr>
</tbody>
</table>

Table 1.1: Comparison between knowledge-based and behaviour-based intrusion detection systems.

1.2.2 Data Collection

Network-Based IDS

A network-based intrusion detection system (NIDS) watches all the network traffic on one or more physical connection links. This is usually done by setting a network device in promiscuous mode. Therefore, a network-based intrusion detection system cannot monitor network traffic on network segments other than those it is connected to. Note that network-based IDS cannot look for patterns in encrypted traffic (such as generated by an SSH-session for example). This is one of the main disadvantages of network-based IDS. One of the main advantages is that there is no loss in performance on the protected hosts and you do not need to have privileged access to them.

Host-Based IDS

Host-based intrusion detection systems (HIDS) collect data from sources internal to a computer, usually at the operating system level. These sources can include operating system audits and system logs. In contrast to network-based IDS, host-based IDS are also able to monitor encrypted traffic because they can look at it after it has been decrypted by the receiving host. While network-based IDS monitor all the traffic on a given network segment, the scope of host-based IDS is a given system. This limited scope — which is, in some respect, a disadvantage — can also be seen as an advantage because a host-based IDS is better able to keep track of what happens within a single session. Unlike network-based IDS, host-based IDS have an impact on the performance of the hosts they are protecting and you need privileged access to hosts to run an HIDS.

1.3 False Alerts

As other kinds of alarm systems, intrusion detection systems too can generate false alerts. There are two types of false alerts: false positive alerts and false negative alerts.

\(^2\)The term false positives denotes an event where correct traffic (stemming from data that is not malicious) fools an IDS into thinking that an attack is in progress.
<table>
<thead>
<tr>
<th>Works with Encrypted Traffic</th>
<th>Network-Based</th>
<th>Host-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privileged Access to Hosts Needed</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Performance Loss on Protected Hosts</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scope of Collected Data</td>
<td>Network Segment</td>
<td>Single Host</td>
</tr>
</tbody>
</table>

Table 1.2: Comparison between network-based and host-based intrusion detection systems.

**False Positive Alerts:** The term false positive alert (or false positive) denotes an event when correct traffic (stemming from data that is not malicious) fools an IDS into thinking that an attack is in progress.

**False Negative Alerts:** The term false negative alert (or false negative) is somewhat misleading because it denotes an event when malicious traffic stemming from an attack in progress is not detected as such and therefore no alert is generated, although it should have been.

Of course, IDS should try to keep the number of both false positive and false negative alerts as low as possible. It is easy to have an IDS that does not generate any false positive alerts: just switch it off. On the other hand, to prevent false negatives an IDS would simply declare each observed network packet to be malicious. As can be seen from this, it is easy to minimize one of the two kinds of false alerts, but it is much more difficult to minimize both of them at the same time.

### 1.4 IDS Responses

After having detected an attack, an intrusion detection system reacts with an active or passive response (as described above and shown in figure 1.2). In general, IDS can potentially start any program as a response to an attack, but there is a set of standard reactions that most IDS implement. We will describe those standard reactions below. Note that an essential part of deploying intrusion detection systems is to define a response for each known attack.

#### 1.4.1 Reconfiguring Firewall

IDS may actively respond to detected attacks by reconfiguring firewalls. For example, an IDS may block a given IP source address for a specified amount of time after having detected a port scan from this source. However it should be noted that this kind of reaction is extremely dangerous because it makes the protected site vulnerable to DoS attacks. An attacker could simply send spoofed packets containing patterns that lead to a firewall reconfiguration and therefore to a blocking of traffic from the spoofed address.

#### 1.4.2 Terminating TCP Connection

One of the most commonly used active reactions is terminating the TCP connection. This is done by sending two TCP packets with the reset-flag set. Note that

---

³DoS is short for Denial of Service
it is important that a TCP reset-packet is sent to both the attacker and the victim host (as it is shown in figure 1.3). If a packet was sent to the attacker only, he could simply ignore it and continue his conversation with the victim host. It should be noted that this kind of active response does not work with UDP-based attacks. Another problem are attacks with “immediate impact”. There are cases where all the data that is necessary to complete the attack is contained in the same packet as the attack-specific pattern which will be detected by the IDS. In this case, at the time the reset-packets arrive at their destinations the attack is already successfully completed (assumed that the attacked host is vulnerable).

![Diagram of IDS response](image)

**Figure 1.3:** An IDS has detected an attack and is — as an active response — terminating the TCP connection by sending spoofed TCP packets with reset-flags to both the attacker and the victim host.

**Example**

Figure 1.4 shows an HTTP DotDot attack. The attacker's hostname is `attacker` and the victim's hostname is `victim`. As one can see from the figure, first a TCP three-way handshake is established, then the attacker sends the malicious HTTP request. Finally the HTTP server — which in this case is not vulnerable to the attack — responds by telling the attacker that his request is not valid.

Let us have a look at the same attack again, but this time a RealSecure sensor monitors the conversation. We configured the sensor in such a way that it terminates TCP connections wherein patterns of a DotDot attack have been found. Figure 1.5 shows what is going on in this case. As above, a TCP three-way handshake is established first. Then the attacker sends its malicious request, right after which we can observe two reset-packets. These packets are generated by the RealSecure sensor. We can also see that the sensor spoofes the source IP address of the reset-packets, so that it seems as if one packet is coming from the victim server (victim) and the other from the attacker (attacker). The sensor

---

4For example by using a modified TCP/IP stack or a special firewall configuration.

5This attack attempts to obtain information above the ServerRoot directory. Web servers vulnerable to this attack will allow remote users to read any file on the target webserver readable by the user-ID of the webserver process.

6Information about this commercially available intrusion detection product can be found in chapter 2.
also automatically uses the correct TCP sequence-numbers, which is necessary for the packets to be accepted on both sides.

<table>
<thead>
<tr>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.81079</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>1027 &gt; http [SYN] Seq=67099970 Ack=0 Win=32120 Len=0</td>
</tr>
<tr>
<td>7.81082</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>http &gt; 1027 [SYN, ACK] Seq=67099970 Ack=600950180 Win=32120 Len=0</td>
</tr>
<tr>
<td>7.83246</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>1027 &gt; http [ACK] Seq=67099970 Ack=600950181 Win=32120 Len=0</td>
</tr>
<tr>
<td>24.51072</td>
<td>attacker</td>
<td>victim</td>
<td>HTTP</td>
<td>GET / / [ / / HTTP/1.1</td>
</tr>
<tr>
<td>39.83159</td>
<td>attacker</td>
<td>victim</td>
<td>HTTP</td>
<td>Continuation</td>
</tr>
<tr>
<td>39.85410</td>
<td>victim</td>
<td>attacker</td>
<td>TCP</td>
<td>http &gt; 1027 [ACK] Seq=67099970 Ack=60099704 Win=32120 Len=0</td>
</tr>
<tr>
<td>39.97020</td>
<td>victim</td>
<td>attacker</td>
<td>HTTP</td>
<td>Continuation</td>
</tr>
<tr>
<td>39.98170</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>1027 &gt; http [ACK] Seq=67099970 Ack=60095076 Win=32120 Len=0</td>
</tr>
<tr>
<td>39.98487</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>http &gt; 1027 [FIN, ACK] Seq=600950764 Ack=670999706 Win=32120 Len=0</td>
</tr>
<tr>
<td>39.99170</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>1027 &gt; http [ACK] Seq=670999706 Ack=600950765 Win=32120 Len=0</td>
</tr>
<tr>
<td>39.99246</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>http &gt; 1027 [ACK] Seq=600950765 Ack=670999707 Win=32120 Len=0</td>
</tr>
</tbody>
</table>

Figure 1.4: A DoDot attack from attacker to victim.

<table>
<thead>
<tr>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>57.44854</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>1028 &gt; http [SYN] Seq=1488503332 Ack=0 Win=32120 Len=0</td>
</tr>
<tr>
<td>57.47161</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>http &gt; 1028 [SYN, ACK] Seq=1488503332 Ack=1375567083 Win=32120 Len=0</td>
</tr>
<tr>
<td>57.47250</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>1028 &gt; http [ACK] Seq=1488503332 Ack=1375567084 Ack=600950765 Len=0</td>
</tr>
<tr>
<td>66.11306</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>http &gt; 1028 [HST] Seq=1488503333 Ack=568 Len=0</td>
</tr>
<tr>
<td>66.11540</td>
<td>victim</td>
<td>attacker</td>
<td>TCP</td>
<td>http &gt; 1028 [HST] Seq=1375567084 Ack=600950765 Len=0</td>
</tr>
<tr>
<td>66.12156</td>
<td>attacker</td>
<td>victim</td>
<td>TCP</td>
<td>http &gt; 1028 [HST] Seq=1488503333 Ack=568 Ack=600950765 Len=0</td>
</tr>
</tbody>
</table>

Figure 1.5: A DoDot attack from attacker to victim. A network-based IDS which is protecting victim terminates the TCP connection after having detected the attack attempt.

1.4.3 Launch a Program

Some IDS also have the ability to start a user defined program after having detected an attack. This enables the user to actively or passively respond to attacks in ways not directly implemented in the IDS. It should be noted that the program which is being called by the IDS has limited knowledge about the attack in progress. This limits the flexibility of the user defined program. For example, without knowledge of the sequence-numbers of the attack in progress, it would be difficult to reset a TCP connection.

1.4.4 Log Event

Logging an event is the standard way of passively responding to attacks. It is implemented in all the intrusion detection products we evaluated. The detected attack is logged to a database or to a special monitoring console for further inspection.

1.4.5 Notification via E-Mail

A very common way to passively respond to detected attacks is to send an e-mail to a predefined address. This is actually thought of as a way to notify people of attacks or suspicious activity detected by the intrusion detection system. We will use this notification facility to connect our extension to existing intrusion detection systems. The extension will parse the mails it receives and take further investigations to gain information about the correctness of the alert.
1.5 Location of the IDS

As already mentioned in the introduction, intrusion detection systems are a key element in computer security. So are other systems such as firewalls or virus scanners. It is important to note that all these systems are complementary. It is not true that firewalls or virus scanners can be replaced by an IDS. Likewise it is wrong that firewalls make intrusion detection systems superfluous.

Ideally, in a protected network, there are network-based IDS in each network segment and additionally, every computer runs a host-based IDS. This is not practical and therefore host-based intrusion detection systems are located on a few of the most important hosts only. The location of network-based intrusion detection systems depends on the general topology of the network which is to be protected. Switched networks pose a problem to network-based IDS. There is no easy place to “plug in” a sensor in order to see all the network traffic. One solution to this problem is to use several network-based IDS sensors which report their findings to a centralized IDS manager. This is shown in figure 1.6. As described in chapter 2 most commercial IDS products provide such a distributed architecture. In some cases it might be sufficient to use network-switches which have a special port to which packets from all network segments are being dumped.

![Diagram of network segments with sensors and switches](image)

**Figure 1.6:** With switched networks it might be necessary to use several sensors reporting pattern-matches to a centralized manager.

If a network-based IDS is employed at the network perimeter, a decision must be made whether to put the sensor in front of the firewall or behind it. Both solutions have advantages as well as disadvantages.

**IDS in Front of Firewall:** An intrusion detection system placed between the outermost firewall and the internet service provider is able to monitor all the traffic from and to the Internet. It is therefore potentially able to detect all attacks, even those which do not pass the firewall rules. Strictly speaking, this is only partially true because in a lot of attacks the attack-specific pattern will not be sent unless the TCP three-way handshake has been completed. Nevertheless an IDS in front of the firewall has a larger set of data to match patterns to than an IDS behind the firewall. If the IDS does not have a comprehensive knowledge of the firewall rules, this advantage may...

---

*There are several reasons for this: Host-based IDS are not available for all platforms. It is difficult to guarantee that every newly deployed system runs an IDS. On platforms without the concept of privileged and unprivileged users, everybody can stop a host-based IDS from running.*
also easily turn into a disadvantage because in this case, many false positive alerts will probably be generated. That is, the IDS will probably report a lot of attacks which are blocked by the firewall anyway. Although it is interesting to log such attacks, one would normally not like to have alerts triggered in such cases. A further disadvantage is that an IDS in front of the firewall cannot detect internal attacks.

**IDS Behind Firewall**: Intrusion detection systems behind the firewall are less vulnerable but do not see the network traffic filtered out by the firewall. Therefore — as discussed above — they might detect fewer attacks because some have already been blocked by the firewall but they will probably also generate fewer false positive alerts. Another advantage is that behind a firewall, IDS are able to detect whether or not a firewall is misconfigured or compromised. Observing an attack that is supposed to be stopped by the firewall is certainly worth an alert. Such an integrity check requires comprehensive knowledge of the firewall configuration. While ideally an IDS knows everything about the firewall, in a real world situation this might not be the case. In contrast to an IDS in front of the firewall, an IDS behind the firewall can also detect internal attacks.

1.6 Problems and Limitations

1.6.1 Missed Packets

There are several limitations and problems regarding network-based intrusion detection systems. One of the most serious is keeping up with network speed. It is possible that an IDS is not able to capture and process all traffic from the network. For example, this can happen if the pattern-matching engine is not fast enough to hold with the network speed.\(^8\) This imposes that an IDS should not rely on protocol states when discarding alerts if it wants to prevent an increase in the amount of false negative alerts. For example, imagine an IDS raising an alert for a certain CGI attack if it detects the corresponding pattern on an established TCP connection only.\(^9\) However, if the IDS misses one of the TCP handshake packets, the three-way handshake is actually completed but the IDS drops the alert since there is no TCP connection from its point of view. Therefore, most IDS do not use protocol states.

While keeping up with network speed is a technical problem which can be solved by using faster components (if available), there are also some more fundamental problems which we will discuss below.

1.6.2 Denial of Service

In most cases network-based IDS do not run any interactive services. They simply read traffic from the network and on a few occasions, they send some packets out (as for example to reset a TCP connection). Therefore, an IDS is less vulnerable to attacks than a conventional server. Nevertheless IDS are — as most other systems

---

\(^8\) This might not be a problem on a 2Mbit/s link from a company to its ISP, but it certainly is not that simple on a 100Mbit/s LAN with full load.

\(^9\) This may sound reasonable since a packet that doesn’t belong to a TCP connection will not pass beyond the TCP/IP stack on the target host.
too — vulnerable to DoS-attacks. For example, if an IDS does IP fragmentation reassembly and someone sends a large stream of uncompleted IP fragments, the IDS will eventually run out of memory. Thus it must either drop old fragments or not handle any new ones. In both cases important information might evade the system. There are also other ways to exhaust an IDS’ capacities. Denial of service is especially critical with IDS because in contrast to e.g. firewalls an overloaded IDS will not protect the network any longer.

1.6.3 IDS are Detectable

Another problem with most network-based intrusion detection systems is that they claim to be in so called stealth-mode but they really are rather easily detectable. For example, RealSecure’s IDS is able to reset TCP connections. It does so by sending TCP packets with the reset-flag set (as described in 1.4.2). Unfortunately, for debugging purpose, these packets have a special payload. Thus an attacker receiving such a packet is able to tell that there is a RealSecure IDS protecting the targeted network.

1.6.4 Insertion and Evasion

There is an inherent problem with network-based intrusion detection systems (as pointed out in [19]): the insufficiency of information available in network packets stemming from the discrepancy between the IDS and the targets of attacks. While the IDS reading off network packets has full knowledge about the data that the target host will receive, it does not have full knowledge about how the target host will process this data. The reason for this is that a number of issues exist which make the actual meaning of a captured packet ambiguous, either because not properly specified by the corresponding standards or because some implementations differ from them. This problem could be solved by giving the network IDS more information about the network behaviour of the endsystems, which is a complicated task in real world situations.

A serious problem occurs when the intrusion detection system cannot accurately tell whether some data is processed or dropped by the target host. This makes so called insertion or evasion attacks possible.

Insertion Attack

With an insertion attack the attacker is “inserting data into the IDS”. This is shown in figure 1.7 where the attacker has prepared a packet containing the letter “X” in such a way that it will be processed by the IDS but dropped by the target host. While the attacked host sees the pattern “ATTACK”, the IDS sees the pattern “ATXTACK” and might therefore not detect the attack. As discussed above, this problem occurs because the IDS does not have enough information about the behaviour of the endsystem.

Evasion Attack

The opposite is also possible: With an evasion attack the attacker prepares packets in such a way that the IDS is going to drop them while the endsystem accepts them. This is shown in figure 1.8. Entire sessions can be carried forth in packets that evade the IDS.
Figure 1.7: An Insertion Attack: An attacker has prepared a packet in such a way that the target host does not process it but the IDS does.

Figure 1.8: An Evasion Attack: An attacker has prepared a packet in such a way that the IDS does not process it but the target host does.
Ambiguities and Discrepancies

As discussed above, insertion and evasion attacks are based on the existence of ambiguities within the communication protocols, or other kinds of discrepancies between the IDS and the target host. In such situations important conclusions about a packet can not be made without a secondary source of information. Here we will give an overview of some of the techniques that can be used to launch such insertion or evasion attacks.

**Bad IP Header Fields:** Assume that the authors of an IDS thought that it was unnecessary for an IDS to verify the checksum of captured packets. At first, this seems to be a reasonable assumption but obviously an IDS that does not reject packets with bad checksums is vulnerable to insertion attacks because IP packets with a bad checksum will be discarded by a target host. Strictly speaking this is not really an ambiguity but rather a bug in the IDS. IDS should always discard packets with bad checksums, as hosts do.

**IP Options:** There are some IP options which are also leading to discrepancies. For example, most endsystems will drop a packet that is “strict source routed” when the host’s own address is not contained in the specified source route. However, many operating systems can also be configured to automatically reject all source routed packets. Unless the IDS knows about the destination’s behaviour, it is ambiguous whether or not to drop the packet.

**IP Fragmentation:** IP packets can be broken into smaller packets. This is called IP fragmentation and is an integral part of the IP protocol to support different transport infrastructures. Endsystems reassemble a stream of IP fragments and therefore it is important that IDS do the same. A serious problem of IP fragmentation is that IP fragments are allowed to overlap each other and that target hosts may differ in the way they reassemble those fragments. As shown in figure 1.9, some hosts may favor the old data, others the new. As an example, table 1.3 lists some OSs’ IP fragmentation behaviour.

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Overlap Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows NT 4.0</td>
<td>Always favors old data</td>
</tr>
<tr>
<td>4.4 BSD</td>
<td>Favor new data for forward overlap</td>
</tr>
<tr>
<td>Linux</td>
<td>Favor new data for forward overlap</td>
</tr>
<tr>
<td>Solaris 2.6</td>
<td>Always favors old data</td>
</tr>
<tr>
<td>HP-UX 9.01</td>
<td>Favor new data for forward overlap</td>
</tr>
<tr>
<td>Irix 5.3</td>
<td>Favor new data for forward overlap</td>
</tr>
</tbody>
</table>

Table 1.3: Different OSs vary in their behaviour regarding IP fragmentation reassembly.

**Link-Layer Addresses:** Another possibility for insertion attacks to take place is the link-layer. An attacker may generate packets with the link-layer address of the IDS and the IP address of the target host. This is possible if the attacker is on the same network segment as the IDS and if he knows the link-layer address of the IDS. If the IDS does not check the link-layer address of such packets, it will wrongly assume that the contained data will be processed by the target specified by the IP address. Such a scenario is shown in figure 1.10.
**Figure 1.9:** IP Fragment Overlapping: Two different fragments overlap each other. Target hosts may differ in the way they reassemble those fragments.

**Figure 1.10:** An attacker generates a packet with the network address of the target host and the link-layer address of the IDS. If the IDS only checks the network address of this packet, the attacker is able to successfully issue an insertion attack.
1.7 Common Vulnerabilities and Exposures (CVE)

If different security-related products need to work together, a common naming for vulnerabilities is mandatory. For this purpose, CVE [2] has been introduced.

CVE is a dictionary that assigns numbers to many known security problems. The number is of the form CVE-yyyy-nnnn, where yyyy denotes the year the vulnerability was discovered and nnnn is a number assigned to the different vulnerabilities sequentially.

Before a vulnerability is registered in the CVE dictionary, it first becomes CVE candidate. The CVE candidate number is of the form CAN-yyyy-nnnn, where yyyy and nnnn have the same meaning as explained above.

Nowadays, many security related software developers incorporate CVE numbers into their products. It is possible to get information about a vulnerability from various sources under its CVE number.

1.8 Summary

Network intrusion detection systems are unreliable enough to be considered only as secondary systems designed to backup the primary security systems. Primary systems such as firewalls, encryption, and authentication are solid. Bugs or misconfiguration often lead to problems in these systems, but the underlying concepts are “provably accurate”. Nevertheless, suitably configured intrusion detection systems can substantially improve security.

As we will see, a general naming scheme for vulnerabilities is needed for our work. Therefore, the CVE dictionary is very useful for our smart IDS extensions.
Chapter 2

Intrusion Detection Products

2.1 Overview

In this chapter we will present four intrusion detection products:

- Internet Security Systems' RealSecure
- Network Security Wizards' Dragon
- Network Flight Recorder
- Snort

We are going to give a short description of each product, explain its pattern management — the update of patterns in particular — and describe the its abilities. But first, we present our abstract view of an IDS and some thoughts concerning IDS design.

As mentioned in the introduction, we will focus on network-based intrusion detection only, even if the products also offer host-based intrusion detection.

2.2 Abstract Structure

In order to characterize intrusion detection products, we divide the intrusion detection functionality into three main entities, as figure 2.1 illustrates. First, the sensor entity collects network data. Then it issues local responses according to a policy, such as shutting down a communication channel or sending an event to a central server which stores those events and may trigger an alarm. Finally, the so called console entity provides the user interfaces for configuration and reporting.

The intrusion detection products we will present implement some variants of this abstract scheme. The variations are due to the compromise between functionality, ease-of-use, and performance. Every product also introduces its own terminology. The sensor entity is called agent, sensor, sniffer, capture device, or packet sucker. The central server entity is usually referred to by server or, if integrated in the console entity, management console.

In general, the central server is capable of handling more than one sensor, which makes combination and correlation of events from different sources theoretically possible. For example, if the network-based sensor reports an attack, it would be very useful to consult the events of the host-based sensor of the attacked machine
to gather further evidence about whether or not the attack has been successful. However, the products we present here do not make use of this possibility to raise alerts based on combined and correlated events from different sensors.

