Tuning an IDS/IPS From The Ground UP

GCIA Gold Certification

Author: Brandon Greenwood

Version 4.0

Table of Contents

List of Figures

Abstract

This paper examines one of the many different methodologies to configuring or tuning an Intrusion Detection System or Intrusion Prevention System (IDS/IPS). The proper configuration of an IDS is a bit of an art because there are so many different ways to do it. I have seen and listened to many people explain the ‘best’ way to configure a detection engine and while I don’t subscribe to a best way, I have taken bits and pieces from some of these methodologies and combined them into a system that has worked for me.

I have seen and listened to these individuals discuss ways to configure a detection engine. While there is documentation on the subject out there, I have yet to come across something that goes into the type of detail that this paper will. I have heard the complaint that just being told to properly tune pre-processors (SNORT specific but there are similar features in other offerings), or set a threshold hasn’t been much help. How do you determine where to set a specific pre-processor? How do you determine where to set a threshold versus