2.3 The Performance vs. Accuracy Compromise

We again want to point out that ideally an IDS would know exactly how a target host will react upon each packet on the net and whether or not the observed packet will harm it. Since this is not possible (see section 1.6.4), the IDS should approximate the behaviour of the protected hosts as good as possible in order to produce the most accurate results. However, the better the approximation, the more computing resources are needed. This may not be a problem on a 2Mbit/s link, but it certainly is on a 100Mbit/s or even Gigabit network. Therefore, the products implement a compromise between performance and accuracy.

2.4 Products

This section describes four IDS products. Their key features are summarized in table 2.1 at the end of this chapter.

2.4.1 Network Security Wizards Dragon

*Dragon* [24] by *Network Security Wizards* (NSW) is a commercial IDS providing both host- and network-based intrusion detection. Open Systems AG uses Dragon
because it is a commercially supported product and also very well suited for embedding in a managed security solution. This is because some components of Dragon are open source.

Platform and Architecture

The architecture of Dragon is illustrated in figure 2.2. All components are available for most Unix platforms and Linux. The user interface is implemented as web pages, therefore it is platform independent.

The sensors (called Dragon IDS for the network-based and Dragon Squire HIDS for the host-based sensor) store events in their local databases and push them via Rider Client to the central manager, called Dragon Rider Server.

![Figure 2.2: Dragon features a well-structured architecture. The Alarmtool component is able to generate alarms triggered by events originating from different sources.](image)

The server component Alarmtool acts as a central alarm filter. It watches incoming events and triggers alarms according to the alarm policy. Since it operates on the central database it is able to trigger alerts by events originating from all sensors. However, it is not able to correlate different events in order to decide on an alert. Out of the box, Alarmtool features SNMP, e-mail, syslog and pager alerts. It is easy to add custom actions since Alarmtool is written in Perl.

The sensors can be configured via the server using its web interface. Reporting is done via a separate web interface. As an example, figure 2.3 shows a simple one-day summary report.1

1The web interfaces are only fully compatible with Internet Explorer browsers.
Pattern Management

Network Security Wizards maintains the pattern database and provides regular updates. The rules are stored in plain text format and can easily be customized. Figure 2.4 shows an example rule.

A rule mainly specifies a pattern and where to look for it in the raw network data. The protocol argument specifies the IP load protocol number to look for, where the special letters T, U and I specify TCP, UDP and ICMP respectively. With the second argument (direction) it can be specified whether source and/or destination port must match the port number specified in the seventh argument (UDP/TCP port). The third argument (protected networks) allows to restrict the rule to packets that match certain IP source and destination addresses. For example, this is used to specify rules that only match packets that originate from outside the protected network and have their destination inside the protected network. The fourth argument (search type) specifies whether the search string in
the ninth argument (search string) is compared case sensitively (binary) or case
insensitively (string). For example, case insensitive comparison is useful to cover
all combination of mixed case HTTP requests.\footnote{2}

Dragon has the ability to record additional packets after a detected attack.
This is useful when it comes to reconstructing the intruder's activity. The fifth
argument (dynamic log) specifies how many packets with the same source and
destination IP addresses should be recorded. With the sixth argument (bytes to
compare) the search for patterns can be restricted to the first n bytes of a packet.
If zero is specified, the captured packet is completely searched through. The
signature name in the eighth argument is the name this rule is referred to in all
Dragon messages.

The rules delivered by Network Security Wizards are written to look for pat-
tterns in packets for standard ports only. For example, the rules for CGI attacks
only look at packets with destination port 80. If a user in the network starts a
service on his own using a different port number, attacks to this service are not
being detected.

The network-based sensor features both IP fragmentation reassembly and TCP
stream decoding. This is needed to prevent evasion attacks as described in section
1.6.4. Dragon does not inspect TCP connection states. As explained in 1.6.1,
this is because it is not guaranteed that Dragon can process all packets on the
net. This has the disadvantage that a sensor event can be triggered with a single
packet that contains a pattern. For example, we generated a single TCP packet
containing the string get /cgi-bin/phf directed to some host on port 80. Since
the TCP three-way handshake did not happen, the target host certainly would
have dropped the packet. Nevertheless, the Dragon rule for the PHF attack matched
and the corresponding alert has been raised.

As shown above, events have proprietary names. However, there is a description
file that includes CVE numbers. Therefore it is possible to extract a table that
converts proprietary Dragon names to CVE numbers.

Responses

Besides TCP connection resets, the network-based sensor features ICMP unreach-
able messages (trying to shut down other IP traffic than TCP, namely UDP). It
also implements a mechanism to prevent attackers from guessing the operating
system of a protected host using TCP fingerprinting\footnote{3}. If TCP fingerprinting ac-
tivity is being detected, the sensor emanates TCP packets with randomly set flags,
therefore irritating the attacker. However, active responses on the network can be
dangerous (see section 1.6.3) and should be well thought out.

\footnote{2} Whereas the behaviour of a web-server concerning mixed case requests is clear, HTTP requests
encoded in Unicode impose a huge problem to network-based IDS since there are many different
ways to encode a string in Unicode and the web-servers accept not all of them. Therefore, besides
the actual Unicode parsing, the IDS must know which encodings are accepted by the attacked
web-server and which are refused.

\footnote{3} Responses to ambiguous TCP packets vary between TCP/IP stack implementations. TCP
fingerprinting uses this fact to determine the operating system of a host since most operating
systems use their very own TCP/IP stack. [3] describes basic TCP fingerprinting.
2.4.2 Internet Security Systems RealSecure

*RealSecure* [22] by *Internet Security Systems* (ISS) is a commercial IDS too. It provides both host- and network-based sensors and a central management console.

**Platform and Architecture**

The sensors are available for most operating systems (NT, Unix), whereas the management console is available as a Windows application only. The architecture of RealSecure is illustrated in figure 2.5.

![RealSecure Architecture Diagram](image)

**Figure 2.5:** RealSecure is designed to take all responses in the sensor software. The management console serves as a reporting and configuration tool only.

Once configured using the console, the sensors operate independently. The console acts as a central configuration and reporting tool only. All responses are performed by the sensors. It is not possible to trigger an alarm by correlated events originating from different sensors. In return, the system is able to work without a central server or a console running.

**Pattern Management**

The patterns the sensors use are provided by ISS in a proprietary format. The IDS operator can define new patterns only in the sense that there are general patterns whose parameters can be adjusted. For example, there exists a pattern that looks for operator-defined strings in e-mail subject headers.

As with Dragon Sensor, the RealSecure network-based sensor features both IP fragmentation reassembly and TCP stream decoding but no TCP connection state inspection. While Dragon rules are written for fixed ports but can be changed to
work with any port, the RealSecure rules are written for fixed ports and can not be changed.

If available, the CVE number of a vulnerability is included in the description of the corresponding alert.

Responses

Only the sensors themselves can issue responses to events, therefore acting independently of each other. Additionally to the simple logging response the network-based sensor supports TCP connection reset, e-mail, SNMP trap and firewall reconfiguration, as can be seen from figure 2.6.

Figure 2.6: RealSecure provides various responses, from mere display on the console (assumed it is running) to reconfiguring firewalls.

2.4.3 Network Flight Recorder

*Network Flight Recorder* (NFR) [23] is both a company and a product brand. NFR offers network-based intrusion detection only. The name is taken from the notion of a flight recorder, which allows detailed analysis of an airplane crash based on the recorded data.

The first versions of NFR were developed under GNU General Public License (GPL) [41], but today NFR is a commercial product.

Platform and Architecture

NFR has a very open architecture, illustrated in figure 2.7. The sensors provide events which either directly generate alerts or can be passed to a so called *back-
end. Out of the box there are different backends (e.g., for table-based storage or histogram statistics) and the operator can create his own backends.

![Network Flight Recorder diagram]

**Figure 2.7:** Network Flight Recorder does not support local responses in the sensor, but allows some kind of event correlation in the backends.

The backends themselves have the possibility of triggering alarms based on their data. Thus NFR provides a limited ability to raise alerts based on correlation of events from different sources.

**Pattern Management**

The pattern-matching engine of the network sensor is programmable using the so-called *N-Code* language [20]. Figure 2.8 shows an example N-Code script that detects a widely known CGI abuse attack. N-Code allows the implementation of highly customized events. NFR maintains a database of scripts, and there exist many other sources for specific problems. The network sensor supports IP fragmentation reassembly and TCP decoding. Using global variables it is possible to take more than one packet into account to decide on a response, thus making some kind of correlation possible. Unfortunately, NFR doesn’t support CVE numbers.

**Responses**

From the sensor’s view, responses are either *logs* or *alerts*. Logs are sent to a backend for further processing and storage. A backend can trigger an alert based on the logs.

---

4 For example, L0pht Heavy Industries [4] provides N-Code scripts for various known vulnerabilities.
badString = "GET /cgi-bin/phf" ;

phf_attack_schema = library_schemas.new ( 1, [ "time", "ip", "int", "ip", "int", "str" ] . scope ( ) ) ;
phf_attack_recorder = recorder ( "bin/list packages/id/phf_attack.cfg", "phf_attack_schema" ) ;

filter phf_attack tcp (client, dport: 80) {
    declare $thisBlob inside tcp.connSym ;
    if ( $thisBlob ) {
        $thisBlob = cat ( $thisBlob. tcp.blob ) ;
    } else {
        $thisBlob = tcp.blob ;
    }
    if ( index ( $thisBlob. badString ) < 0 ) {
        return ;
    }
    record packet.sec. tcp.connSrc. tcp.connSport. tcp.connDst. tcp.connDport, $thisBlob to phf_attack_recorder ;
    $message = cat ( tcp.connSrc, " is attempting a phf attack on ".
        tcp.connDst. ", port ", tcp.connDport, ": ", $thisBlob ) ;
    alert ( alert:PHF_ATTACK. alert:SUSPICIOUS_ACTIVITY. $message ) ;
}

**Figure 2.8:** This example N-Code detects the CGI PHF attack, logs (records) important data and triggers an alert.

TCP reset and other local responses are not supported by the network sensor, following the notion of a flight recorder. Since the sensor is implemented using a dedicated host (with a proprietary, OpenBSD based operating system), it would not be easily possible to add local responses manually.

### 2.4.4 Snort

*Snort* [25] is developed by the open source community and is available under GPL. The developers describe their product themselves as “lightweight intrusion detection system for networks”. They also compare Snort to NFR as “little brother to NFR’s college-bound football hero”. NFR is far more flexible and has important features that Snort is lacking when it comes to smart intrusion detection. However, Snort is still open source freeware whereas NFR is not anymore.

**Platform and Architecture**

Snort runs on top of the libpcap library [27], therefore supporting various platforms.

The architecture of Snort is illustrated in figure 2.9. In Snort terminology, an IDS is divided into three functional entities: *packet decoding, detection*, and *alerting*. Although Snort assigns each functionality a separate process, the alerting process is not able to raise alerts based on events from different sources. All user interaction is done via command line. There is no reporting tool or other frontend.
Pattern Management

The patterns are defined by simple rules. The internet community provides many sources for rule sets suited to many situations. It is easy to adapt the system to one’s needs. Figure 2.10 shows an example rule. Snort includes CVE numbers in its default rules.

```
alert tcp !10.0.0.0/8 any -> 10.0.0.0/8 80
(msg:"IDS128 - CVE-1999-0067 - CGI phf attempt";
 flags:PA; content:"/phf";flags:AP; nocase;)
```

Figure 2.10: This example rule detects a CGI PHF attack from outside the network (10.0.0.0/8) to any host inside.

Snort does neither IP fragmentation reassembly nor TCP stream decoding. However, these features are planned to be implemented in future versions.

Responses

Snort supports two categories of action: logging and alerting. Logging writes the packet that triggered the action to a file whereas alerting sends the packet to syslog, an alarm file or — as a WinPopup message — to a Windows host.
2.5 Summary

In table 2.1 we summarize the features of the four presented IDS products. All products have in common that they issue alerts based on a pattern-match on the network traffic. Since they do not check the success of an observed attack, alerts are risen even if the attacked host actually is not vulnerable and therefore the attack was not successful. We will address this problem by introducing extensions to IDS that verify the success of an observed attack-attempt.

<table>
<thead>
<tr>
<th></th>
<th>Dragon</th>
<th>Real-Secure</th>
<th>NFR</th>
<th>Snort</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensors</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network-Based</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Host-Based</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IP Fragmentation Reassembly</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP Stream Decoding</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Local Responses</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual Pattern Database Update</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>GUI Console</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Reporting</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>CVE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CVE/CAN Number Support</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td><strong>License</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Freeware</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

*Table 2.1:* Overview of the presented products. The checkmark in parenthesis indicates that only a few components of Dragon are open source.
Chapter 3

Smart IDS Extensions

3.1 The Need for Extensions

Planning, installing, and maintaining company-wide IT security is a very expensive task that needs a lot of know-how and experience. Since there is much human interaction, the running costs are high. Therefore, many companies outsource IT security to specialized firms. If such a firm offers installation and surveillance of IT security equipment\(^1\), the service is called monitored security. If the firm also takes actions when incidents are detected, thus actively protecting the network, the offered service is called managed security.

Open Systems AG offers managed security services. Many client installations are observed and controlled centrally. In order to keep maintaining effort low, Open Systems wants to automate as many tasks as possible.

One of the managed security services is managed intrusion detection: When an IDS reports an alert, a security engineer from Open Systems has to manually investigate the alert and take appropriate actions. Often, he finds that indeed the attack took place but was not successful. For example, an IDS detected an IIS-specific exploit but the attacked site is running an Apache webserver and is therefore not vulnerable against the observed exploit.

Companies providing managed intrusion detection are very much interested in reducing the amount of such false positive alerts, because false alerts mean additional operating expense. Furthermore they have special demands on intrusion detection systems. Our goal is to design and implement extensions to existing IDS that help to decrease the number of false alerts while holding the requirements listed below.

**Scalability:** It should be possible to protect large networks. This makes continuous auditing of hosts impossible, because this method does not scale with the number of hosts.

**No Privileged Access:** It should also be possible to protect hosts to which no privileged access is given. Therefore we cannot use host-based IDS.

**No Knowledge of Configuration:** Although it is easier to protect hosts whose configuration is known, it must be possible to protect hosts with unknown or frequently changing configuration.

---

\(^1\)For example firewalls, intrusion detection systems, backup systems, virtual private network gateways, and virus scanners.
No Increase in False Negatives: The primary goal of the new IDS concepts which have to be designed is to reduce the number of false positive alerts. Nevertheless they should not increase the number of false negatives under any circumstances. This makes conservative decisions necessary: Alerts may only be discarded, if it's completely sure that they are not correct.

3.2 Problems Addressed by Our Extensions

As explained earlier, knowledge-based intrusion detection systems have a set of patterns which correspond to known vulnerabilities. Network-based IDS try to find occurrences of those patterns in previously collected network traffic data. A company or person deploying such systems has to decide for each known vulnerability what action to take when the corresponding attack occurs. In particular it has to be defined for each recognized attack-pattern whether or not to generate an alert upon occurrence.

We think that this is a severe limitation in functionality and one of the main disadvantages of current commercial IDS products, because whether or not a specific attack, which has been observed on the wire by a network-based IDS, has been successful depends on the circumstances. For example an Apache webserver is not going to be vulnerable to an IIS specific exploit. But the IDS observing such a pattern on the wire does not have enough information to take such contexts into consideration. It even could be the case that the pattern observed by the IDS does not come from a real exploit but rather is a forged packet from someone who'd like the IDS to think that an attack is in progress.

These are reasons we decided to design extensions that help to take further investigation to find out whether or not an observed attack has been successful.

3.3 Interaction of Extensions with IDS

Our solution for extensions to interact with existing intrusion detection systems is to capture the alerts generated by the existing IDS and take further investigations to decide whether or not the observed event is really worth an alert. Figure 3.1 shows the basic idea: We will implement the extensions presented in this chapter in a so called smart IDS add-on which is independent of the underlying IDS-technology. To connect existing IDS products to our add-on we will write an abstracting IDS interface which converts alerts coming from a specific IDS from their proprietary format to the format understood by the IDS-independent SIDS add-on.

We shortly summarize the different terms we will use throughout the report to refer to our ideas and implementations:

**Smart IDS Extensions**: The concepts and ideas we designed to help decrease the number of false positive alerts by taking further investigations trying to find out whether or not an attack has been successful. We describe these concepts in the current chapter.

**SIDS Add-On**: The program that implements the smart extensions. Implementation issues are covered in chapter 5.
Abstracting IDS Interface: The program that connects our add-on to existing IDS products.

Smart IDS (SIDS): A conventional IDS product combined with our smart IDS add-on becomes a smart intrusion detection system (SIDS).

Figure 3.1: An abstracting IDS interface is capturing alerts coming from an existing IDS and passes them to the SIDS add-on. The add-on in turn decides based on the investigation of the extensions it implements whether to discard or pass on the alert.

3.4 Proposed Investigation Extensions

In this section we will propose extension that will take further investigations that help to decided whether or not an alert may be dropped.

Availability Tester Checks whether or not a host or service is available.

Triggered Scanning Checks whether or not a host or service is vulnerable to a specific attack.

Configuration Database Contains information about the configuration of host.

Severity Levels Helps to detect suspicious activity.

3.4.1 Availability Tester

After having detected an attack, it would sometimes be handy to have a program which is able to test whether or not a specific service is running on a given host. For example, this would make sense after detection of a DoS attack. But comprehensive knowledge of the services running on the different machines is required in this case. Assume you detect an attack against port 80, of a machine. If you do not know whether this machine really is running a service on port 80 it does not make sense to test the availability of port 80 after the attack.

Another example where one could use an availability tester to decrease the number of false positives is with attacks that exploit a specific service but do not have any DoS-effects on it. After detection of such an attack one could simply test for the availability of the corresponding service on the victim host. If there is no such service running, the attack could impossibly have been successful.

Note that the fact that an attack takes place in application-layer data does not necessarily mean that there really is a corresponding service running on the
attacked host. One could think that application-layer data can only be sent after completion of a TCP three-way handshake with the corresponding service. Unfortunately this is not true because an enemy could also forge a single packet containing an attack-specific pattern without first going through a handshake with a listening service. Such a forged packet would of course be discarded by the receiving entity. But most IDS (like RealSecure or Dragon) do not take this fact into consideration because they do not keep track of established connections. They rather look at single packets for pattern-matches. Accordingly, in the case above of a single forged packet, a false positive alert would be generated and thus further investigations using an availability tester as described above could be helpful. If no such investigation takes place someone might fool the operators of an IDS at will. This is especially a problem for companies providing managed security.

How to generate single packets that will fool most IDS in such a manner is explained in appendix C.5.

3.4.2 Triggered Scanning

Another idea for a SIDS extension is to use a vulnerability scanner to see whether or not a detected attack has been successful. That is, after having detected a specific attack against a protected host, we would use a scanner to test the attacked host's vulnerability to this particular attack.

It could be argued that the vulnerability of systems should be checked in advance, that is before a specific attack indeed occurs. Such a kind of auditing has several disadvantages. As the systems which are to be protected often aren't under control of the people running the IDS, their configuration may often change within small amounts of time. This makes persistent auditing necessary which is not practicable because it does not scale with the amount of hosts which have to be checked. Furthermore such a persistent auditing would generate a lot of misleading entries in the hosts' log-files and it also might increase the network load dramatically.

Several vulnerability scanners exist on the market today. One of them is called Nessus. Figures 3.2 and 3.3 show two screenshots of a GUI-client of this security scanner. To use Nessus for triggered scanning (in the manner as described above) the scanning-process has to be automated. This means it must be possible to call the scanner from within a script and tell it exactly what vulnerability on which host it has to check for. Further the results of the scanning-process should be easily accessible from within a programming environment, too. Both above requirements are not met by the GUI-client shown in the figures. We have solved this problem by directly calling the core of Nessus thus evading its client/server-architecture (see section 5.5.3 for further implementation details).

Using triggered scanning, one has to assure that the IDS does not generate alerts for traffic caused by the scanner. The solution to this problem would be that in a certain time interval, alerts originating from the IDS (determined by the source IP address) with identical destination IP address and port number concerning the same event are cancelled.

3.4.3 Configuration Database

A serious problem with taking further investigations after occurrence of an attack is that the behaviour of the victim host after the attack may differ from the
3.4 Proposed Investigation Extensions

Figure 3.2: The Nessus Security Scanner: A plugin selection screen shows all installed plugins. Only one plugin has been activated here. This plugin looks for a serious security hole found in some versions of the wu-ftp daemon.

Figure 3.3: The Nessus Security Scanner: This screen shows the results of a scan which has been run. As can be seen, Nessus has found a serious security hole within the wu-ftp daemon running on the target machine.

behaviour before. Therefore it would often be very helpful to have accurate information about the network and the configuration of the hosts which have to be protected. This can be achieved by having a configuration database containing information such as which host does run what operating system or which machine provides what services. Unfortunately, as already mentioned earlier, this is not a practicable approach because host configurations may change very often and without the people running the IDS being notified about it. For example a machine previously running only an ftp-service is being chosen to also act as a webserver, or an entirely new host is being added to the network which has to be protected by the IDS.

Therefore, a better approach would be to use a dynamic\(^2\) database. This can

\(^2\)Dynamic in the sense that it would continuously update itself automatically.
be achieved by continuously seeking\(^3\) for new hosts and by continuously fetching banners of the services running on the active hosts. In contrast to continuous auditing this does not generate traffic that looks malicious to a single host, except maybe that somebody might think that a port scan is in progress. But this problem can be solved by masking portscan attempts coming from the IDS’s IP address. Also this approach is not likely to increase the network load dramatically.

Of course such a dynamic database has got the disadvantage that information taken out of it could potentially be expired. For example if there was no database update between a change in a system’s configuration and a query regarding something that has just previously been changed. Such a situation is shown in figure 3.4.

\[\text{Figure 3.4: In this example the information taken from the dynamic database has expired because a change in system configuration and a database query happen within a time-slot in between two database updates.}\]

The consequences of the above discussion are the following:

**Dynamic Database:** Because of the potentially expired information contained in such a database, there is the possibility of an increase in the amount of false negative alerts. This is against the preconditions for our smart extensions and thus dynamic databases cannot be used.

**Static Database:** Because this requires every change in network or host configuration be notified to the people responsible for the IDS, this is not a practicable approach.

Thus it seems that configuration databases are of no use. But there is a solution to the above problems: to distinguish between very important hosts and less important hosts in a protected network:

**Very Important Hosts:** Normally in a protected network there are a few hosts which are far more important than most others (for example the company’s webserver or other key-components). The configuration of such hosts is usually more stable than those of less important hosts. Also there are only a few persons which have the right to change this configuration. Therefore in this situation it might be practicable to use the static database approach described above.

**Less Important Hosts:** With less important hosts one might accept the small amount of increase in the number of false negatives to have the huge decrease in the number of false positives a dynamic database approach offers.

\(^3\) Continuously means automatically in given time periods or triggered by operators whenever they alter system configurations.
Especially because the increase in false negatives can be made very small by updating the database very frequently.

Table 3.1 shortly summarizes these points.

<table>
<thead>
<tr>
<th></th>
<th>Static Configuration Database</th>
<th>Dynamic Configuration Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updates</td>
<td>manually, whenever configuration changes happen</td>
<td>automatically, in given time intervals</td>
</tr>
<tr>
<td>Advantages</td>
<td>always up to date</td>
<td>configuration changes need not be reported</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>requires changes be reported</td>
<td>potentially contains expired information</td>
</tr>
<tr>
<td>Applicable</td>
<td>for a few very important key-element hosts</td>
<td>for the rest of the protected network containing less important hosts</td>
</tr>
</tbody>
</table>

**Table 3.1:** A comparison between static and dynamic configuration databases.

### 3.4.4 Severity Levels

Normally when configuring an intrusion detection system one has to determine for each recognized attack-pattern whether or not it will generate an alert upon match, as shown in figure 3.5.a. But this is not an optimal solution for all patterns.

With smart extensions like the availability tester, the triggered scanner, or the dynamic configuration database (all discussed above) the situation can be improved by taking additional information into consideration, as shown in figure 3.5.b. But the same pattern-match under the same momentary circumstances will still either generate an alert or it will not. It’s only that the decision depends on more things than just the IDS’ configuration. Correlations between single pattern-matches are not taken into consideration. Each pattern-match is further investigated separately and independently of others.

It would be nice if one could correlate events, such that some events will not generate alerts on their own but together with others they may become important enough to do so (see figure 3.5.c). We would then have three classes of patterns:

1. Patterns that will always generate an alert or be discarded based on the result of module investigations.
2. Patterns that will always immediately generate an alert.
3. Patterns that will generate alerts depending on what patterns occurred previously.

Figure 3.5 shows the three different cases we have discussed just now. Situation a) shows the standard IDS solution: Whether or not a pattern-match will generate an alert is being decided solely on what kind of pattern it was. The second figure shows the situation where further information stemming from smart extensions is taken into consideration. The decision whether or not an alert is being generated is
Figure 3.5: With the concept of correlation even more information is used to decide whether or not an alert has to be generated upon pattern-match.

now more sound standing because it depends on the current circumstances. In the last figure the decision even depends on the past occurrence of pattern-matches.

Using such a concept of correlation a situation as shown in figure 3.6 could occur. Three events which are not critical enough to generate alerts on their own appear closely in time. But because they appear in a sequence, altogether they will finally trigger an alert.

Figure 3.6: A, B and C are patterns that would not generate alerts on their own but can do so if they occur in a sequence closely in time.

We now propose a possible way to realize such a correlation which we will call severity levels:

A so called activation-level is being assigned to each recognized attack pattern. Further all patterns also have a severity-level assigned which is initially set to zero. An alert is being generated only if the current severity-level of the observed attack pattern exceeds its activation-level. The occurrence of attack patterns can increase other pattern's severity-levels as shown in figure 3.7. For example when pattern A is being observed, the current severity-level of pattern A is being increased by 1, the level of B by 5 and the level of C by 2. Pattern C has an activation-level of 5. Therefore on its own it would not generate an alert but the pattern sequence C,A,C would set the current severity-level of C to a value of 6 and therefore generate an alert. The severity-levels of patterns are being decreased by a fixed value per time
unless they are zero. This makes sure that the correlation of attacks takes place dependent on how close in time they appear.

![Diagram](image)

**Figure 3.7:** With the concept of severity-levels attacks can be correlated.

### 3.5 Related Work

Intrusion detection is relatively new area and under continuous development. In the following sections we are going to present work which is related to our diploma thesis.

#### 3.5.1 Theoretical Work

Due to the complexity of computer network systems there is only few theoretical work about intrusion detection. Axelsson [8] applied the so called *Base-Rate Fallacy* to intrusion detection concluding that the limiting factor in intrusion detection useability is not the rate of detected attacks but the rate of false alerts.

This situation is what we came across to at Open Systems AG and is a main motivation to address the decrease of false alerts with smart intrusion detection extensions.

#### 3.5.2 Event Analysis

Event analysis means processing the collected events and decide upon alert or not. There are events that do not indicate harm on their own but in conjunction with other events, they indicate pre-attack activity. Traditionally, the network security officer keeps watching the event logs of the intrusion detection system and detects such anomalous event combinations.

With larger networks and more events, this is not practicable. Therefore recent results in *data mining* and *machine learning* are used to search huge amounts of event data for anomalous activity.

for the same task. Data mining methodology is used in Manganaris et al. [13] as well as in Lee et al. [10], which also use adaptive data mining patterns. All those approaches try to help the network security officer in detecting anomalous activity in a large set of events.

In order to generate more accurate alerts, Vigna et al. [14] and Qian et al. [15] take protocol states into account. For those approaches it is important to be able to process each packet on the observed network segment. See section 1.6.1 for a description of the problem of missed packets.

As an example of a new proposition for host-based intrusion detection, we describe Helmer et al. [7] in short. They suggest to use system call traces of known running software to determine whether this software is used in a normal manner or is maybe being abused. An artificial intelligent system, which needs to be trained before, analyzes the system call traces. In the learning phase, while disconnected from the productive net, operating system call traces of installed software are recorded both while correctly used and misused. Then, those traces are used as training set on the artificial intelligent system. The trained system is then used to qualify system call traces from the productive running software and to alarm the staff when it finds such anomalous activity.

### 3.5.3 Architectural Improvements

There are proposals for architectural improvements of intrusion detection systems. They don’t affect the accuracy of a sensor but the scalability of intrusion detection systems for large networks. Mell et al. [9] compare distributed approaches to hierarchical ones.

### 3.5.4 IDS Software Evaluation

In order to evaluate IDS software a suitable test environment is needed. Champion [12] describes the network used by the Air Force Research Laboratory for intrusion detection software evaluation. It features a sophisticated traffic generation system that allows testing under realistic circumstances.

### 3.5.5 Reactions

In the previous sections we presented work that addresses the intrusion detection systems themselves. It remains the question what to do when an incident is reported by an IDS. After a successful attack the equipment in the network needs to be restored to normal operation as soon as possible. Yuill et al. [11] describe ways of handling successful intrusions. They mainly propose to use the traces an intruder left in the system for repair and give a scheme for efficient processing of incidents.

### 3.6 Summary

As the section about related work shows, current work in intrusion detection mainly aims at detecting suspicious activity in huge amount of event data, using many different approaches. We are not aware of any work addressing intrusion detection with regard to reducing false positives by taking further investigations
in an environment as described in section 3.1. Therefore we will implement our proposed extensions from section 3.4.

Note that (as explained in section 3.1) an important requirement is that our extensions must not increase the number of false negatives. To make sure that this condition holds, we have to classify attacks. In chapter 4 we describe why and how we classified known attacks and vulnerabilities. In chapter 5 we describe how we implemented our extensions.
Chapter 4

Classification of Attacks

In section 3.3 we presented the idea of a smart IDS filtering alerts from a conventional IDS based on further investigations. In the current chapter we explain that the information from investigations could be misleading since an attack could have changed the victims behaviour. In order to be sure to only discard alerts that haven’t been successful, we classify the attacks and present meaningful combinations of investigations for each class.

4.1 Investigations

There are two ways to investigate. One is to have a database with information about the protected network. When an attack is detected, the SIDS can consult this database to see whether or not this attack could have been successful. The second way is to collect information after the attack has been detected and based on this to try to find out whether or not the detected attack has been successful.

The first three extensions proposed in chapter 3 act as information collecting units. The availability tester and the triggered scanning extension provide information about the system behaviour after an attack whereas the configuration database holds information collected before an occurrence attack.

4.1.1 Pre-Attack Investigations

As discussed in section 3.4.3, the problem with investigations before occurrence of an attack is the accuracy of the information the configuration database contains at the time it is queried. There are several ways to insert information into this database, including the following two possibilities.

- The network operators update the information. We call this possibility static configuration database.

- The configuration information is collected by automated auditing of the network. This possibility is called dynamic configuration database.

If server hosts are to be protected, the network operators would be able to keep the configuration database up to date, since the servers are well-known by the operators. Changes to the configuration of a server are only done by them. It remains a question of policy whether or not the operators are believed to update the database accurately.
However, if we want to protect arbitrary hosts in a network, it is not feasible to have all users report changes in their systems. Therefore, automated auditing is preferred. It remains a question of the site policy in which intervals automated auditing is carried out and whether or not this information is considered reliable.

As we just saw, it makes sense to distinguish between two types of hosts. One type of hosts we know much about and this knowledge is reliable and another type of hosts we don't know anything. Section 5.3.1 describes how we implemented this distinction in our smart IDS.

4.1.2 Post-Attack Investigations

All investigations that take place after detection of an attack have a problem in common. The behaviour of a system could have changed with the attack, as illustrated in figure 4.1. Therefore, we could see a system behaving completely differently after an attack than before. For example, a successful DoS attack against a service changes the system behaviour in the way that a service was running before but is no more running after the attack. Therefore, investigations after an attack could yield misleading information.

A solution to the described problem is to classify the attacks by their impact on the investigations and to take different combinations of information sources into account for each class, as presented in the next section.

4.2 Classification

We want to classify the alerts by their impact on the investigations. Since the post-attack investigations are done by the availability tester and the triggered scanner extensions, we divide the attacks into the following classes:

1. The first class comprises attacks that do not have any impact on the investigations.

2. The second class consists of attacks whose impacts on the investigations are only regarding service shut down (denial of service).

3. The third class covers attacks that potentially have any impact on the investigations.

\textsuperscript{1}It is important to distinguish between what kind of impact attacks have from the viewpoint of our investigations and what they really do. For example, imagine an attack that may alter some system configuration and then kills the service it exploited. Our investigations might only observe the denial of service effect but the attack possibly caused more damage.
This classification is illustrated in figure 4.2. With attacks of the first class, we are allowed to take all post-attack investigations into account. This is clear because by definition of this class, an attack did not change the system behaviour in a way that potentially fools our investigations. If we encounter an attack of the third class, we can not rely on any investigation after the attack and therefore are bound to incorporate at most our configuration database. Attacks of the second class require a clever combination of investigations.

![Figure 4.2: We divide all attacks into three classes depending on their impact on our investigations.](image)

### 4.3 Attack-Class Dependent Use of Investigations

Since the use of a configuration database depends on the site policy, we describe the use of investigations depending on the configuration database type.

#### 4.3.1 Ideal Configuration Database

In this section we present the combinations of extensions for each class for the ideal case where the configuration database always contains accurate information.

**Attacks without Impact on Investigations**

For the class of attacks without impact on the extensions we are allowed to use all of the three extensions. A meaningful combination uses the triggered scanner to discard alerts if it reports not vulnerable. If the triggered scanner doesn’t know how to check for a specific attack, the configuration database or the availability tester can be consulted to find out whether or not there was a corresponding service running. Additionally, the configuration database and the availability tester extensions can be used to detect errors in the classification of attacks. If the configuration database and the availability tester don’t report the same (either service running or no service running), the attack was not an attack without impact and the alarm should be passed on.

**Attacks with Denial of Service Impact Only**

With attacks of the denial of service only class the triggered scanner must be combined with the availability tester, since a successful denial of service attack yields a no more available service. This fools the scanner into thinking that the host is not vulnerable. The alert should be dropped only if the scanner returns not vulnerable 

*and* the service tester indicates a running service. Independently,
if according to the configuration database the host is not vulnerable, the alert can be dropped, too.

If we additionally include the information on what the attack really does, we can optimize the use of investigations for this class. If we are sure that an attack of this class really does nothing more than a DoS, we can rely on the availability tester alone. This is an optimization since the vulnerability scanner may not know a certain attack and therefore we must assume that the host is vulnerable. If we do not have to incorporate the vulnerability scanner, we can drop more alerts without increasing the number of false negatives. This additional separation is shown in figure 4.3.

![Figure 4.3](image-url)

**Figure 4.3:** We divide all attacks into three classes depending on their impact on our investigations. Additionally, with the denial of service class, we distinguish between attacks that have a denial of service effect only, and attacks that have a denial of service impact on the investigations but may have other effects not seen by the investigations, too.

**Attacks with Potentially any Impact**

If it comes to attacks of the class that potentially can have any impact on the extensions, post-attack investigations cannot be used. The configuration database is the only source of information that can be relied on. If it reports that the target host was not vulnerable, the alert can be dropped. Otherwise, the alarm has to be passed on.

**4.3.2 Real World Configuration Database**

In a real world situation, we have to give up the assumption of an always accurate configuration database. As described in section 4.1.1, the system may have changed between the last actualization and the attack. We present possible site policies together with their advantages and disadvantages for three different cases of configuration database maintenance. Table 4.1 summarizes these thoughts.

**No Configuration Database**

If we do not want to rely on a configuration database at all, we have to modify the combinations of investigations as follows.

Attacks that don’t have impact on the investigations are only dropped if the triggered scanner reports not vulnerable or if according to the availability tester there is no service running. If there is a service running, the scanner has to decide whether or not this service is vulnerable.
With attacks from the denial of service only class we use the combination of triggered scanner and availability tester but we can’t use the configuration database to drop alerts any more.

An attack that potentially can have any impact must be passed on since all three extensions yield possibly wrong information.

The disadvantage of this realization is obvious: there is less decrease in false positives compared to the case with an ideal configuration database. However, there is no increase in false negatives.

**Dynamic Configuration Database**

If we want to make use of a dynamic configuration database, the rules would basically be the same as in the ideal case. We must be aware that the decrease in false positives is paid with possible false negatives due to the inaccuracy of the configuration database. However, this inaccuracy can be lowered by updating the database more frequently.

**Static Configuration Database**

Since a static configuration database is maintained by the operator of a network, the content is accurate. Thus, the reduction in false positives is not paid with false negatives but with the additional administration tasks.

<table>
<thead>
<tr>
<th>Real World DB</th>
<th>Static DB</th>
<th>Dynamic DB</th>
<th>No DB at all</th>
</tr>
</thead>
<tbody>
<tr>
<td>needs</td>
<td>configuration must be reported</td>
<td>information potentially outdated</td>
<td></td>
</tr>
<tr>
<td>potentially increase in false negatives</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decrease in false positives</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>appropriate usage</td>
<td>important hosts</td>
<td>mass of less important hosts</td>
<td>no information about configuration available or database undesired</td>
</tr>
</tbody>
</table>

*Table 4.1:* Configuration databases are a way to decrease false positives. But this decrease comes with disadvantages.

### 4.4 Summary

The classification of attacks by their impact on the investigations allows the use of investigations without violating the requirements.
Chapter 5

SIDS Implementation

This chapter explains how we implemented the ideas and concepts presented in chapters 3 and 4. We implemented\(^1\) all proposed extensions except the concept of severity levels because in contrast to the other extensions there are already a lot of people working at similar concepts.

5.1 Overview

Normally, when a network-based IDS detects a pattern stored in its database of attacks, it looks up in its configuration whether or not an alert has to be generated. This configuration is static such that there are two sets of patterns: patterns that generate alerts and patterns that won’t. Further information about the circumstances under which the pattern has been observed are not taken into consideration. In most cases the configuration of the IDS is such that there are more false positives than false negatives.\(^2\) This is because false negatives — from the viewpoint of a customer of security solutions — have greater impact than false positives.

To reduce the number of false positives with our smart IDS, alerts generated by an IDS will first be sent via what we call the *abstracting interface* to the *SIDS add-on* which then decides whether or not to pass on the alert. It is very important that the SIDS will only discard alerts if they are false positives for sure. Every decrease in the amount of false positives is very welcome but the number of false negatives must not increase under any circumstances.

The SIDS add-on will base its decision on the results of investigations taken by so called *modules*. They implement the smart extensions presented in chapter 3. An overview of the involved components is given in figure 5.1 and the subsequent sections describe them in greater detail.

5.2 Abstracting IDS Interface — The Dragon Mail-Parser

As we have described in section 3.3 our SIDS add-on, which is independent of the underlying IDS, needs a so called abstracting IDS interface which connects the add-on to a specific intrusion detection product. In our case the underlying IDS

\(^1\)The main part of our code is written in Perl (Practical Extraction and Reporting Language).

\(^2\)The terms false positive and false negative are explained in section 1.3.
Abstracting IDS Interface

SIDS Add-On

Attack Information

Hand On or Discard

M0 Host Tester

M1 Service Tester

M2 Scanner

M3 Static DB

M4 Dynamic DB

Figure 5.1: Overview of the Smart IDS and its Components.
product is Dragon\(^3\) because this is the IDS product Open Systems AG uses for their managed security services. To connect Dragon to the SIDS add-on, we have chosen to use its facility to send out alerts via mail upon detection of an attack pattern. Therefore we call this particular abstracting interface Dragon mail-parser.

When the mail-parser receives a real-time alert from the intrusion detection system it extracts the attack type and information about the victim host from it. Figure 5.2 illustrates the mail-parser and figure 5.3 shows two examples for real-time alerts. It then converts the Dragon specific name of the attack to a CVE number using the \texttt{dragon2cve} file. As an example, figure 5.4 shows a part of the \texttt{dragon2cve} file.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure52.png}
\caption{The Dragon mail-parser receives alerts from the Dragon IDS. With the help of the \texttt{dragon2cve} file it extracts attack information out of it and calls the SIDS-core. Based on the return value of the SIDS-core it then discards or passes on the alert.}
\end{figure}

For some investigations the SIDS add-on needs to know the hostname of the victim. Since this information is not included in the Dragon alert mail, the mail-parser looks it up in the Dragon log database.\(^4\) If this is not possible the mail-parser assigns a NULL-string to the victim hostname. Finally it passes the extracted information (CVE number, victim IP, attacked port, and victim hostname)\(^5\) to the SIDS add-on for further investigation.

In the following cases the mail-parser will directly pass on the alert without even calling the SIDS-core.

- The mail-parser could not extract the necessary information from the mail it received. For example because it did not understand the format of the mail that contained the real-time alert.

- The mail-parser could not open the \texttt{dragon2cve} file, which contains the necessary information to convert the proprietary attack-names to CVE numbers.

- The mail-parser could not convert the name of the attack (given in the mail) to a CVE number because the corresponding information is missing in the \texttt{dragon2cve} file.

\(^3\)See section 2.4.1 for more information about the Dragon IDS product.

\(^4\)This implies that the SIDS runs on the same host as the Dragon server.

\(^5\)From now on referred to as attack information.
From dragon  Fri Dec 15 11:39:25 2000
Return-Path: <dragon@localhost.localdomain>
Received: (from dragon@localhost)
    by localhost.localdomain (8.11.0/8.8.7) id eBFAdPR16871
    for root@localhost.localdomain; Fri, 15 Dec 2000 11:39:25 +0100
Date: Fri, 15 Dec 2000 11:39:25 +0100
From: dragon <dragon@localhost.localdomain>
Message-Id: <X00012151039.eBFAdPR16871@localhost.localdomain>
To: root@localhost.localdomain
Subject: Dragon Real Time Alert

***This is a Real Time Alert***

The following event(s) were detected as of:

00Dec15 (Friday) at 11:39:25

2000-12-15 11:30:45 I 192.166.50.3 192.168.50.71 [ICMP=BACKDOOR]
    (icmp.type=8.id=0) dsensor

From dragon  Fri Dec 15 11:39:25 2000
Return-Path: <dragon@localhost.localdomain>
Received: (from dragon@localhost)
    by localhost.localdomain (8.11.0/8.8.7) id eBFAdPR16871
    for root@localhost.localdomain; Fri, 15 Dec 2000 11:39:25 +0100
Date: Fri, 15 Dec 2000 11:39:25 +0100
From: dragon <dragon@localhost.localdomain>
Message-Id: <X00012141713.eBFAdPR16871@localhost.localdomain>
To: root@localhost.localdomain
Subject: Dragon Real Time Alert

***This is a Real Time Alert***

The following event(s) were detected as of:

00Dec14 (Thursday) at 18:13:02

2000-12-14 18:04:15 I 192.166.50.2 192.168.50.6 [WEB:D7T-D7T]
    (tcp.dp=80.sp=1036) dsensor

Figure 5.3: This figure shows two examples for real-time alert mails generated by the Dragon intrusion detection system.
[SMB:CD-DOT-DOT];1999-0179
[SMB:SAMBA-CLIENT];1999-0391
[SNTP:DEBUG];1999-0006
[SNTP:PIPE-DECODE];1999-0096
[SSH:VERSION-1];1999-0310
[TEL:PM-DOS];1999-0218
[TEL:HOST-PLUS];1999-0326
[TFTP:Linux-DOT-DOT];1999-0183
[UDP-BMB;CYBERCUP];1999-0103
[WEB:CGI-AGLIMPS];1999-0147
[WEB:CGI-CAMPAS];1999-0146

Figure 5.4: This is a part of the dragon2cve file which is used to convert the attack names reported by the Dragon IDS to CVE numbers for further processing by the SIDS.

In the following case the mail-parser will directly discard the mail containing the alert without calling the SIDS-core.

- The mail received by the mail-parser is not a real-time alert from the intrusion detection system. Whether this is the case or not is decided by checking the subject line of the mail.

Having received a request from the mail-parser, the SIDS-core further investigates the attack which has been detected by the IDS. It then tells the mail-parser whether the alert should be discarded or passed on.

The mail-parser logs all its activity to a dedicated log-file.

5.3 The SIDS-Core

The SIDS-core is being called by the mail-parser and given the information about the observed attack. As shown in figure 5.5 and figure 5.10, a so-called module-caller with several modules is attached to the SIDS-core. Given information about the observed attack, each of these modules is able to gather further information which helps to investigate the success of an attack. Depending on the type of the detected attack the SIDS-core will tell the module-caller to start some of these modules. Then based on their results (which are boolean return values) it will decide whether the mail-parser shall discard or hand on the alert.

As explained in the previous chapter, for different types of attacks different kinds of investigations are needed. Therefore we introduced tables which hold boolean expressions for CVE numbers. The boolean expressions specify which modules have to be interrogated and how their answers have to be combined. If a boolean expression evaluates to 1 the alert will be discarded, otherwise the alert will be passed on. A file containing such a table of expressions is called a policy-file. As an example a part of a policy-file is shown in figure 5.6. The syntax of a line is as follows:

| CVE number; bypass-flag; boolean expression |

Each line contains instructions concerning a specific CVE number. The bypass-flag is a boolean value. If it is set to 1, the following boolean expression will be
Figure 5.5: The SIDS-core receives information about an observed attack from the mail-parser. Based on the rules specified in the policy it then starts different modules via the module-caller.

ignored and all alerts regarding this CVE number will be passed on directly without further investigations.

For example the occurrence of an attack corresponding to the CVE number 1999-0060 requires the interrogation of modules M1 and M3. If either M1 returns a false value or M3 returns a true value, the SIDS-core will tell the mail-parser to discard the alert. In contrast, on occurrence of an attack corresponding to CVE number 2000-0244 the SIDS-core will tell the mail-parser to hand on the alert without even calling any modules.

1999-0202;0;M3
2000-0244;1
1999-0060;0;|M1||M3
1999-0148;0;(|M1&M2)||M3
1999-0278;0;|M0||M1||M2||M3
1999-0759;0;|M0||M1||M2||M3
1999-0218;0;(|M1[1]|M2)|M3[0]

Figure 5.6: This is a part of an example policy-file which is used to determined what modules have to be started given a particular attack and how the results of the modules have to be combined to determine whether or not an alert has to be discarded.

To make sure that deciding whether or not to discard an alert does not take too long, the SIDS-core stops all modules after some time\(^6\) regardless of whether or not the modules have completed their work. Using square brackets one may optionally specify default values in the boolean expressions of the policy-file. If a module does not complete its work in time, these default values are being used to evaluate the boolean expression. As an example let’s look at the boolean expression for the CVE number 1999-0218 in the above example policy-file. It reads\(^6\)

\(^6\)This time period may be specified in the configuration section of the corresponding program.
5.3 The SIDS-Core

(!M1 \&\& M2) | | M3 [0]. Assume an attack of type CVE-1999-0218 has been reported to the SIDS-core by the mail-parser. The SIDS-core would then start the modules M1, M2 and M3. Let’s further assume that module M1 returns the boolean value 0, module M2 returns 1 and module M3 times out. The boolean expression would then become (1 \&\& 1) | | 0 and evaluate to 1. Thus the SIDS-core would tell the mail-parser to discard the alert. If a module times out and there is at least one occurrence of this module in the boolean expression without default value, the alert is being passed on, as if the whole expression had evaluated to 0.

One could also have had the module-caller assign a default return value to each module (for example M1 in the boolean expression always evaluates to 0 in case the module times out). But this wouldn’t be a good approach because in case of a timeout we would like to assume a conservative return value\(^1\) which in turn depends on the context the module is used in. Therefore we have chosen to specify default values within the boolean expressions.

The following example illustrates why the conservative return value depends on the context the module is used in: Assume we have a module M1 that checks the availability of a specific service on a host (returning 1 if the service is available, 0 otherwise) and a module M2 that checks the existence of a specific security vulnerability (returning 0 if the host is vulnerable, 1 otherwise). We would like M1 to check the availability of a specific service on a host, because we have observed a DoS-attack against it. A suitable boolean expression defining when to discard the alert would be “M1” and in case M1 would time out we would assume the corresponding conservative return value 0. Now let’s think of another situation where we have observed a non-DoS attack exploiting a known security vulnerability. In this case we could ask the scanner-module whether it thinks that the alert has been successful or not. But since the scanner might not know this vulnerability we could, to improve the effectiveness, also additionally employ the availability checker. We would therefore use the rule “!M1 | | M2” because if the corresponding service does not run after the non-DoS attack it did not do so before and therefore the attack could not possibly be successful. In contrast to the above situation, the conservative return value of M1 would be 0.

As one can see from the above explanations we have two important requirements regarding the SIDS-core:

- A not properly working module should not be able to block the SIDS in its work processing the alert.
- In case of doubt the alert should always be passed on. That is we do not want to have an increase in the amount of false negative alerts.\(^8\)

In the following cases the SIDS-core will tell the mail-parser to pass on the alert without even calling any modules:

- The SIDS-core cannot open the policy-file.
- There is no entry regarding this CVE number in the policy-file.
- The corresponding entry in the policy-file has its bypass-flag set to 1.

The SIDS-core logs all its activity to a dedicated log-file.

\(^1\)Here conservative means we’d like to assure that the timed-out module returns the value it would have if the attack had been successful.

\(^8\)This follows directly from the requirements listed in section 3.1.
5.3.1 Separated Policies

As we outlined in section 3.4.3, it’s useful to have separate policy-files for different types of hosts. This is because there are hosts that are more important than others and thus need more restrictive rules.

![Diagram showing separated policies]

We implemented a general scheme for separated policies, as illustrated in figure 5.7. The policy configuration file maps IP addresses or ranges to a policy-file name. Additionally, it is possible to specify a default policy-file. The syntax of one line in the policy configuration file is as follows:

| IP Range: policy-file name |

There are three possibilities for the IP Range entry: a single IP address in the notation \(X.X.X.X\), an IP address range in the format \(X.X.X.X-Y.Y.Y.Y\) or the reserved word *default*. After the (mandatory) semicolon and (optional) white-spaces, the corresponding policy-file name follows. Figure 5.8 shows an example policy configuration file.

```
192.168.50.2-192.168.50.10; policies/main servers
192.168.50.1; policies/firewall
default; policies/default
```

Figure 5.8: The policy configuration file assigns a policy-file to an IP address range, allowing the use of different policy-files for different hosts. The figure shows an example policy-file.

5.3.2 Macros in the Policy-Files

To make it more comfortable to assemble the policy-files, we have included the possibility to use macros. This means, you can assign a name to a given boolean expression and later on when specifying rules you may use this name instead of the expression itself. The syntax of a line defining a macro is as follows:

```
%macroame=expression
```

First there is a % sign followed by the name for the macro, then there is a = sign followed by the expression associated with the given macro name. All macro
definitions must be at the top of the particular file, before the first occurrence of a rule. The scope of macro definitions is the specific policy-file they are stated in. Figure 5.9 shows a part of an example policy-file using macros.

\%
example=M0 | M1 | M2 | M3
1999-0202:0;M3
2000-0244:1
1999-0060:0;M1 | M3
1999-0148:0;(!M1&&M2) | M3
1999-0278:0;\%example
1999-0739:0;\%example
1999-0218:0;(!M1[1]&&M2) | M3 [0]

Figure 5.9: This is part of an example policy-file using a macro to define some of the rules.

5.4 Module-Caller

The module-caller is the interface between the SIDS-core and the module implementations. Invoked by the SIDS-core with the module number and the attack information, it passes the attack information to the corresponding module and checks the module’s return value for validity before returning. As illustrated in figure 5.10, the module-caller has its own log-file.

![Module-Caller Diagram](image)

Figure 5.10: The module-caller has several modules attached to it. These modules help to take further investigations to find out whether the attack has been successful or not.

When new modules are being integrated into our smart IDS add-on, the SIDS-core needs not to be modified. The new module only has to be registered in the module-caller.
5.5 Modules

This section describes the implementation of the smart IDS extensions. As shown in table 5.1, every module has a symbol which must be used when specifying rules in policy-files.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>M0</td>
<td>Host Tester</td>
</tr>
<tr>
<td>M1</td>
<td>TCP Port Tester</td>
</tr>
<tr>
<td>M2</td>
<td>Vulnerability Scanner</td>
</tr>
<tr>
<td>M3</td>
<td>Static Configuration Database</td>
</tr>
<tr>
<td>M4</td>
<td>Dynamic Configuration Database</td>
</tr>
</tbody>
</table>

Table 5.1: Our five modules and their corresponding shortcuts.

The modules implement our proposed SIDS extensions. They are being called with the attack information\(^9\) and return a binary value. Each module logs all its activity to a dedicated file.

5.5.1 Host Tester

One of the availability tester extensions is the host tester module, implemented in the SIDS\(_{hosttester}\) package. As illustrated in figure 5.11, it mainly is a wrapper around the well known ping command. It is used to test whether or not a host is up and reachable at the moment.

![Diagram of Host Tester](image)

Figure 5.11: The host tester uses the well known ping command in order to check whether or not a host at the victim IP address is up.

Like all modules, the host tester module gets the attack information from the module-caller, but it only makes use of the IP address.

Table 5.2 provides a short overview of the host tester module. If the host replies, the module returns 1, otherwise the module returns 0. For this module, the return values can not be interpreted as the module's opinion on whether or not to discard an alert. If the attack has been a denial of service attack, a dead host indicates that the attack has been successful. But if the observed attack does not have denial of service effect, a dead host indicates that the alert can safely be dropped.

---

\(^9\)Attack information refers to the CVE number, the victim IP address, the attacked port-number, and the victim's hostname.
5.5 Modules

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Host Tester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl Package</td>
<td>SIDS_{hosttester}</td>
</tr>
<tr>
<td>Requirements</td>
<td>ping</td>
</tr>
<tr>
<td>Call</td>
<td>hosttester(CVE, IP, port, hostname);</td>
</tr>
<tr>
<td>Returns 1</td>
<td>if the host is alive</td>
</tr>
<tr>
<td>Returns 0</td>
<td>if the host does not answer</td>
</tr>
<tr>
<td>Logfile Name</td>
<td>hosttester.log</td>
</tr>
</tbody>
</table>

Table 5.2: Host Tester Module Overview.

5.5.2 TCP Port Tester

The TCP port tester module also belongs to the availability tester extension. With this module the information on whether or not there is a service listening on a port can be incorporated into the SIDS rules. The module tries to connect to IP:port using standard stream sockets, as illustrated in figure 5.12.

Figure 5.12: The TCP port tester uses standard stream sockets to check whether or not there is a service listening on the victim IP:port.

Table 5.3 provides a short overview of the TCP port tester module. If the module is able to connect to a service it returns 1, otherwise 0. As with the host tester module, the return value of this module cannot be interpreted as its opinion on whether or not to discard an alert. It has to be seen in the context of the attack class.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>TCP Port Tester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl Package</td>
<td>SIDS_{tcptester}</td>
</tr>
<tr>
<td>Requirements</td>
<td>Socket perl standard package</td>
</tr>
<tr>
<td>Call</td>
<td>tcptester(CVE, IP, port, hostname);</td>
</tr>
<tr>
<td>Returns 1</td>
<td>if there is a service listening on IP:port</td>
</tr>
<tr>
<td>Returns 0</td>
<td>if there is no service listening on IP:port</td>
</tr>
<tr>
<td>Logfile Name</td>
<td>tcptester.log</td>
</tr>
</tbody>
</table>

Table 5.3: TCP Port Tester Module Overview.

5.5.3 Vulnerability Scanner

The vulnerability scanner module implements the triggered scanner extension. Table 5.4 provides a short overview of the vulnerability scanner module.
<table>
<thead>
<tr>
<th>Module Name</th>
<th>Vulnerability Scanner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl Package</td>
<td>SIDS_scanner</td>
</tr>
<tr>
<td>Requirements</td>
<td>Nessus; nasl interpreter and plugin scripts</td>
</tr>
<tr>
<td>Call</td>
<td>scanner(CVE, IP, port, hostname);</td>
</tr>
<tr>
<td>Returns 1</td>
<td>if Nessus reports that it could not verify the vulnerability. This event happens if the host really is not vulnerable or if the script could not execute properly.</td>
</tr>
<tr>
<td>Returns 0</td>
<td>if no Nessus script is found, the script does not check the correct port, the hostname parameter is empty, or Nessus reports that the host is vulnerable.</td>
</tr>
<tr>
<td>Logfile Name</td>
<td>scanner_log</td>
</tr>
</tbody>
</table>

Table 5.4: Vulnerability Scanner Module Overview.

For our SIDS implementation we chose \textit{Nessus} \cite{26} among different scanners available. Its main advantages are that it is distributed under GNU Public License and that the checks are specified using a scripting language (NASL). Therefore, new checks are available from many sources and can easily be integrated.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.13.png}
\caption{Nessus features a client-server architecture allowing remote-auditing of a network. The checks are carried out by the NASL interpreter. The gray box indicates the parts of Nessus we use to implement the vulnerability scanner module.}
\end{figure}

Nessus is intended as an audit tool. As figure 5.13 illustrates, Nessus features a client-server architecture, allowing the administrator to place the server in the network to be audited and control the tests from outside. To carry out the tests, the server uses a NASL interpreter.

For the implementation of our triggered scanner we use \texttt{nasl}, the command line NASL interpreter. Thus we are able to scan an arbitrary host for just one vulnerability.

If Nessus reports a \textit{successful scan}, the vulnerability exists on the checked host and the module returns 0, indicating that the alarm should not be dropped. However, if Nessus reports an \textit{unsuccessful scan}, either the vulnerability really does not exist or the script was not able to execute properly. For example, this happens if there is no service listening on the port the script checks. In these cases, the module returns 1.
This ambiguity regarding the meaning of the return value 1 arises because Nessus is a vulnerability scanner that is intended to check one or more hosts for known vulnerabilities. It reports vulnerable only if it is sure to have found a security hole. In our case we want to be sure that a certain vulnerability does not exist. As a consequence, for denial of service attacks, the use of the scanner module must be combined with the port tester module. An unsuccessful scan result could be caused by a service that ran before the attack but is no more running (which rather was a successful attack).

![Diagram](image)

**Figure 5.14:** In order to achieve the functionality of the triggered scanning extension we introduce three tests before Nessus is invoked.

In our implementation with Dragon, the problem of a triggered scan provoking a new alert described in section 3.4.2 is handled in the Alarmtool component. It doesn’t raise alerts for previously generated ones.

There are several issues with the implementation of triggered scanning using Nessus. They are discussed below. In order to solve those issues, three tests are introduced before Nessus is invoked, as is illustrated in figure 5.14.

### Destructive Scans

There are Nessus scans that check for a vulnerability by executing a corresponding attack. This is no problem for non-destructive attacks, for example getting a file below the document root of a web server. However, we don’t want to check for a vulnerability by carrying out destructive code.

Fortunately, NASL specifies a command that identifies the type of the script. This type qualifies the script, that is the way it checks, not the attack it is checking for. The NASL command `script_category()` accepts the following arguments: `ACT_GATHER_INFO`, `ACT_SCANNER`, `ACT_ATTACK`, and `ACT_DENIAL`, whereas the latter two are considered destructive. Our scanner module checks a script on-the-fly for the `script_category()` entry. Nessus is only being invoked if the module finds either `ACT_GATHER_INFO` or `ACT_SCANNER` as `script_category()` argument. Otherwise, Nessus will not be invoked and the module returns 0.

### Fixed TCP Ports

Using Nessus, a problem with TCP port numbers arises. We have to assure that the triggered scanner checks for the same vulnerability on the same host on the
same port as reported by the IDS. Vulnerabilities are distinguished by their CVE numbers, hosts by their IP addresses and ports by their numbers. But we can not assure that a certain Nessus script checks on the same port the IDS reported. This is because the script language allows hardcoded port numbers. An author of a script could open a socket or even forge a packet with an arbitrary destination port number. Therefore, we introduced a table translating CVE numbers to TCP port numbers used by the corresponding Nessus scripts. The file containing the table is called `cve2nessus` and is being described below. Before invoking Nessus, the (CVE, port) tuple is checked against this table and only if a match is found the scan is started. Otherwise, the module returns 0.

**HTTP Virtual Hosts**

The HTTP protocol offers the possibility to address the hostname of a webserver independently of the IP address. This allows to host several different domains on one and the same IP address. On the server side the HTTP daemon has to support so called virtual hosts, and the client has to specify the host HTTP header field.\(^\text{10}\)

If we want to verify a web-based attack, the scanner needs to know which host to check. The IP address alone is not sufficient. It might be that a webserver hosting several virtual hosts has a vulnerable CGI program only in one of its virtual hosts. If the scanner doesn’t check on the correct virtual host, the result may be wrong. Therefore, if the module has to check for a web-based vulnerability, the hostname parameter is mandatory. If it is not available in this case, the module returns 0.

Whether or not a certain vulnerability is web-based is specified in the `cve2nessus` file, too. A line in this file is of the following form:

| CVE number;port;host-flag |

Figure 5.15 shows an excerpt of a `cve2nessus` file.

1999-0449;80;1
1999-0660;30100;0
2000-0543;4000;0
2000-0665;23;0
1999-0178;80;1
1999-0383;23;0
1999-0776;80;1

**Figure 5.15:** The `cve2nessus` file holds information extracted from Nessus scripts. Every line specifies for the corresponding script which TCP port it checks and whether or not the attack is web-based, and therefore whether or not the hostname is needed.

For every CVE number, the port field denotes the port the corresponding script checks on. The host-flag indicates whether the attack is web-based (1) or not (0).

We wrote a script that automatically determines the port and whether or not the check is web-based out of naas scripts. This script is called `gen_cve2nessus.pl` and is included on the enclosed CD. See appendix D.2 for more details.

\(^{10}\)The host header field exists since HTTP 1.0 and became mandatory for HTTP 1.1 requests.
5.5.4 Configuration Database

The configuration database module is a simple way of incorporating pre-attack information into our smart IDS add-on. The database stores (IP address, TCP port) tuples. Entries indicate that on the corresponding host's port, there is no service running. Figure 5.16 illustrates the module's architecture, and table 5.5 gives a brief description of the configuration database module.

![Diagram of configuration database module]

Figure 5.16: Since the two modules M3 and M4 share the same functionality, they are implemented in one package.

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Configuration Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perl Package</td>
<td>SIDS.db</td>
</tr>
<tr>
<td>Requirements</td>
<td>nmap for dynamic database updates.</td>
</tr>
<tr>
<td>Call</td>
<td>static(CVE, IP, port, hostname); dynamic(CVE, IP, port, hostname);</td>
</tr>
<tr>
<td>Returns 1</td>
<td>if IP:port was found in database</td>
</tr>
<tr>
<td>Returns 0</td>
<td>if IP:port was not found in database</td>
</tr>
<tr>
<td>Logfile Name</td>
<td>db.1og</td>
</tr>
</tbody>
</table>

Table 5.5: Configuration Database Module Overview.

More precisely, the module allows to query two functionally identical databases. One database, called static database, is filled by the network operator. As the name of the other database suggests, the entries in the dynamic database are generated by a script (gen_dynamic_db.pl, see appendix D.5 for more details) that automatically checks hosts for closed ports. This distinction is necessary for flexibility when it comes to write policies for different hosts. Depending on the type of host a policy rule is written for, the dynamic database may be trusted or not.

The two databases are stored in two separate files (dbstatic and dbdynamic). Lines are of the following form:

<table>
<thead>
<tr>
<th>IP Address:Port Specification</th>
</tr>
</thead>
</table>

The IP address must be specified in the standard notation form X.X.X.X. For the port specification, either * (indicating all ports closed) or a port set can be specified. A port set consists of port ranges separated by commas. A port range
is either a single port number, or a range of ports notated as `startport-endport`. Start and end port are treated inclusive. Figure 5.17 shows an excerpt of an example configuration database.

```
192.168.50.71;1-20.22-79.81-65535
192.168.50.100;*
192.168.50.101;*
```

**Figure 5.17:** In the configuration database file, one line per IP address specifies the closed ports of the corresponding host. For example, the host on IP address 192.168.50.71 only runs services on port 21 and 80 (FTP and HTTP).

### 5.6 Summary

The way we implemented our smart IDS extensions guarantees that the requirement of no increase in false negatives is held. Furthermore, our implementation is independent of the underlying IDS.
Chapter 6

Accuracy Comparison

In the previous chapter, we presented the smart IDS implementation we developed to achieve the goals of our diploma thesis. This chapter gives a description of the tests we performed and shows the results of a comparison between a conventional IDS and our smart IDS regarding the number of false negatives.

6.1 Test Environment

To compare the accuracy of an IDS using our smart IDS add-on with an IDS not using it, we set up the environment shown in figure 6.1. An attacker runs a set of exploits against a victim host while a Dragon IDS is monitoring the conversation. From inspecting the log-files, we can tell how many false positive alerts have been discarded by our add-on.

![Diagram](image)

**Figure 6.1:** The test environment we set up to test the increase of accuracy resulting from our smart IDS add-on.

61
6.1.1 The Attacker

The results of the accuracy comparison will of course depend on the exploits run by the attacker. If there are a lot of unknown or successful exploits, the increase in accuracy will only be small because alerts regarding unknown or successful attacks will pass our add-on. On the other hand, if there are a lot of unsuccessful exploits and our SIDS add-on is able to detect them as such, the increase in accuracy will be very large. From this we can see that it does not make sense to assemble a set of exploits by hand. If we did so, we could choose an unfair set which makes the improvement seem larger than it really is. To have a fair test set of exploits run against the victim we have chosen to let the attacker run the Nessus auditing tool with all its currently available plug-ins.

Note that Nessus only runs attacks against ports on the victim host he has previously found as open. Therefore, the database-module with its current functionality (see section 5.5.4) could not help to decrease the number of false positives. This means that in a real-world situation we could have discarded even more false positive alerts.

6.1.2 The Victim

We ran two tests: one with an old version of Linux (RedHat 5.2) and the other one with Windows NT (4.0 Server, Service Pack 1.0). Both operating systems are likely to be vulnerable to some of the attacks in our test set and not vulnerable to others. Figures 6.2 and 6.3 show the open ports of the two hosts as reported by an Nmap-scan.

Interesting ports on (192.168.50.9):
(The 65515 ports scanned but not shown below are in state: closed)

<table>
<thead>
<tr>
<th>Port</th>
<th>State</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>21/tcp</td>
<td>open</td>
<td>ftp</td>
</tr>
<tr>
<td>23/tcp</td>
<td>open</td>
<td>telnet</td>
</tr>
<tr>
<td>25/tcp</td>
<td>open</td>
<td>smtp</td>
</tr>
<tr>
<td>37/tcp</td>
<td>open</td>
<td>time</td>
</tr>
<tr>
<td>53/tcp</td>
<td>open</td>
<td>domain</td>
</tr>
<tr>
<td>70/tcp</td>
<td>open</td>
<td>gopher</td>
</tr>
<tr>
<td>79/tcp</td>
<td>open</td>
<td>finger</td>
</tr>
<tr>
<td>80/tcp</td>
<td>open</td>
<td>http</td>
</tr>
<tr>
<td>98/tcp</td>
<td>open</td>
<td>linuxconf</td>
</tr>
<tr>
<td>109/tcp</td>
<td>open</td>
<td>pop-2</td>
</tr>
<tr>
<td>110/tcp</td>
<td>open</td>
<td>pop-3</td>
</tr>
<tr>
<td>111/tcp</td>
<td>open</td>
<td>sunrpc</td>
</tr>
<tr>
<td>113/tcp</td>
<td>open</td>
<td>auth</td>
</tr>
<tr>
<td>119/tcp</td>
<td>open</td>
<td>nntp</td>
</tr>
<tr>
<td>139/tcp</td>
<td>open</td>
<td>netbios-ssn</td>
</tr>
<tr>
<td>143/tcp</td>
<td>open</td>
<td>imap2</td>
</tr>
<tr>
<td>513/tcp</td>
<td>open</td>
<td>login</td>
</tr>
<tr>
<td>514/tcp</td>
<td>open</td>
<td>shell</td>
</tr>
<tr>
<td>635/tcp</td>
<td>open</td>
<td>unknown</td>
</tr>
<tr>
<td>2049/tcp</td>
<td>open</td>
<td>nfs</td>
</tr>
</tbody>
</table>

Nmap run completed -- 1 IP address (1 host up) scanned in 51 seconds

Figure 6.2: This figure shows the results of an Nmap-scan against the victim host running RedHat Linux 5.2.
The TCP connect scan took 45 seconds to scan 65535 ports.
Interesting ports on (192.168.50.8):
(The 65529 ports scanned but not shown below are in state: closed)

<table>
<thead>
<tr>
<th>Port</th>
<th>State</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>21/tcp</td>
<td>open</td>
<td>ftp</td>
</tr>
<tr>
<td>70/tcp</td>
<td>open</td>
<td>gopher</td>
</tr>
<tr>
<td>80/tcp</td>
<td>open</td>
<td>http</td>
</tr>
<tr>
<td>135/tcp</td>
<td>open</td>
<td>loc-srv</td>
</tr>
<tr>
<td>139/tcp</td>
<td>open</td>
<td>netbios-ssn</td>
</tr>
<tr>
<td>1028/tcp</td>
<td>open</td>
<td>unknown</td>
</tr>
</tbody>
</table>

Nmap run completed -- 1 IP address (1 host up) scanned in 45 seconds

**Figure 6.3:** This figure shows the results of an Nmap-scan against the victim host running Windows NT 4.0 SP1.

### 6.1.3 The SIDS Configuration

As described in appendix D.3, we wrote a tool that automatically generates the mapping from Dragon attack names to CVE numbers by extracting the required information out of Dragon’s signature files. For our accuracy test we ran this tool on the most recent Dragon signature base. As shown in table 6.1, not all patterns stored in Dragon’s signature base have CVE numbers assigned to them.

<table>
<thead>
<tr>
<th>Number of Attack Patterns in Dragon’s Signature Base We Used 1172</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Attack Patterns Without CVE Number 922</td>
</tr>
<tr>
<td>Related to Attacks and Security Holes in the Conventional Sense 639</td>
</tr>
<tr>
<td>Related to Porn, Stools, Job-Search, Sports or Online-Gambling 283</td>
</tr>
<tr>
<td>Number of Attack Patterns Having a CVE Number Assigned 250</td>
</tr>
<tr>
<td>Related to Attacks and Security Holes in the Conventional Sense 250</td>
</tr>
<tr>
<td>Related to Porn, Stools, Job-Search, Sports or Online-Gambling 0</td>
</tr>
</tbody>
</table>

**Table 6.1:** Overview of the attack patterns in Dragon's signature base.

There is a substantial amount of attack patterns recognized by the Dragon IDS that cannot be processed by the add-on because they have no CVE number assigned. However — as can be seen from the table above — some patterns recognized by Dragon are related to detecting requests for porn-material, stock-trading, job-search, sports, or online-gambling activities. Such patterns do not have CVE numbers assigned to them because they do not deal with attacks or system vulnerabilities in the conventional sense and thus are of no interest to the CVE editorial board. Neither are these patterns of interest to our add-on. Consequently many of the patterns that do not have CVE number assigned to them also are of no security concern.

After having built the `dragon2cve` file — as described in appendix D.3 — we assigned rules to each of the resulting CVE numbers in a policy-file. In chapter 4 we explained that different types of attacks need different kinds of investigations. Accordingly we have defined five classes of attacks:
**nodelta:** Attacks of the class *nodelta* do not change the way an attacked system behaves from the viewpoint of our SIDS add-on. This implies that all available modules provide correct information even after occurrence of the attack. Therefore, we may use all available modules to take further investigations in the corresponding policy-file rules.

**dosext:** From the viewpoint of our add-on, attacks of the class *dosext* have DoS impacts only. But these attacks may have other impacts, too. In this case it is not enough to check whether the corresponding service is still running after the attack, we also have to check whether or not the host is vulnerable to the observed attack because the non-DoS part of the attack might have been successful while the DoS part of it was not. Thus in case of an attack of class *dosext* we may discard an alert only if the corresponding service is still running and the scanner tells us that it is not vulnerable, or we may also discard it if the database module tells us that the attacked host did not run such a service at the time the attack occurred.

**dos:** Attacks that only have denial of service impacts belong to the *dos* class. With attacks of this class we have to be careful using the scanner module, because it may indicate that a host is not vulnerable for two reasons. Either because the corresponding service on the attacked host really is not vulnerable. Or because the corresponding service has been vulnerable and therefore now no longer responds to requests which makes the scanner think that no vulnerable service is running on the attacked host. We may only discard an alert for an attack of this class if either the corresponding service is still running after the attack or the configuration database tells us that no such service had been running before the occurrence of the attack.

**any:** Attacks of the class *any* may completely change the way an attacked system behaves in every aspect. Therefore, the only appropriate source of information are the two database modules. Using a scanner or availability tester does not make sense in this case.

**unknown:** Attacks which do not fit into any of the above classes and for which we could not find a suitable customized rule we put into the class *unknown*.

The macros corresponding to these attack-types are shown in figure 6.4. Most of the CVE numbers in our policy-file belong to an attack or system vulnerability which falls into one of our five attack classes. Very few rules needed customized boolean expressions or bypass-flags set to true. Note that the rule for attacks of the type unknown just reads “0” such that all alerts concerning attacks of this type are always passed on. For this class of attacks we chose to use this rule instead of a bypass-flag, because we want to distinguish between attack-patterns that really need a bypass-flag because they are that important, and attack-patterns that we do not know how to treat suitably and therefore just pass them on using the rule unknown. Figure 6.5 shows parts of the policy-file we assembled and table 6.2 summarizes how many times each of the four macros was used.

As can be seen from the macros we defined for our policy-file, we assumed that the attacked host in our test is an “important host” in the sense described in section 3.4.3. This means that we use the static database instead of the dynamic database to make sure that there is absolutely no increase in the amount of false negatives.
Figure 6.4: These are the macros we defined in the policy-file used for the accuracy test. The names represent the classes of attacks the macros are used for.

2000-1094;1  
1999-0660;0;%unknown  
1999-0009;0;%amy  
1999-0060;0;%dos  
1999-0612;0;%nodelta  
2000-0413;0;%nodelta  
1999-0362;0;%dos  
2000-0131;0;%amy  
2000-0636;0;%dos  
2000-0244;1  
2000-0138;0;%unknown  
1999-0278;0;%nodelta  
1999-0349;0;%amy  
1999-0736;0;%nodelta  
2000-0017;0;%amy  
2000-0198;0;%dos  
1999-0472;0;%nodelta  
1999-0273;0;%dos  
1999-0073;0;%amy  
1999-0218;0;%dosext  
1999-0045;0;%nodelta  
1999-0067;0;%dosext

Figure 6.5: This figure shows parts of the policy-file we used for our accuracy tests. There are four predefined macros. Most of the rules make use of one of these macros. Only a few rules need customized expressions.

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodelta</td>
<td>50</td>
</tr>
<tr>
<td>dos</td>
<td>22</td>
</tr>
<tr>
<td>dosext</td>
<td>6</td>
</tr>
<tr>
<td>any</td>
<td>60</td>
</tr>
<tr>
<td>unknown</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 6.2: This table shows the distribution of the attack-types in the policy-file we assembled for our tests.
6.2 Results

6.2.1 Linux Victim

As can be seen from table 6.3, when observing the attacks on the victim host running Linux, the Dragon intrusion detection system generated a total of 93 alerts. Of these 93 alerts 56 were passed on directly because there was no CVE number associated with the attack name reported by Dragon, and 37 were processed by the smart IDS add-on. From the 37 alerts that the add-on further investigated, 13 were discarded because they were identified as false positives and 24 alerts had to be passed on. As the two pie-charts in figure 6.6 and 6.7 illustrate, the amount of alerts was substantially reduced by the add-on. Note that due to the conservative configuration of our add-on it is guaranteed that all discarded alerts are false positives for sure, which we also verified manually for this test.

<table>
<thead>
<tr>
<th>Total of Alerts Generated by Dragon IDS</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alerts Processed by SIDS Add-On</td>
<td>37</td>
</tr>
<tr>
<td>Discarded</td>
<td>13</td>
</tr>
<tr>
<td>Passed</td>
<td>24</td>
</tr>
<tr>
<td>Alerts not Processed by SIDS Add-On</td>
<td>56</td>
</tr>
<tr>
<td>No CVE</td>
<td>56</td>
</tr>
<tr>
<td>Unsupported Format</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 6.3: This table shows the results of the accuracy test with the Linux victim.

Table 6.4 shows how many attacks of each class defined above have occurred when using the scanner against the victim host running the old version of Linux.

<table>
<thead>
<tr>
<th>Attack-Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>%modelta</td>
<td>16</td>
</tr>
<tr>
<td>%dos</td>
<td>2</td>
</tr>
<tr>
<td>%doext</td>
<td>6</td>
</tr>
<tr>
<td>%any</td>
<td>11</td>
</tr>
<tr>
<td>%unknown</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.4: This table shows the distribution of the attack-types for the Linux victim.

Normally when a module is called because it is used in the rule of an attack that has been observed, the module will simply do its work and return its investigations to the SIDS-core. The scanner is a bit more complex than the other modules in that there are several reasons why it may not be possible for it to do its work. As shown table 6.5, the scanner-module in our test was called 22 times. There was one case where the scanner could not do its job because the attacked port differed from the port the corresponding plugin would have checked. There was a total of eight times when no plugin checking for the corresponding vulnerability was available. Three attacks would have triggered a plugin with destructive code, and
Figure 6.6: This chart shows how many of the alerts generated by the Dragon IDS were discarded by our smart IDS add-on and how many were passed.

Figure 6.7: This chart shows how many of the alerts processed by our smart IDS add-on were discarded and how many were passed.
finally, there were ten times when the scanner was able to investigate the success of the observed attack. For further information about the scanner module see section 5.5.3.

<table>
<thead>
<tr>
<th>Scanner Status</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>10</td>
</tr>
<tr>
<td>no plugin</td>
<td>8</td>
</tr>
<tr>
<td>acts destructive</td>
<td>3</td>
</tr>
<tr>
<td>wrong port</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.5: This table shows how the scanner module behaved for the test with the Linux victim.

### 6.2.2 NT Victim

This section describes how our smart IDS add-on performed when simulating an attack against the victim host running Windows NT. Similarly to the above case of a victim running Linux, table 6.6 shows that the Dragon intrusion detection system generated a total of 57 alerts. Of these 57 alerts 32 were passed on directly because there was no CVE number associated with the attack name reported by Dragon, and 25 were processed by the smart IDS add-on. From the 25 alerts that the add-on further investigated, 11 were discarded because they were identified as false positives and 14 alerts had to be passed on. As with the Linux victim, here too the amount of alerts has been substantially reduced by the add-on as shown by the two pie-charts in figures 6.8 and 6.9.

| Total of Alerts Generated by Dragon IDS | 57   |
| Alerts Processed by SIDS Add-On        | 25   |
| Discarded                               | 11   |
| Passed                                  | 14   |

| Alerts not Processed by SIDS Add-On | 32   |
| No CVE                                | 32   |
| Unsupported Format                    | 0    |

Table 6.6: This table shows the results of the accuracy test with the NT victim.

Table 6.7 shows how many attacks of each class defined above occurred when using the scanner against the victim host running the old version of Windows NT.

When running the attacks on the NT victim host, the scanner-module was called 17 times. There were ten times when the scanner further investigated the observed attack by running the corresponding plugin to check for vulnerabilities. Four times no suitable plugin was found and twice the corresponding plugin had destructive code and could therefore not be run. Finally, in one case the plugin that was related to the observed attack was not run because it would have checked the wrong port. These numbers are summarized in table 6.8.
**Figure 6.8:** This chart shows how many of the alerts generated by the Dragon IDS were discarded by our smart IDS add-on and how many were passed.

**Figure 6.9:** This chart shows how many of the alerts processed by our smart IDS add-on were discarded and how many were passed.
<table>
<thead>
<tr>
<th>Attack-Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>%nodelta</td>
<td>13</td>
</tr>
<tr>
<td>%dos</td>
<td>0</td>
</tr>
<tr>
<td>%doext</td>
<td>4</td>
</tr>
<tr>
<td>%any</td>
<td>7</td>
</tr>
<tr>
<td>%unknown</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 6.7:** This table shows the distribution of the attack-types for the NT victim.

<table>
<thead>
<tr>
<th>Scanner Status</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>10</td>
</tr>
<tr>
<td>no plugin</td>
<td>4</td>
</tr>
<tr>
<td>acts destructive</td>
<td>2</td>
</tr>
<tr>
<td>wrong port</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 6.8:** This table shows how the scanner module behaved for the test with the NT victim.

### 6.3 Summary

In this chapter we showed how we tested the accuracy of our smart IDS add-on. It was important to chose a fair set of attacks such that the results are representative for a real-world situation. That’s why we have decided to use a security scanner to launch attacks against a victim host instead of manually attacking it. As we have seen, our smart IDS add-on substantially decreases the number of false positives in our tests.
Chapter 7

Conclusions

Today, commercial network-based IDS generate alerts upon pattern-match on network traffic, although the observed attack may not be successful on the addressed host. This discrepancy can lead to false alerts. In order to reduce the number of such false alerts, we verify the success of an observed attack by taking further information about the attacked system into account. This information is gathered by carrying out so called investigations. However, some of them are not suited for all attacks. Therefore, we chose a modular approach for the implementation of the investigations. For each attack a so called rule controls which modules to apply and how to combine the gained information in order to decide whether or not the observed attack is a false positive.

To find the optimal rule for an attack, we classified the attacks and specified the optimal rule for each class. This greatly simplifies to find rules for new attacks because only the class of the attack has to be determined.

The modular approach also allows to easily add new aspects of verification by writing additional modules and incorporating them into the rules for the different classes of attacks.

As can be seen from the previous chapter, false positive alerts can be reduced substantially with our smart IDS add-on. Thus we successfully achieved the primary goal of our work. Furthermore, our add-on does not only detect potentially harmful attacks that have not been successful, it also protects companies providing managed intrusion detection products from being fooled by strategic enemies sending spoofed packets containing patterns that raise IDS alerts. This aspect is not reflected by our accuracy tests, because the security scanner we used to simulate an attacker does not spoof packets in such a manner. Nevertheless this is an important feature of our add-on because it is very easy to send masses of such spoofed packets to protected hosts and generate an arbitrary number of false alerts. Without our add-on the operator of an IDS is helpless because there is no other easy way to distinguish between such alerts and alerts stemming from real attacks.

The benefit from the database module could be further enhanced by adding more information to it. For example, in addition to a list of closed ports the database could also contain information about the operating system and the type and version of services running on the protected hosts. Together with the necessary information about the relation between CVE numbers and affected or vulnerable operating systems and services, even more false positive alerts could be discarded.

The value of the scanner module depends on the security scanner used within.
CHAPTER 7. Conclusions

The more attacks the scanner knows the more alerts it will be able to investigate. Further, the scanner must not use destructive code to check the existence of vulnerabilities and it must know the CVE numbers of the security holes it is able to check.

At the moment, the commercial intrusion detection products’ insufficient support of CVE numbers restricts the portion of alerts that can be handled by our add-on. This is because IDS-alerts without CVE numbers must be passed directly without further investigations regarding the success of the corresponding attack. But we think that in the near future the CVE naming scheme will be adopted by all product suppliers making the collaboration of a conventional IDS with our add-on even more powerful.
Appendices
Appendix A

Official Project Description

The following two pages are the official description of our diploma thesis project.
Smart Intrusion Detection

1 Introduction

Open Systems AG, based in Zurich and founded 7 years ago, is a company focusing on Internet Security. Besides security products and consulting services, Open Systems AG also offers managed firewalls, managed audits, and managed intrusion detection services. Consequently, Open Systems AG is very interested in new approaches to intrusion detection. Smart intrusion detection is one of them. Intrusion detection (ID) is the art of detecting inappropriate, incorrect, or anomalous activity on computer networks. Intrusion detection systems (IDS) which operate on a particular host to detect malicious activity on that host are called host-based IDS, and IDS that monitor and filter network data are called network-based IDS. Until today, very few large IDS have been implemented. Most of the time, one box is put to the outside of a firewall to protect a couple of important servers in a DMZ. They usually lack 24x7 monitoring and have the problematic property to generate a lot of false positive alerts. Too many alerts decrease the usability and the accuracy of an IDS. Consequently they are not taken seriously any more after a while since they are considered a waste of time. These false positives prevent IDS deployment to protect large scale networks. This flaw comes from the simplicity of today’s intrusion detection software. They are knowledge-based, very similar to anti-virus software. They usually read the network traffic and compare the flow of data to known attack patterns. Each time such a pattern is seen, an alert is generated. The problem is that today’s IDS are not sophisticated enough to really understand or at least provide the right information to manually verify if an attack was successful or not, or if there is suspicious follow-up activity showing that an attack has probably been successful.
2 Assignment

2.1 Objectives

The objective of this thesis is to extend the preliminary work done at Open Systems AG. The main problems are to determine what ‘intelligent’ features a smart intrusion detection system should have, to implement them, and to test them in a real life environment.

2.2 Tasks

- Get experience and become familiar with real life Intrusion Detection Systems. Recently published books discussing the subject are [Nor99, Amo99, MSK99, Ano99]. Get a feeling for how traditional IDS detect attacks and focus especially on the question why they report so many false alerts.

- Based on the experience acquired, develop a set of automated reactions that are able to decrease the number of false alerts. This can probably achieved by collecting additional information and by generating the right decisions out of them. Of course, the probability to detect an attack should not be reduced by the newly introduced measures.

- Develop, implement, and test a network-based smart intrusion detection system based on the theoretical results that were achieved during the previous phase. Compare its performance with existing IDS regarding the decrease in the number of false positives.

2.3 Deliverables

- At the end of the second week, a detailed time schedule of the diploma thesis must be given and discussed with the advisors.

- At half time of the diploma thesis, a short discussion of 15 minutes with the professor and the advisors will take place. The students have to talk about the major aspects of the ongoing work. At this point, the students should already have a preliminary version of the written report, including a table of contents. This preliminary version should be brought along to the short discussion.

- At the end of the diploma thesis, a presentation of 20 minutes must be given during the TIK or the communication systems group meeting. It should give an overview as well as the most important details of the work.

- The final report may be written in English or German. It must contain a summary written in both English and German, the assignment and the time schedule. Its structure should include an introduction, an analysis of related work, and a complete documentation of all used software tools. Three copies of the final report must be delivered to TIK.

References


Appendix B

Timeplan

The next page shows the timeplan we have chosen to achieve our goals.
## TIME PLAN

<table>
<thead>
<tr>
<th>Week</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2-5</td>
<td>Pre-study phase</td>
</tr>
<tr>
<td>6-7</td>
<td>Planning phase</td>
</tr>
<tr>
<td>8-11</td>
<td>Implementation phase</td>
</tr>
<tr>
<td>12-14</td>
<td>Testing phase</td>
</tr>
<tr>
<td>15-18</td>
<td>Report writing phase</td>
</tr>
</tbody>
</table>

### Introduction and Pre-Study Phase

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (44)</td>
<td>Ordinary administration tasks</td>
</tr>
<tr>
<td>1 (45)</td>
<td>Study the basics of Intrusion Detection Technology</td>
</tr>
<tr>
<td>1 (46)</td>
<td>Study in depth existing vulnerabilities and develop a classification thereof</td>
</tr>
<tr>
<td>2 (47-48)</td>
<td>Search for, study and test existing ID solutions and compare them</td>
</tr>
</tbody>
</table>

→ Presentation of the conclusions from the pre-study phase

### Planning Phase

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (49)</td>
<td>Think about useful smart extension to the existing ID solutions and plan their implementation</td>
</tr>
<tr>
<td>1 (50)</td>
<td>Choose and build up the programming environment to implement the smart extensions and learn the basic concepts needed</td>
</tr>
</tbody>
</table>

→ Presentation of the planned extension

### Implementation Phase

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 (1-4)</td>
<td>Implement the smart extensions</td>
</tr>
</tbody>
</table>

→ Presentation of the smart extension which have been implemented

### Testing Phase

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (5)</td>
<td>Define a test procedure and build up the testing environment</td>
</tr>
<tr>
<td>2 (6-7)</td>
<td>Perform the tests and analyze the results</td>
</tr>
</tbody>
</table>

### Report Writing Phase

<table>
<thead>
<tr>
<th>Weeks</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 (8-10)</td>
<td>Write the report</td>
</tr>
<tr>
<td>1 (11)</td>
<td>Prepare the presentation</td>
</tr>
</tbody>
</table>

→ Final presentation of the diplomathesis at ETH (week 11)
Appendix C

Important Tools and Manuals

C.1 Tcpdump

The command line tool tcpdump [28] allows network traffic capture. It is based on the libpcap [27] packet capture library available for many operating systems. The captured traffic is stored in a format readable by many other tools, in particular by tcpreplay (see appendix C.2) and ethereal (see appendix C.4).

The libpcap library implements a filter mechanism that can be controlled from the tcpdump command line. Therefore it is possible to restrict capturing to packets that match some filter rule. Among others, libpcap can filter based on the IP addresses, the protocol type, and the TCP/UDP ports.

We used tcpdump to capture network traffic in many situations. For example we observed the local response packets generated by the IDS products and discovered the special payload of RealSecure reset-packets.
NAME
tcpdump — dump traffic on a network

SYNOPSIS
tcpdump [-alnqqRSTvxX] [-i interface] [-m mode] [-n snaplen] [-p expression]
- SHOWS the headers of packets on a network interface that match the boolean expression.

DESCRIPTION
Tcpdump prints out the headers of packets on a network interface that match the boolean expression.

OPTIONS

-a Show captured data in ASCII.

-b Only capture packets using specified data-link layer protocol. Possible protocols include: ip, ipv6, 802.2, 802.3, arp, rarp, dec, link, atalk, aarp, x25 and bsd. Protocol can also be specified as a decimal value (RFC 1340).

-c Exit after receiving count packets.

-d Dump the compiled packet-matching code in a human readable form to standard output and stop.

-dd Dump packet-matching code as decimal numbers (preceded with a count).

-e Print the link-level header on each dump line.

-f Print foreign internet addresses numerically rather than symbolically (this option is intended to get around serious brain damage in Sun's svr add — usually it hangs forever translating non-local internet numbers).

-F Use file as input for the filter expression. An additional expression given on the command line is ignored.

-i Listen on interface. If unspecified, tcpdump searches the system interface list for the lowest numbered, configured up interface (excluding loopback). Ties are broken by choosing the earliest match.

-l Make output line buffered. Useful if you want to see the data while capturing it. E.g., ‘tcpdump -l’ to be ‘tcpdump -i’ etc.

-m Don’t convert host addresses to names. This eliminates the need for DNS lookups.

-n Don’t convert port numbers to service names either. Ipv4 addressing is used for port number conversion, so this doesn’t need to be defined for ‘passive’ operation.

-R Don’t print domain name qualification of host names. E.g., if you give this flag then tcpdump will print ‘inc.’ instead of ‘inc.domain’.

-S Don’t run the packet-matching code optimizer. This is useful if you suspect a bug in the optimizer.

-t Do not put the interface into promiscuous mode. Note that legacy mode was to use promiscuous mode by default. Also note that the interface might be in promiscuous mode for some other reason.

-x Quick (output) output. Print less protocol information so output lines are shorter.

-R Use RAW socket interface. If you want to use multiple interfaces simultaneously, packet socket mode (the default) is required.

-x Smart snaplen bytes of data from each packet rather than the default of 68. With SunOS’s NIT, the minimum is actually 96. 68 bytes is adequate for IP, ICMP, TCP and UDP but may truncate protocol information from name server and NFS packets (see below). Packets truncated because of a limited snapshot are indicated in the output with ‘[Ignored]’, where proto is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit snapshots to the smallest number that will capture the protocol information you’re interested in.

-T For packets selected by ‘expression’, tcpdump prints the specified type. Currently known types are ‘type’ (Remote Procedure Call), ‘rtp’ (Real-Time Applications protocol), ‘step’ (Real-Time Applications control protocol), ‘snmp’ (Simple Network Management Protocol), ‘vat’ (Visual Audio Tool), ‘<name or number>’ (RFC 1340).

-S Print binary, rather than relative, TCP sequence numbers.

-d Don’t print a timestamp on each dump line.

-g Print an unformatted timestamp on each dump line.

-h (Slightly more) verbose output. For example, the time to live and type of service information in an IP packet is printed.

-ww Even more verbose output. For example, additional fields are printed from NFS reply packets.

-w Write the raw packets to file rather than parsing and printing them out. They can later be printed with the −v option. Standard output is used if file is ‘−’.

-x Print each packet (minus its link level header) in hex. The smaller of the entire packet or snaplen bytes will be printed.

-X Use packet socket interface. This is the default.

expression selects which packets will be dumped. If no expression is given, all packets on the net will be dumped. Otherwise, only packets for which expression is true will be dumped.

The expression consists of one or more predicates. Predicates usually consist of an id (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

type qualifiers say what kind of thing the id name or number refers to. Possible types are host, net and port. E.g., ‘host foo’, ‘net 128.3’, ‘port 20’. If there is no type qualifier, all protocols consistent with the type are assumed. E.g., ‘host foo’ means ‘(ip or rarp or rarp or ... that legacy mode was to use promiscuous mode by default. Also note that the interface might be in promiscuous mode for some other reason.

-q Quick (output) output. Print less protocol information so output lines are shorter.

-R Use RAW socket interface. If you want to use multiple interfaces simultaneously, packet socket mode (the default) is required.

-x Smart snaplen bytes of data from each packet rather than the default of 68. With SunOS’s NIT, the minimum is actually 96. 68 bytes is adequate for IP, ICMP, TCP and UDP but may truncate protocol information from name server and NFS packets (see below). Packets truncated because of a limited snapshot are indicated in the output with ‘[Ignored]’, where proto is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit snapshots to the smallest number that will capture the protocol information you’re interested in.

-T For packets selected by ‘expression’, tcpdump prints the specified type. Currently known types are ‘type’ (Remote Procedure Call), ‘rtp’ (Real-Time Applications protocol), ‘step’ (Real-Time Applications control protocol), ‘snmp’ (Simple Network Management Protocol), ‘vat’ (Visual Audio Tool), ‘<name or number>’ (RFC 1340).

-S Print binary, rather than relative, TCP sequence numbers.

-d Don’t print a timestamp on each dump line.

-g Print an unformatted timestamp on each dump line.

-h (Slightly more) verbose output. For example, the time to live and type of service information in an IP packet is printed.

-ww Even more verbose output. For example, additional fields are printed from NFS reply packets.

-w Write the raw packets to file rather than parsing and printing them out. They can later be printed with the −v option. Standard output is used if file is ‘−’.

-x Print each packet (minus its link level header) in hex. The smaller of the entire packet or snaplen bytes will be printed.

-X Use packet socket interface. This is the default.

expression selects which packets will be dumped. If no expression is given, all packets on the net will be dumped. Otherwise, only packets for which expression is true will be dumped.

The expression consists of one or more predicates. Predicates usually consist of an id (name or number) preceded by one or more qualifiers. There are three different kinds of qualifier:

-type qualifiers say what kind of thing the id name or number refers to. Possible types are host, net and port. E.g., ‘host foo’, ‘net 128.3’, ‘port 20’. If there is no type qualifier, all protocols consistent with the type are assumed. E.g., ‘host foo’ means ‘(ip or rarp or rarp or ... that legacy mode was to use promiscuous mode by default. Also note that the interface might be in promiscuous mode for some other reason.

-q Quick (output) output. Print less protocol information so output lines are shorter.
In addition to the above, there are some special "primitive" keywords that don't follow the pattern gateway, broadcast, less, greater, or arithmetic expressions. They are described below.

More complex filter expressions are built up by using the words and, or, and not to combine primitives. E.g., "host foo and not port 80 and not port 3306". To save typing identical qualifier lists can be omitted. E.g., "tcp dst port ftp or udp port domain" is exactly the same as "tcp dst port ftp or udp port data or tcp dst port domain".

Allowable primitives are:

dst host <name>

True if the destination address of the packet is <name>, which may be either an address or a name.

src host <name>

True if the source address of the packet is <name>.

host <name>

True if either the source or destination address of the packet is <name>. Any of the above host expressions can be prepended with the keywords, ip, arp, or rarp

ether host <name>

which is equivalent to:

ether proto ip and host <name>

If <name> is a name with multiple IP addresses, each address will be checked for a match.

ether dst ehost

True if the ethernet destination address is ehost. Ehost may be either a name from ethers or a number (see ethers(3N) for numeric format).

ether src ehost

True if the ethernet source address is ehost.

ether host ehost

True if either the ethernet source or destination address is ehost.

gateway host

True if the packet used host as a gateway. I.e., the ethernet source or destination address was host but neither the IP source nor the IP destination was host. Host must be a name and must be found in both ethers(3N) and /etc/hosts. (An equivalent expression is ether host <name> and not host <name> which can be used with either names or numbers for host (ehost)).

dst net <net>

True if the IP destination address of the packet has a network number of <net>. Not may be either a name from networks or a network number (see networks(4) for details).

src net <net>

True if the IP source address of the packet has a network number of <net>.

net <mask>

True if the IP source or destination address of the packet has a network number of <net>.

net or mask match <mask>

True if the IP address matches <net> with the specific mask. May be qualified with src or dst.

net not <mask>

True if the IP address matches <net> but not with the specific mask. May be qualified with src or dst.

dst port <port>

True if the packet has a destination port value of <port>. The port can be a number or a name used in /etc/services (see tcp(4P) and udp(4P)). If a name is used, both the port number and protocol are checked. If a number or ambiguous name is used, only the port number is checked (e.g., dst port FTP will print both tcp/login traffic and udp/who traffic, and port domain will print both tcp/domain and udp/domain traffic).

src port <port>

True if the packet has a source port value of <port>.

port <port>

True if either the source or destination port of the packet is <port>. Any of the above port expressions can be prepended with the keywords, tcp or udp as in:

tcp src port <port>

which matches only tcp packets whose source port is <port>.

less <length>

True if the packet has a length less than or equal to <length>. This is equivalent to:

len <= <length>

greater <length>

True if the packet has a length greater than or equal to <length>. This is equivalent to:

len >= <length>

ip proto <protocol>

True if the packet is an ip packet (see ip(4P)) of protocol type <protocol>

ip <protocol> protocol

which assumes, when filtering on the protocol identifier, that all FDDI packets include an LLC header, and that the LLC header is in so-called SNAP format.

ether proto <protocol>

True if the packet is an ethernet packet of type <protocol>. Protocol can be a number or one of the names decnet, arp, rarp, or tcp. Note that the identifiers decnet, rarp, and tcp are also keywords and must be escaped via backslashes (\), which is \ in the C-shell.

ether broadcast

True if the packet is an ethernet broadcast packet. The ether keyword is optional.

ip broadcast

True if the packet is an IP broadcast packet. It checks for both the all-addresses and all-sources broadcast conventions, and looks up the local subnet mask.

ether multicast

True if the packet is an ethernet multicast packet. The ether keyword is optional. This is shorthand for ether(4P) & 1 != 0.

ip multicast

True if the packet is an IP multicast packet.

ether proto <protocol>

True if the packet is of other type <protocol>. Protocol can be a number or a name like ip, arp, or rarp. Note these identifiers are also keywords and must be escaped via backslashes (\). In the case of FDDI (e.g., "fddi proto protocol"), the protocol identification comes from the 80.2 Logical Link Control (LLC) header, which is usually layered on top of the FDDI header. Tcpdump assumes, when filtering on the protocol identifier, that all FDDI packets include an LLC header, and that the LLC header is in so-called SNAP format.

decnet src <name>

True if the DECNET source address is <name>, which may be an address of the form "10.123.45.67", or a DECNET host name. [DECNET host-name support is only available on Unix systems that are configured to run DECNET]
Expression arguments can be passed to `tcpdump` as either a single argument or as multiple arguments, whichever is more convenient. Generally, if the expression contains Shell metacharacters, it is easier to pass it as a single, quoted argument. Multiple arguments are concatenated with spaces before being parsed.

**Examples**

To print all packets arriving at or departing from `sundown`:

```
tcpdump host sundown
```

To print all IP packets between `helios` and `hot` or `ace`:

```
tcpdump ip host helios and (hot or ace)
```

To print all UDP packets between any host except `helios`:

```
tcpdump udp host helios and not helios
```

To print all IP packets longer than 576 bytes sent through gateway `snup` and (port ftp or ftp-data):

```
tcpdump ip and not net localnet
```

To print packets longer than 576 bytes with options:

```
tcpdump ip[6:2] & 0x1fff = 0
```

To print IP broadcast or multicast packets that were sent via ethernet broadcast or multicast:

```
tcpdump ether[0] & 1 != 0 or ether[0] & 2 != 0
```

To print all ICMP packets with options:

```
tcpdump icmp[0] != 8 and icmp[0] != 0
```

**Output Format**

The output of `tcpdump` is protocol-dependent. The following gives a brief description and examples of most of the formats.

### Link Level Headers

If the `-e` option is given, the link level header is printed out. On ethernet, the source and destination addresses, protocol, and packet length are printed.

On FDDI networks, the `-e` option causes `tcpdump` to print the `Frame Control` field, source and destination addresses, and the packet length. The `Frame Control` field governs the interpretation of the rest of the packet. Normal packets (such as those containing IP datagrams) are `async` packets, with a priority value between 0 and 7; for example, `async-2`. Such packets are assumed to contain an 802.2 Logical Link Control (LLC) packet; the LLC header is printed if it is an ISO datagram or an so-called SNAP packet.

(N.B: The following description assumes familiarity with the SDEP compression algorithm described in RFC-1144.)

On SLIP links, a direction indicator (`-I` for inbound, `-O` for outbound), packet type, and compression information are printed out. The packet type is printed first. The three types are ip, udp, and tcp. No further link information is printed for ip packets. For TCP packets, the connection identifier is printed following the type. If the packet is compressed, its encoded header is printed out. The special cases are printed out as `0x00` and `0x80`, where `x` is the amount by which the sequence number or sequence number and ack has changed. If it is not a special case, zero or more changes are printed. A change is indicated by U
ARP/RARP Packets

Arp/Rarp output shows the type of request and its arguments. The format is intended to be self-explanatory. The first line says that the source host (host who) requests an address for the host (host to) host who has the name csam. The arp who-has command is a request to find the MAC address associated with the specified IP address. The arp who-has command is a request to find the MAC address associated with the specified IP address.

TCP Packets

The general format of a TCP protocol line is:

```
src > dst: sport dport srcsport dstsport proto [flags] srcip dstip [srcport] [dport] [sport] [dport] [proto] [flags]
```

Here is the opening portion of an rlogin from host actinide to host csam:

```
actinide.who > csam.login: S 768512:768512(0) win 4096 <mss 1024>
```

UDP Packets

The general format of a UDP protocol line is:

```
src > dst: sport dport [proto] [flags] srcip dstip [sport] [dport] [proto] [flags] [srcport] [dport] [proto] [flags]
```

Here is the opening portion of an rlogin from host actinide to host csam:

```
actinide.who > csam.login: S 768512:768512(0) win 4096 <mss 1024>
```

Some UDP services are recognized (from the source or destination port number) and the higher level protocol information is printed. In particular, Domain Name service requests (RFC-1034/1035) and Sun RPC calls (RFC-1050) to NFS.
between the '3' and the '6'. Similarly, the qclass was the normal one, C_IN, and omitted. Any other qclass would have been printed immediately after the '3'.

A few numbers are checked and may result in extra fields enclosed in square brackets: If query contains an answer, name server or authority section, answer, reverse, or answer are printed as [name], [type], or [class], where s is the appropriate count. If any of these response bits are set (A, R, A or R only) or any of the 'must be set' bits are set in bytes two and three, the [name] is printed, where s is the hex value of header bytes two and three.

**UDP Name Server Responses**

Name server responses are formatted as:

```
src > dst: id op rcode
flags a/n/au type class data (len)
```

For example:

```
sushi.6709 > wrl.nfs: 112 readlink fh 21,24/10.73165
```

The first two lines give the names of appletalk networks. The third line gives the name of a particular host (note that the number following the src host is a transaction id, not the source port). The request was 112 bytes, excluding the UDP and IP headers. The operation was a readlink (read symbolic link) on the file handle fh 21,24/10.73165. (If one is lucky, as in this case, the file handle can be interpreted as a major,minor device number pair, followed by the inode number and generation number.) If the response is successful, it contains an entry for the file handle, the data printed on the operation type. The format is intended to be self explanatory if read in conjunction with an NFS protocol spec.

If the `-v` (verbose) flag is given, additional information is printed. For example:

```
sushi.1372a > wrl.nfs: 348 read fh 21,11/12.156 8392 bytes 0 24576
```

This also prints the IP header T TL, ID, and fragment size fields, which have been omitted from this example.

The first line, `sushi...` tells the reader that 8192 bytes from file 21,11/12.156, at byte offset 24576. The reply itself begins with the content of the first fragment of the reply, and hence is only 1472 bytes long (the other bytes will follow in subsequent fragments, but these fragments do not have NFS or even UDP headers and so might not be printed, depending on the filter expression used). Because the `-v` flag is given, some of the file attributes (which are returned in addition to the file data) are printed; the file type `REG` for regular file, the file mode (in octal), the uid and gid, and the file size.

If the `-v` flag is given more than once, even more details are printed. Note that NFS requests are very large and much of the detail won't be printed unless `snaplen` is increased. Try using `-P 64` to watch NFS traffic.

**KIP Appletalk (DID in UDP)**

Appletalk DID packets encapsulated in UDP datagrams are de-encapsulated and dumped as DIDP packets (all the UDP header information is discarded). The `file décidno/name` is used to translate appletalk net and node numbers to names. Lines in this file have the form

```
number name
1 254 ether
16 1 icd-net
1 254 110
```

The first two lines give the names of appletalk networks. The third line gives the name of a particular host (a host is distinguished from a net by the Y id in the number—a net number may have two octets and a host number must have three octets.) The number and name should be separated by whitespace (blanks or tabs). The `file décidno/name` may contain blank lines or comment lines (lines starting with a `#`). Appletalk addresses are printed as the form

```
net/port
```

The `#port` column contains the address of the appletalk host. The `#port` column contains the address of the appletalk host. This line is a send from port 235 on net jssmag node 149 to broadcast on the icsd-net NBP port (note that the broadcast address (255) is indicated by a net name with no host number—for this reason it's a good idea to keep node names and net names distinct in `atalk.names`).

NFS name binding protocol and ATP (Appletalk transaction protocol) packets have their contents interpreted. Other protocols just dump the protocol name (or number if no name is registered for the protocol).
NBP packets are formatted like the following examples:

```
  NET/NET:1.2.3.4 > NET/NET:5.6.7.8
```

The first line is a name lookup request for laserwriters sent by net icsd host 112 and broadcast on net jssmag. The reply for the lookup is 190. The second line shows a reply for this request (note that it has the same id) from host jssmag 209 saying that it has a laserwriter resource named "RM1140" registered on port 230. The third line is another reply to the same request saying host techpit has laserwriter "techpit" registered on port 186.

**ATP packet formatting is demonstrated by the following example:**

```
jssmag.209.133 < helios.132.165,1: atp-req 12267 0xae030002
  helios.132 > jssmag.209.165: atp-resp 12266:0 (512) 0xae040000
  jssmag.209.133 > helios.132: atp-rel 12266<0-7> 0xae030001
```

Jssmag 209 initiates transaction id 12266 with host helios by requesting up to 8 packets (the <0-7>). The box number of the end of the line is the value of the 'sequence' field in the request. Helios responds with 8 512-byte packets. The 'nf' following the transaction id gives the packet sequence number in the transaction and the number in parentheses is the amount of data in the packet, excluding the atp header. The * on packet 7 indicates that the EOM bit was set.

**Name server inverse queries are not dumped correctly:** The (empty) question section is printed rather than the real query in the answer section. Some believe that inverse queries are themselves a bug and prefer to reassemble IP fragments or, at least to compute the right length for the higher level protocol.

**Apple Ethertalk DDP packets could be dumped as easily as KIP DDP packets but aren't. Even if we were inclined to do anything to promote the use of Ethertalk (we aren't), LBL doesn't allow Ethertalk on any of its networks so we would have no way of testing this code.**

**IP Fragmentation:**

```
  Id is the fragment id. S= Size is the fragment size (in bytes) excluding the IP header. Offset is this fragment's offset (in bytes) in the original datagram.

  The fragment information is output for each fragment. The first fragment contains the higher level protocol header and the frag info is printed after the protocol info. Fragments after the first contain no higher level protocol header and the frag info is printed after the source and destination addresses. For example, here is part of an ftp from arizona.edu to lbl-rtsg.arpa over a CSNET connection that doesn't appear to handle 576-byte datagrams:
```

```
arizona.ftp-data > rtsg.1170: . 1024:1332(308) ack 1 win 4096 (frag 595a:328@
  arizona > rtsg: (frag 595a:204@328)rtsg.1170 > arizona.ftp-data: . ack 1536 win 2560
  250techpit.2 > icsd-net.112.220: nbp-reply 190: "techpit:LaserWriter@*"
```

**There are a couple of things to note here:** First, addresses in the second line don't include port numbers. This is because the TCP protocol information is all in the first fragment and we have no idea what the port or sequence numbers are when we print the later fragments. Second, the tcp sequence information in the first line is printed as if there were 308 bytes of user data when, in fact, there are 512 bytes (308 in the first frag and 204 in the second). If you are looking for holes in the sequence space or trying to match up acks with packets, this can fool you.

A packet with the IP 'don't fragment' flag is marked with a trailing (DF) after the fragment size in the output. Filters expressions that manipulate FDDI headers assume that all FDDI packets are encapsulated Ethernet packets. This is true for IP, ARP, and DECNET Phase IV, but is not true for protocols such as ISO CLNS. Therefore, the filter may inadvertently accept certain packets that do not properly match the filter expression.
C.2 Tcpreplay

The purpose of tcpreplay [31] is to replay previously captured TCP traffic. This may be used for reproducible tests. Tcpreplay also allows to change the speed at which the packets are sent out.

Tcpreplay is part of the nidsbench [30] network-based intrusion detection test suite.
NAME
tcpreplay – replay traffic from a saved tcpdump file

SYNOPSIS
$tcpreplay [-i intf] [-l loop count] [-r rate] [-m multiplier] file ...

DESCRIPTION
Tcpreplay is a program for replaying network traffic saved in packet-trace files generated using tcpdump(8)’s −w flag.

The basic operation of tcpreplay is to resend all packets from its input file(s) at the speed at which they were recorded, at some specified data rate, or as fast as the hardware is capable of. If no rate or multiplier are given, tcpreplay will replay packets as fast as the hardware will allow. If no file is given, tcpreplay will accept packet data from stdin.

OPTIONS
−i Specify the interface to send packets out on.
−r Resend the packets at the rate specified (in Mbps).
−m Resend the packets at a multiple of the speed at which they were recorded.
−l Resend the pcap file(s) loop count times.

SEE ALSO
tcpdump(8)

AUTHOR
Matt Undy, Anzen Computing.
The current version is available via HTTP:
http://www.anzen.com/research/nidsbench/

BUGS
tcpreplay can only send packets as fast as your machine’s interface, processor, and disk will allow.
“N write attempts failed from full buffers and were repeated” does not indicate that these packets were not sent, but that the send was retired N times until it succeeded.
Looping captured traffic may simulate odd conditions on a network. For example, opening the same TCP connection multiple times may exhaust resources on machines tracking the connection. The −l flag is provided to allow faster sending on machines with greater I/O constraints.
On BSD-based systems, kernel modifications are required to preserve outgoing link layer source addresses.
Refer to the libnet(3) documentation for more information on how to do this.
Please send bug reports to nidsbench@anzen.com.
C.3 Fragrouter

Fragrouter is an IP router that has the capability of modifying packets, trying to perform insertion or evasion attacks on network-based IDS. The main categories of these modifications include IP fragmentation (one incoming IP packet is sent out as many IP fragments with varying order and size), TCP segmentation after a completed TCP handshake, and others exploiting specific TCP/IP stack behaviour.

We used fragrouter to verify the IDS products behaviour when it comes to reassembling packets (IP defragmentation and TCP stream reassembling).

Fragrouter is part of the nidsbench [30] network-based intrusion detection test suite.
NAME

fragrouter – network intrusion detection evasion toolkit

SYNOPSIS

fragrouter [-4 interface] [-p] [-g hop] [-G hopcount] ATTACK

DESCRIPTION

Fragrouter is a program for routing network traffic in such a way as to elude most network intrusion detection
systems.


OPTIONS

-4 Specify the interface to accept packets on.

-p Preserve the entire protocol header in the first fragment. This is useful in bypassing packet filters that drop short IP fragments.

-g Specify a hop along a loose source routed path. Can be used more than once to build a chain of hop points.

-G Positions the “hop counter” within the list of hosts in the path of a source routed packet. Would be a multiple of 4. Can be set past the length of the loose source routed path to implement Anthony Osborne’s Windows IP source routing attack of September 1999.

The following attack options are mutually exclusive - you may only specify one type of attack to run at a
time.

-B1 baseline-1: Normal IP forwarding.

-F1 frag-1: Send data in ordered 8-byte IP fragments.

-F2 frag-2: Send data in ordered 1-byte IP fragments.

-F3 frag-3: Send data in ordered 8-byte IP fragments, with one fragment sent out of order.

-F4 frag-4: Send data in ordered 8-byte IP fragments, duplicating the penultimate fragment in each packet.

-F5 frag-5: Send data in out of order 8-byte IP fragments, duplicating the penultimate fragment in each packet.

-F6 frag-6: Send data in ordered 8-byte IP fragments, sending the marked last fragment first.

-F7 frag-7: Send data in ordered 8-byte IP fragments, with one fragment sent out of order.

-scapt-1: Complete TCP handshake, send fake FIN and RST (with bad checksums) before sending data in ordered 1-byte segments.

-scapt-2: Complete TCP handshake, send data in ordered 1-byte segments interleaved with SYN packets for the same connection parameters.

-scapt-3: Do not complete TCP handshake, but send null data in ordered 1-byte segments as if one had occurred. Then, complete a TCP handshake with same connection parameters, and send the real data in ordered 1-byte segments.

-scapt-4: Complete TCP handshake, shut connection down with a RST, re-connect with drastically different sequence numbers and send data in ordered 1-byte segments.

-scapt-5: Complete TCP handshake, send data in ordered 1-byte segments but with no ACK flag set.

-scapt-6: Complete TCP handshake, send data in ordered 1-byte segments but with SYN-ACK flag set.

-scapt-7: Complete TCP handshake, send data in out of order 1-byte segments.

-scapt-8: Complete TCP handshake, send data in ordered 1-byte segments with one segment sent out of order.

-see ALSO

tcpdump(8), tcpreplay(8), pcap(3), libnet(3)

AUTHOR

Dug Song, Anzen Computing.

The current version is available via HTTP:

http://www.anzen.com/research/nidsbench/

BUGS

IP options will carry across all fragments of a packet. Fragrouter is not smart enough to determine which IP
options are valid only in the first fragment. This is considered a feature, not a bug. :-)

Similarly, TCP options will carry across all segments of a split TCP packet - except for null data packets
preceding a forward overwrite, which lack any TCP options in order to elude TCP PAWS elimination.

Please send bug reports to nidsbench@anzen.com.
C.4 Ethereal

The GUI tool *ethtool* [29] reads tcpdump files and displays the packets in a human-readable way. It also analyzes various protocols and shows their header data, as can be seen from the screenshot in figure C.1.

Ethereal is not only capable of displaying but also of recording dump network data. However, we mainly used ethereal to view previously recorded dump files for various purposes.

![Ethereal Screen Shot](image)

**Figure C.1:** Ethereal displays protocol information in a human-readable format. This is a trace of a fragmented ping request, generated by fragrouter. Note the out-of-order sequence of the fragments.

C.5 Nemesis

Based on the *libnet* library [33], *nemesis* allows the creation of customized ARP, DNS, ICMP, IGMP, OSPF, RIP, UDP, and TCP packets. This is very useful for example when it comes to test the network-based sensors.

For each of the supported protocols there is a separate command line tool called *nemesis-*<protocol> with corresponding manual page. Since we mainly generated customized TCP packets, here is the manual page for *nemesis-*tcp.
NAME
nemesis-tcp — TCP Protocol (The Nemesis Project)

SYNOPSIS
tcp [-v] [optlist]

DESCRIPTION
The Nemesis Project is designed to be a command-line-based, portable human IP stack for UNIX/Linux. The
suite is broken down by protocol, and should allow for useful scripting of injected packet streams from
simple shell scripts.

TCP Options
[-s Source Port] Source Port of injected packet.
[-d Destination Port] Target Port of injected packet.
[-x TCP Flag Options (-xS/-xA/-xR/-xP/-xF/-xU)]
SYN, ACK, RST, PSH, FIN, URG
[-w Window Size] TCP Window Size.
[-s Sequence Number] TCP Sequence Number.
[-a Acknowledgement Number] TCP Acknowledgment Number.
[-t TCP Urgent Pointer] TCP Urgent Pointer.
[-P Payload File] Filename to read for packet payload. Use with -b for binary
packet payloads (unprintable ASCII).
[-v Verbose Mode] Display human readable output of currently injected packet.

IP Options
[-S Source IP Address] Source Address of injected packet.
[-D Destination IP Address] Target Address of injected packet.
[-I IP ID] IP ID HEADER tag.
[-T IP TTL] IP Time To Live field.
[-s IP flow] IP TypeOf Service field.
[-F IP frag] IP Fragmentation field.
[-O IP Options] IP Options field.

Data Link Options
[-d Ethernet DeviceName of ethernet device (eg. ne0, ed0, eth0)].
[-H Source MAC Address] Source MAC Address (XX:XX:XX:XX:XX:XX)

-M Destination MAC Address
Target MAC Address (XX:XX:XX:XX:XX:XX)

AUTHOR
obecian <obecian@celerity.bartoli.org>

SEE ALSO
nemesis-arp(1), nemesis-dns(1), nemesis-icmp(1), nemesis-igmp(1), nemesis-ospf(1), nemesis-rip(1), nemesis-udp(1)

BUGS
None known
C.6 Nessus

*Nessus* [26] is a network-based audit tool. Our smart IDS uses Nessus for the triggered scanner extension. Nessus is described in 5.5.3

C.7 Nmap

The probably most famous port-scanner tool is called *nmap* [35]. It features several port-scanning methods and TCP/IP stack fingerprinting.
NAME
nmap — Network exploration tool and security scanner

SYNOPSIS
nmap [Scan Type(s)] [Options] <host or net #1 ... #N>

DESCRIPTION
nmap is designed to allow system administrators and curious individuals to scan large networks to determine which hosts are up and what services they are offering. nmap supports a large number of scanning techniques such as: UDP, TCP connect(), TCP SYN (half open), TCP SYN/ACK, RST, passive (ICMP ping sweep), FIN, ACK, Xmas Tree, SYN sweep, and Null scan. See the Scan Types section for more details. nmap also offers a large number of unique features such as remote OS detection via TCP/IP fingerprinting, stealth scanning, port delay and return time calculations, parallel scanning, detection of down hosts via parallel pings, decay scanning, port filtering detection, direct (non-rootmapper) RPC filtering detection, direct (non-portmapper) RPC scanning, fragmentation scanning, and flexible target and port specification.

Significant effort has been put into decent nmap performance for non-root users. Unfortunately, many critical kernel interfaces (such as raw sockets) require root privileges. nmap should be run as root whenever possible.

The result of running nmap is usually a list of interesting ports on the machine(s) being scanned (if any).

nmap always gives the port's "well known" service name (if any), number, state, and protocol. The state is either 'open', 'filtered', or 'unfiltered'. Open means that the target machine will accept() connections on that port. Filtered means that a firewall, filter, or other network obstacle is covering the port and preventing nmap from determining whether the port is open. Unfiltered means that the port is known by nmap to be closed and no firewall/filter seems to be interfering with nmap's attempts to determine this. Unfiltered ports are the common case and are only shown when most of the scanned ports are in the filtered state.

Depending on options used, nmap may also report the following characteristics of the remote host OS in use: TCP sequenceability, timestamping the programs which have bound to each port, the DNS name, whether the host is a named address, and a few others.

OPTIONS

Options that make sense together can generally be combined. Some options are specific to certain scan modes. nmap tries to catch and warn you about obvious or unsupported option combinations.

If you are impatient, you can skip to the examples section at the end, which demonstrates common usage. You can also run nmap -h for a quick reference page listing all the options.

SCAN TYPES

-sT TCP connect() scan. This is the most basic form of TCP scanning. The connect() system call provided by your operating system is used to open a connection to every interesting port on the machine. If the port is listening, connect() will succeed, otherwise the port isn't reachable. One strong advantage to this technique is that you don't need any special privileges. Any user on most UNIX boxes is free to use this call.

This sort of scan is easily detectable as target host logs will show a bunch of connection and error messages for the services which accept() this connection just to have it immediately shutdown.

-sS TCP SYN scan. This technique is often referred to as "half-open" scanning, because you don't open a full TCP connection. You send a SYN packet, and if you are going to open a full connection and you wait for a response. A SYNACK indicates the port is listening. A RST is indicative of a non-listener. If a SYNACK is received, a RST is immediately sent to tear down the connection (actually our OS kernel does this for us). The primary advantage to this scanning technique is that fewer hosts will log it. Unfortunately you need root privileges to build these custom SYN packets.

-sF -sX -sN SYNFIN, Xmas Tree, or Null scan mode. There are times when even SYN scanning isn't clandestine enough. Some firewalls and packet filters watch for SYN to restricted ports, and programs like SynStalker and Courtsey are available to detect these scans. These advanced scans, on the other hand, may be able to pass through unscathed.

The idea is that closed ports are required to reply to your probe packet with an RST, while open ports must ignore the standard and do things their own way. Thus this scan type will not work against systems running Windows 95/NT. On the positive side, this is a good way to distinguish between the two platforms. If the scan finds open ports, you know the machine is not a Windows box. If a 'TCP-SYN+ACK->SYN' scan shows all ports closed, yet a SYN (-sS) scan shows ports being opened, you are probably looking at a Windows box. This is less useful now that nmap has proper OS detection built in. There are also a few other systems that are broken in the same way Windows is. They include Cisco, BSDI, HP-UX, MVS, and IRIX. All of the above send resets from the open ports when they should just drop the packet.

-sP Ping scanning: Sometimes you only want to know which hosts on a network are up. Nmap can do this by sending ICMP ping packets to every IP address on the networks you specify. Hosts that respond are up. If the Nagios (http://www.nagios.org) package is installed, you can configure your firewall to generate ping packets to hosts that nmap declares up, and the ping packets are immediately returned.

-sU UDP scan. This method is used to determine which UDP (User Datagram Protocol, RFC 768) ports are open on a host. This technique is to send 0 byte udp packets to each port on the target machine. If we receive an ICMP port unreachable message, then the port is closed. Otherwise we assume it is open.

Some people find UDP scanning is pointless. I usually remind them of the recent Solaris tcpd hole. Rpcbind can be found hiding on an undocumented UDP port somewhere above 32760. It doesn't matter that 111 is blocked by the firewall. But you can find which of the more than 30,000 high ports it is listening on! With a UDP scanner you can! There is also the cdc Back Orifice hackdoze program which hides on a configurable UDP port on Windows machines. Not to mention the many commonly vulnerable services that utilize UDP such as smtp, http, NFS, etc.

Unfortunately UDP scanning is sometimes painfully slow since most hosts implement a suggestion in RFC 1812 (section 4.3.2.8) of limiting the ICMP error message rate. For example, the Linux kernel (in net/ipv4/icmp.h) will later limit the rate of ICMP messages to 100 per second for each address that is exceeded. Solaris has much more strict limits (about 2 messages per second) and thus takes even longer to scan. nmap detects this rate limiting and slows down accordingly, rather than flood the network with useless packets that will be ignored by the target machine.

As is typical, Microsoft ignores the suggestion of the RFC and does not seem to do any rate limiting at all. Flooding Windows NT may not work as well.

-sA TCP ACK scan. This advanced method is usually used to map out firewall rules. In particular, it can help determine whether a firewall is stateful or just a simple packet filter that blocks incoming SYN packets.

This scan type sends an ACK packet (random looking acknowledgement/sequence numbers) to the ports specified. If a RST comes back, the port is classified as "filtered". If nothing comes back (and an ICMP unreachable is returned), the port is classified as "silenced". Note that nmap usually doesn’t print "unfiltered" ports, so geting no ports shown in the output is usually a sign that all the probes got through (and returned RSTs). This scan will obviously never show ports in the "open" state.
The Window scan. This advanced scan is very similar to the ACK scan, except that it can sometimes detect open ports as well as filtered/allowed due to an anomaly in the TCP window size reporting by some operating systems. Systems vulnerable to this include at least some versions of AIX, Amiga, BSDI, Cray, EnviROS, Digital UNIX, FreeBSD, HP-UX, IRIX, Mach, OpenBSD, OpenVMS, QNX, Rhapsody, Solaris 4.x, Ultrix, VAX, and V/Work. See the nmap hackers mailing list archive for a full list.

The RPC scan. This method works in combination with the various port scan methods of Nmap. It takes all the TCP/UDP ports found open and then floods them with SunRPC program NULL commands in an attempt to determine whether they are RPC ports, and if so, what program and version number they serve up. Thus you can effectively obtain the same info as firewall (or protected by TCP wrappers). Decoys do not currently work with RPC scan, at some point I may add decoy support for UDP/RPC scans.

This option activates remote host identification via TCP/IP fingerprinting. In other words, it uses a bunch of techniques to detect subtleties in the underlying operating system network stack of the computers you are scanning. It uses this information to create a ‘fingerprint’ which it compares with in a database of known OS fingerprints (the map of fingerprints file) to decide what type of system you are scanning.

If you find a machine that is misdiagnosed and has at least one port open, it would be useful if you mail me the details (ie OS blah version foo was detected as OS blah version bar). If you find a machine with at least one port open for which map says ‘unknown operating system’, then it would be useful if you send me the IP address along with the OS name and version number. If you can’t send the IP address, the next best thing is to run nmap with the -o option and send me the three fingerprints that should result along with the OR name and version number. By doing this you contribute to the pool of operating systems known to nmap and thus it will be more accurate for everyone.

If this turns on TCP reverse ident scanning. As noted by Dave Goldsmith in a 1996 Bugtraq post, the ident protocol ([1, 4]) allows for the disclosure of the username that owns any process connected via TCP, even if that process didn’t initiate the connection. So you can, for example, connect to the http port and then use ident to find out whether the server is running as root. This can only be done with a full TCP connection to the target port (ie the -s scanning options). When -I is used, the remote host’s ident is queried for each open port found. Obviously this won’t work if the host is not running ident.

This option causes the requested SYN, FIN, XMAS, or NULL scan to use tiny fragmented IP packets. The idea is to split up the TCP header into several packets to make it harder for packet filters, intrusion detection systems, and other annoyances to detect what you are doing. Be careful with this; Some programs have trouble handling these tiny packets. My favorite sniffer summations flicked immediately upon receiving the first 36-byte fragment. After that comes a 24-byte one! While this method won’t get by packet filters and intras that queue all IP fragments (like the CONFIG_IP_ALWAYS_DEFrag in the Linux kernel), some networks can’t afford the performance hit this causes and thus leave it disabled.

Note that I do not yet have this option working on all systems. It works fine for my Linux, FreeBSD, and OpenBSD boxes and some people have reported success with other *NIX variants.

This way you can get rid of the default ident options, and it gives you more information about what is going on. You can use it twice for greater effect. Use -d a couple of times if you really want to get crazy with scanning the screen!

This logs the results of your scan in a normal human readable form into the file you specify as an argument.

This logs the results of your scan in a machine parsable form into the file you specify as an argument. You can give the argument -t (without quotes) to shoot output into stdout (for shell pipelines, etc). In this case normal output will be suppressed. Watch out for error messages if you use this they will still go to stderr. Also note that -w will cause some extra information to be printed.

This logs the results of your scan in a machine parsable form into the file you specify as an argument. You can give the argument -t (without quotes) to shoot output into stdout (for shell pipelines, etc). In this case normal output will be suppressed. Watch out for error messages if you use this they will still go to stderr. Also note that -w will cause some extra information to be printed.

This option activates remote host identification via TCP/IP fingerprinting. In other words, it uses a bunch of techniques to detect subtleties in the underlying operating system network stack of the computers you are scanning. It uses this information to create a ‘fingerprint’ which it compares with in a database of known OS fingerprints (the map of fingerprints file) to decide what type of system you are scanning.

If you find a machine that is misdiagnosed and has at least one port open, it would be useful if you mail me the details (ie OS blah version foo was detected as OS blah version bar). If you find a machine with at least one port open for which map says ‘unknown operating system’, then it would be useful if you send me the IP address along with the OS name and version number. If you can’t send the IP address, the next best thing is to run nmap with the -o option and send me the three fingerprints that should result along with the OR name and version number. By doing this you contribute to the pool of operating systems known to nmap and thus it will be more accurate for everyone.
aborted scan. No other options can be given (they will be the same as the aborted scan). Nmap will start on the machine after the last one successfully scanned in the log file.

-Il <input filename>
Reads target specifications from the file specified RATHER than from the command line. The file should contain a list of host or network expression separated by spaces, tabs, or newlines. Use a higher (-i getMaxFilename) if you want nmap to read host expressions from stdin (like at the end of a pipe). See the section target specification for more information on the expressions you fill the file with.

-IR
This option tells Nmap to generate its own hosts to scan by simply picking random numbers. It will never end. This can be useful for statistical sampling of the Internet to examine various things. If you are ever really bored, try "−IRp 80" to find some web servers to look at.

-p <port ranges>
This option specifies what ports you want to scan. For example, "p 23" will only try port 23 of the target host(s). "p-20-101,109,8000-" scans ports between 20 and 30, port 19, and all ports greater than 60000. The default is to scan all ports between 1 and 1024 as well as any ports listed in the services ffile which comes with nmap.

−F Fast scan mode.
Specifies that you only wish to scan for ports listed in the services file which comes with nmap. This is obviously much faster than scanning all 65535 ports on about.

−D <decoy 2 MFL> Causes a "decoy" scan to be performed which may make it appear to the remote host that the hosts you specify as decoys are scanning the target network too. Thus their IDS might report 5-10 port scans from unique IP addresses, but they won't know which IP was scanning them and which were innocent decoys. While this can be defended through router path tracing, response-dropping, and other "active" mechanisms, it is generally an extremely effective technique for hiding your IP address.
Separate each decoy host with a comma, and you can optionally use "ME" as one of the decoys to represent the position you want your IP address used. If you put "ME" in the 6th position or later, some common port scan detectors (such as Solaris' excellent scancli) are unlikely to show your IP address at all. If you don't use "ME", nmap will put you in a random position.
Note that the hosts you use as decoys should be up or you might suddenly SYN flood your targets. Also it will be pretty easy to determine which host is scanning if only one is actually up on the network. You might want to use IP addresses instead of names (so the decoy network doesn't see your other nameservers)
Also note that some (simple) "port scan detectors" will firewallially block hosts that attempt port scans. Thus you might inadvertently cause the machine to be firewalldr by the decoy machines you are using. This could cause the target machines major problems if the decoy is, say, an internet gateway or even "localcast". Thus you might want to be careful of this option.
Another possible use of this is that detectors of spoofed port scans should not take active attacks against the machine that seems like it's port scanning them. It could just be a decoy!
Decoys are used both in the initial ping scan (using ICMP SYM, ACK, or whatever) and during the actual port scanning phase. Decoys are also used during remote OS detection (−O).
It is worth noting that using too many decoys may slow your scan and potentially even make it less accurate. Also, some IPS will filter out your spoofed packets, although currently most do not restrict spoofed IP packets at all.

−S <IP address>
In some circumstances, nmap may not be able to determine your source address ( nmap will tell you if this is the case). In this situation, use −S with your IP address (of the interface you wish to send packets through).
Another possible use of this flag is to spoof the scan to make the target think that someone else is scanning them. Imagine a company being repeatedly port scanned by a competitor? This is not a supported usage (or the main purpose) of this flag. I just think it raises an interesting possibility that people should be aware of before they go accusing others of port scanning them.

−e <interface>
Tells nmap what interface to send and receive packets on. Nmap should be able to detect this but it will tell you if it can't.

−g <portnumber>
Sets the source port number used in scans. Many naive firewall and packet filter installations make an exception in their ruleset to allow DNS (53) or FTP/DAI (20) packets to come through and establish a connection. Obviously this completely subverts the security advantages of the firewall since intruders can just masquerade as FTP or DNS by modifying their source port. Obviously for a UDP scan you should try 53 first and TCP scans should try 20 before 53. Notice that this is the only request −nmap will honor only if and when it is able to. For example, you can't do TCP SYN sampling all from one host port to one host port so nmap changes the source port even if you used it.
Be aware that there is a small performance penalty on some scans for using this option, because I sometimes store useful information in the source port number.

−r Tells Nmap NOT to randomize the order in which ports are scanned.
−randomize_hosts
Tells Nmap to shuffle each group of up to 2048 hosts before it scans them. This can make the scan less obvious to various network monitoring systems, especially when you combine it with slow timing options (see below).

−M <max sockets>
Sets the maximum number of sockets that will be used in parallel for a TCP scan (the default). This is useful to slow down the scan a little but and avoid crashing remote machines. Another approach is to use −S which is generally easier for machines to handle.

TIME OPTIONS
Generally Nmap does a good job at adapting for Network characteristics at runtime and scanning as fast as possible while minimizing that chances of hosts/ports going undetected. However, there are some cases where Nmap's default timing policy may not meet your objectives. The following options provide a fine level of control over the scan timing.

−T 0 (Paranoid Stealthy) Normal Aggressive) incass
These are canned timing policies for conveniently exporing your priorities to Nmap. Paranoid mode scans very slowly in the hopes of avoiding detection by IDS systems. It saturates all scans (no parallel scanning) and generally waits at least 5 minutes between sending packets. Sneaky is similar, except it only waits 15 seconds between sending packets. Police is meant to ease load on the network and reduce the chances of crashing machines. It saturates the probes and waits at least 0.4 seconds between them. Normal is the default Nmap behavior, which tries to run as quickly as possible without overwhelming the network or missing hosts/ports. Aggressive mode adds a 5 minute timeout per host and it never waits more than 1.25 seconds for probe responses. Incass is only suitable for very fast networks or where you don't mind losing some information. It times out hosts in 75 seconds and only waits 0.3 seconds for individual probes. It does allow for very quick network sweeps though :). You can also reference these by number (0-5). For example, "−T 10" gives you Paranoid mode and "−T 5" is Incass mode.
These canned timing modes should NOT be used in combination with the lower level controls given below.

−host_timeout <milliseconds>
Sets the amount of time Nmap is allowed to spend scanning a single host before giving up on that IP. The default timing mode has no host timeout.
-max_rtt_timeout <milliseconds>

Specifies the maximum amount of time Nmap is allowed to wait for a probe response before retransmitting or timing out that particular probe. The default mode sets this to about 9000.

-min_rtt_timeout <milliseconds>

When the target hosts are known to be responding very quickly, Nmap will shrink the amount of time given per probe. This speeds up the scan, but can lead to missed packets when a response takes longer than usual. With this parameter you can guarantee that Nmap will wait at least the given amount of time before giving up on a probe.

-initial_rtt_timeout <milliseconds>

Specifies the initial probe timeout. This is generally only useful when scanning筚red host with -P0. Normally Nmap can obtain good RTT estimates from the ping and the first few probes. The default mode uses 6000.

-max_parallelism <number>

Specifies the maximum number of scans Nmap is allowed to perform in parallel. Setting this to one means Nmap will never try to scan more than 1 port at a time. It also effects other parallel scans such as ping sweep, RPC scan, etc.

-scan_delay <milliseconds>

Specifies the minimum amount of time Nmap must wait between probes. This is mostly useful to reduce network load to do the scan; slow down under IDS thresholds.

TARGET SPECIFICATION

Everything that isn’t an option (or option argument) in nmap is treated as a target host specification. The simplest case is listing single hosts or IP addresses on the command line. If you want to scan a subnet of IP addresses, you can append “/mask” to the hostname or IP address. mask must be between 0 and the whole address space. We are testing whether the systems run sshd, DNS, pop3d, imapd, or port 456. Note that Xenus scan doesn’t work on Microsoft boxes due to their deficient TCP stack. Some goles with CHSEC, BRX, HIPUX, and BSDI boxes.

nmap -v -sX -p 80,8080,123,4564.128.201.1-127

Rather than focus on a specific IP range, it is sometimes interesting to slice up the entire internet and scan a small sample from each slice. This command finds all web servers on machines with IP addresses ending in 23.2.4. or 256 find more interesting machines starting at 127 so you might want to use 127.2-22 instead of the first asterisk because that section has a greater density of interesting machines (DHMO).

host -4 company.com | cut -d . -f 4 | ./nmap -AL

Do a DNS zone transfer to find the hosts in company.com and then feed the IP addresses to nmap. The above commands are for my GNU/Linux box. You may need different commands/ options on other operating systems.

BUGS

What bugs? Send me any that you find. Patches are nice too :) Remember to also send in new OS fingerprints so we can grow the database. Nmap will give you a submission URL when an appropriate fingerprint is found.

AUTHOR

Fyodor <fyodor@dhp.com>

DISTRIBUTION

The newest version of nmap can be obtained from http://www.insecure.org/nmap/
nmap is (C) 1997, 1998, 1999, 2000 by Fyodor (fyodor@insecure.org)

depcap is also distributed along with nmap. It is copyrighted by Van Jacobson, Craig Leres and Steven McCanne, all of the Lawrence Berkeley National Laboratory, University of California, Berkeley, CA. The version distributed with nmap may be modified, provided sources are available from http://www.lbl.gov/depcap.tar.Z.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; Version 2. This program is distributed in the hope that it will be useful, modify, and redistribute nmap under certain conditions. If this license is unacceptable to you, insecure.org may be willing to sell alternate licenses (contact fyodor@dhp.com).

Source code also allows you to audit the software for security holes (none have been found so far).

Source code also allows you to port nmap to new platforms, fix bugs, and add new features. You are highly encouraged to send any changes to Fyodor for possible incorporation into the main nmap distribution. By sending these changes to Fyodor or nmap-hackers, it is assumed that you are offering Fyodor the unlimited, non-exclusive right to reuse, modify, and redistribute the code. If you wish to specify special license conditions of your contributions, please state them up front.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details (it is in the COPYING file of the nmap distribution).

All versions of nmap equal to or greater than 2.0 are believed to be Year 2000 (Y2K) compliant in all respects. There is no reason to believe version earlier than 2.0 are susceptible to problems, but we have not tested them.

EXAMPLES

Here are some examples of using nmap, from simple and normal to a little more complex/semi-cronic. Note that actual numbers and some actual domain names are used to make things more concrete. In their place you should substitute addresses/hostname from your own network. I do not think perceiving other network is illegal but should penetrate be constrained by others as an attack. I have scanned hundreds of thousands of machines and have received only one complaint. But I am not a lawyer and some (mal) people may be annoyed by scan probes. Get permission first or test on your own risk.

nmap -e target.example.com

This option scans all reserved TCP ports on the machine target.example.com. The -v means turn on verbose mode.

nmap -e -O target.example.com/24

Launches a stealth SYN scan against each machine that is up out of the 255 machines on class ‘C’ where target.example.com resides. It also tries to determine what operating system is running on each host that is up and running. This requires extra privileges because of the SYN scan and the OS detection.

nmap -sX -p 22,80,143,154,4564,128.201.1-127

Sends an Xmas tree scan to the first half of each of the 255 possible 8 bit subsets in the 128.201 class ‘B’ network. We are testing whether the systems run sshd, DNS, pop3d, imapd, or port 456. Note that Xenus scan doesn’t work on Microsoft boxes due to their deficient TCP stack. Some goles with CHSEC, BRX, HIPUX, and BSDI boxes.
Appendix D

How To ...

D.1 Install the Smart IDS Add-On and the Mail-Parser

This section describes the installation process for our smart IDS add-on and the mail-parser on a host running Dragon IDS. We refer to section 2.4.1 for a description of Dragon.

Since the Dragon mail-parser makes use of the Dragon log-files, it must be installed on the same machine as the Dragon Rider Server. The following requirements need to be met:

- There must be a way incoming Dragon real-time alert e-mails can be piped into a program. We use procmail [36] to pipe Dragon real-time alerts into the Dragon mail-parser (mailparser.pl).

- Our code is written in perl [37]. Since we make use of function pointers, version 5.6 or above is needed. We tested our smart IDS with perl 5.6.0.

We installed Dragon and our smart extensions on a Redhat Linux 7.0 box.

D.1.1 Nessus

For the vulnerability scanner module, Nessus [26] is required. We tested our smart IDS with Nessus 1.0.7. This version is included on the enclosed CD in the software/tools/nessus directory. The parts of the Nessus software package we use are installed with the following steps:

NASL Script Interpreter

1. nessus-libraries ⇒ libnessus.so.1
   a) ./configure --enable-release --enable-nessuspcap
   b) make
   c) make install
   d) if needed, add /usr/local/lib (or corresponding path) to linker path (add it to /etc/ld.so.conf and run ldconfig)

2. libnasl (needs libnessus.so.1) ⇒ nasl (NASL interpreter executable)
   a) ./configure
b) make

c) make install

d) if needed, add /usr/local/bin (or corresponding dir) to PATH environment variable

Nessus Plugins

1. nessus-core (needs libnessus and libnas1)
   
a) cd nessus-core/nessusd/
   
b) patch < nessus-core-patch.txt
   
c) cd ..
   
d) ./configure --enable-release --disable-gtk
   
e) make
   
f) make install

2. nessus-plugins (needs include files from nessus-core) ➔ plugins
   
a) ./configure
   
b) make
   
c) make install

The required Nessus components are now installed on the system.

D.1.2 The Smart IDS Add-On

The Smart IDS add-on can be found on the CD in the sids directory. The following steps installs the Smart IDS:

1. Copy the sids directory from the CD to a local directory.

2. Configure procmail (or whatever program you use) to pipe the Dragon real-time alert e-mails into mailparser.pl. Figure D.1 shows the .procmailrc file we used.

3. Depending on the installation locations, you may need to adapt the file-paths in the perl files. For example, you need to specify the location of the nas1 executable in the SIDS_scanner.pm. The configuration variables are in the upper part of every perl-file and described with inline comments.

4. If a newer version of Nessus is installed, you have to update the cve2nessus file, as described in the next section.

D.2 Add New Vulnerability Checks

In section 5.5.3 we explained the various aspects of incorporating Nessus in the vulnerability scanner module. In the following we present the preferred way of adding new plugins to the module.

1. Get the plugin scripts (for example from [26]).
MAILDIR=$HOME  # you'd better make sure it exists
DEFAULT=/var/mail/root  # completely optional
LOGFILE=$HOME/maillog  # recommended
VERBOSE=yes

:0
{: c
| perl /root/mailparser.pl
}
 :0
/var/mail/root

Figure D.1: The configuration file for procmail we use to pipe Dragon real-time alert
e-mails into our mail-parser.

2. Depending on the trust in the source of the NASL-scripts, you may want to
review the scripts manually.

3. Copy the scripts to the nessus plugin directory
(usually /usr/local/lib/nessus/plugins).

4. Update the cve2nessus database. For this purpose, there is a perl-script in
the tools directory called gen_cve2nessus.pl. This tool accomplishes two
tasks while iterating over all plugins in a specific directory: First it scans
each plugin for a port description, and second it decides whether or not this
script is web-based. The output can directly be used as cve2nessus entries.

D.3 Add New Signatures

1. Add the new signatures to the desired Dragon sensors.

2. Configure Alarmtool such that it emanates a real-time alert e-mail for each
new signature. We refer to the Dragon documentation for details concerning
Dragon configuration.

3. Update the dragon2cve file. We wrote the script gen_dragon2cve.pl —
located in the tools directory — which performs this task.

The gen_dragon2cve.pl tool takes the name of a Dragon signature file as an
argument and prints the dragon2cve entries to standard output. On standard
error, progress messages and a summary are displayed. Therefore, the intended
usage is to redirect standard output to the dragon2cve file.

D.4 Clean the Signatures File

For our accuracy tests we wanted to remove rules from the signature-files that are
not related to attacks or vulnerabilities in a conventional sense. For example, rules
for patterns of stock exchange or job sites. We wrote a script that filters out such
rules. This script is called clean_siglib.pl and is located in the tools directory.
It takes a signature file name as argument and prints the new rules without the
filtered ones to standard output, allowing for redirection to another file.
D.5 Update the Database Module

The information in the configuration databases is entered in two different ways. The static database contains information that a human operator decided to enter. For the dynamic database, we wrote a script that uses `nmap` [35] to do a portscan and formats the output of `nmap` such that it can be used as dynamic database entries.

This script is called `gen_dbdynamic.pl` and is located in the `tools` directory. It takes one argument which is passed directly to `nmap` as the host specification. Therefore, multiple hosts can be scanned in one run. The output of `nmap` is processed and the closed ports of each host are dumped in the suited format to standard output.

The main purpose of this script is to “reverse” the port information delivered by `nmap` since `nmap` reports open ports and the configuration database specifies closed ports.

\[1\] The `nmap` manual page can be found in appendix C.7.
Appendix E

Contents of Enclosed CD

This appendix describes the contents of the enclosed compact disc. Figure E.1 shows a directory listing and table E.1 explains the contents of each directory.

**Figure E.1**: This figure shows the directories contained on the enclosed compact disc.
<table>
<thead>
<tr>
<th>Directory</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>docu</td>
<td>Documentation of Our Diploma Thesis</td>
</tr>
<tr>
<td>manuals</td>
<td>Software Manuals</td>
</tr>
<tr>
<td>dragon</td>
<td>Dragon IDS Manual</td>
</tr>
<tr>
<td>nessus</td>
<td>Nessus Vulnerability Scanner Manual</td>
</tr>
<tr>
<td>papers</td>
<td>Papers Regarding Intrusion Detection</td>
</tr>
<tr>
<td>presentation</td>
<td>Foils and other Material from Our Presentations</td>
</tr>
<tr>
<td>sids</td>
<td>Our Smart IDS Add-On and Mail-Parser Software</td>
</tr>
<tr>
<td>software</td>
<td>Third Party Software We Used for Our Work</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection Systems</td>
</tr>
<tr>
<td>dragon</td>
<td>Dragon IDS Software</td>
</tr>
<tr>
<td>snort</td>
<td>Snort IDS Software</td>
</tr>
<tr>
<td>tools</td>
<td>Tools</td>
</tr>
<tr>
<td>ethereal</td>
<td>Ethereal Packet Analyzer</td>
</tr>
<tr>
<td>nemesis</td>
<td>Nemesis Packet Generator</td>
</tr>
<tr>
<td>nessus</td>
<td>Nessus Vulnerability Scanner</td>
</tr>
<tr>
<td>vmware</td>
<td>VMWare Software</td>
</tr>
<tr>
<td>wiropdump</td>
<td>TCPDump for Windows</td>
</tr>
</tbody>
</table>

**Table E.1**: Description of the contents of the enclosed compact disc.
Bibliography

General Computer Security

[1] Stuart McClure & Joel Scambray:
*Hacking Exposed*

*Network Security Secrets & Solutions,*
ISBN 0-07-212127-0

[2] CVE Editorial Board:
*Common Vulnerabilities and Exposures,*
http://cve.mitre.org

[3] Fyodor:
*Remote OS detection via TCP/IP Stack FingerPrinting,*

[4] *L0pht Heavy Industries,*
http://www.l0pht.com

Related Papers

[5] Steven Cheung et al:
*THe Design of GrIDS: A Graph-Based Intrusion Detection System,*
http://seclab.cs.ucdavis.edu/arps/grids
January 26, 1999.

[6] Chirs Sinclair, Lyn Pierce, Sara Matzner:
*An Application of Machine Learning to Network Intrusion Detection,*
http://www.computer.org/Proceedings/acsac/0346/03460371abs.htm
December 1999.

*Intelligent Agents for Intrusion Detection,*
http://citeseer.nj.nec.com/helmer98intelligent.html
1998.

[8] Stefan Axelsson:
*The Base-Rate Fallacy and Its Implications for the Difficulty of Intrusion Detection,*
http://www.ce.chalmers.se/staff/sax/difficulty.ps

[9] Peter Mell & Marc McLarnon:  
*Mobile Agent Attack Resistant Distributed Hierarchical Intrusion Detection Systems*,  
National Institute of Standards and Technology, Gaithersburg, 1999.

[10] Wenke Lee, Salvatore J. Stolfo, Kui W. Mok:  
*A Data Mining Framework for Adaptive Intrusion Detection*,  
Computer Science Department, Columbia University, New York, 1998.

*Intrusion Detection for an On-Going Attack*,  

[12] Terrence G. Champion, Robert S. Durst:  
*Air Force Intrusion Detection System Evaluation Environment*,  

[13] Stefanos Manganaris, Marvin Christensen, Dan Zerkle, Keith Hermitz:  
*A Data Mining Analysis of RTID Alarms*,  

*NetSTAT: A Network-based Intrusion Detection System*,  

[15] Zhang Qianli:  
*Session State Transition Based Large Network IDS*,  

**Intrusion Detection**

[16] Stephen Northcutt:  
*Network Intrusion Detection*  
*An Analyst’s Handbook*,  
ISBN 0-7357-0868-1

[17] Rebecca Gurley Bace:  
*Intrusion Detection*,  
ISBN 1-57870-185-6

[18] SANS Institute Online:  
*A Cooperative Education & Research Organization*,  
http://www.sans.org

*Insertion, Evasion, and Denial of Service:  
Eluding Network Intrusion Detection*,
Secure Networks, Inc.

Network Flight Recorder,
2000.

TCP/IP

[21] Douglas E. Comer:
*Internetworking with TCP/IP*
*Volume 1*
*Principles, Protocols, and Architecture*

Software

[22] Internet Security Systems:
*RealSecure IDS*,
http://www.iss.net

[23] Network Flight Recorder:
*Network Flight Recorder IDS*,
http://www.nfr.com

[24] Network Security Wizards:
*Dragon IDS*,
http://www.securitywizards.com

[25] Open Source Community:
*Snort IDS*,
http://www.snort.org

[26] Open Source Community:
*Nessus*,
http://www.nessus.org

[27] Open Source Community:
*libpcap*,
Promiscuous mode packet capture library,
http://www.tcpdump.org

[28] Open Source Community:
*tcpdump*,
Promiscuous mode packet capture frontend,
http://www.tcpdump.org
[29] Open Source Community: 
ethereal,
A network protocol analyzer,
http://www.ethereal.com

[30] Open Source Community: 
nidsbench,
Network-based Intrusion Detection Test Suite,
http://www.anzen.com/research/nidsbench

[31] Open Source Community: 
tcpreplay,
Part of nidsbench Network-based Intrusion Detection Test Suite,
http://www.anzen.com/research/nidsbench

[32] Open Source Community: 
fragrouter,
Part of nidsbench Network-based Intrusion Detection Test Suite,
http://www.anzen.com/research/nidsbench

[33] Open Source Community: 
libnet,
A packet assembly system,
http://www.packetfactory.net/libnet

[34] Open Source Community: 
nemesis,
A packet injection tool suite,
http://celerity.bartoli.org/nemesis

[35] Open Source Community: 
nmap,
Stealth port scanner,
http://www.insecure.org/nmap

[36] Open Source Community: 
procmail,
Mail processing tool,
http://www.procmail.org

[37] Open Source Community: 
perl,
Scripting language,
http://www.perl.com

Latex

[38] Tobias Oetiker, Hubert Partl, Irene Hyna and Elisabeth Schlegl: 
The Not So Short Introduction to \LaTeX, 
June 1999.
[39] Michel Goossens, Frank Mittelbach, Alexander Samarin: 
Der \textit{\LaTeX\ Begleiter},
Addison-Wesley, 1995. 
ISBN 3-89319-646-3

\section*{Miscellaneous}

[40] DARPA Internet Program Protocol Specification 
\textit{Internet Protocol},
http://www.ietf.org/rfc/rfc0791.txt 

[41] Richard Stallman: 
\textit{GNU General Public License},
http://www.gnu.org/copyleft/gpl.txt 
1989.
List of Figures

1.1 A Classification of Intrusion Detection Systems .......................... 1
1.2 A Knowledge-Based Intrusion Detection System in Action .............. 2
1.3 IDS Terminating TCP Connection of an Attacker ....................... 5
1.4 A DotDot Attack ........................................... 6
1.5 A DotDot Attack Terminated by an IDS ............................. 6
1.6 Network-Based IDS on Switched Networks ............................ 7
1.7 Insertion Attacks ........................................... 10
1.8 Evasion Attacks ........................................... 10
1.9 IP Fragment Overlapping ...................................... 12
1.10 Insertion Attack Using Wrong Link-Layer Addresses ................. 12

2.1 IDS Functionalities ........................................... 16
2.2 Dragon Concept ............................................. 17
2.3 Screenshot of the Dragon Fire Interface ............................. 18
2.4 Dragon Pattern Sample ....................................... 18
2.5 RealSecure Concept ......................................... 20
2.6 Screenshot RealSecure Policy Editor ............................. 21
2.7 NFR Concept ................................................. 22
2.8 N-Code Example ............................................ 23
2.9 Short Concept .............................................. 24
2.10 Short Rule Example ......................................... 24

3.1 Interaction of Extensions with IDS .................................. 29
3.2 Nessus Security Scanner: Plugin Selection .......................... 31
3.3 Nessus Security Scanner: Scan Results ............................. 31
3.4 Expired Information Taken from a Dynamic Database ................ 32
3.5 Correlation of Different Pattern-Matches ........................... 34
3.6 Sequence of Pattern-Matches .................................. 34
3.7 The Concept of Severity Levels ................................ 35

4.1 System Behaviour Before and After Attack .......................... 40
4.2 Attack Classification by Impact on Investigations ................... 41
4.3 Extended Attack Classification by Impact on Investigations ........ 42

5.1 The SIDS ...................................................... 46
5.2 The Dragon Mail-Parser ....................................... 47
5.3 Real-Time Alerts from the Dragon IDS ............................. 48
5.4 A Part of the dragon2cve File .................................. 49
5.5 The SIDS-Core .............................................. 50
5.6 A Part of the Policy-File ..................................... 50
5.7 SIDS-Core Detail ........................................... 52
5.8 A Sample Policy Configuration File .......................... 52
5.9 Example Policy-File Using Macros .......................... 53
5.10 The Module-Caller ......................................... 53
5.11 Host Tester Detail ........................................ 54
5.12 TCP Port Tester Detail .................................... 55
5.13 Nessus Vulnerability Scanner ............................... 56
5.14 Vulnerability Scanner Module ............................... 57
5.15 A Sample cve2nessus File ................................ 58
5.16 Configuration Database Detail .............................. 59
5.17 Configuration Database Sample ............................. 60

6.1 A Test Environment .......................................... 61
6.2 Nmap-Scan Against the Linux Victim ....................... 62
6.3 Nmap-Scan Against the Windows NT Victim ................ 63
6.4 Policy-File Macros For The Accuracy Test ................ 65
6.5 Accuracy Test Policy-File .................................. 65
6.6 Generated and Discarded Alerts for the Linux Victim .... 67
6.7 Processed and Discarded Alerts for the Linux Victim ..... 67
6.8 Generated and Discarded Alerts for the NT Victim ......... 69
6.9 Processed and Discarded Alerts for the NT Victim ......... 69

C.1 Ethereal Protocol Analyzer .................................. 92

D.1 Procmail Configuration File ................................. 101

E.1 Contents of Enclosed CD ..................................... 103
List of Tables

1.1 Comparison of IDS Regarding Data Evaluation 3
1.2 Comparison of IDS Regarding Data Collection 4
1.3 IP Fragment Overlapping Behaviour for Different OSs 11

2.1 IDS Products Overview 25

3.1 Static vs. Dynamic Configuration Databases 33

4.1 Configuration Databases 43

5.1 Modules 54
5.2 Host Tester 55
5.3 TCP Port Tester 55
5.4 Vulnerability Scanner 56
5.5 Configuration Database 59

6.1 Attack Patterns in Dragon’s Signature Base 63
6.2 Attack-Type Distribution in the Policy-File 65
6.3 Test Results for Linux Victim 66
6.4 Attack-Type Distribution for Linux Victim 66
6.5 Scanner-Status for Attack on Victim Linux 68
6.6 Test Results for NT Victim 68
6.7 Attack-Type Distribution for NT Victim 70
6.8 Scanner-Status for Attack on Victim NT 70

E.1 Contents of Enclosed Compact Disc 104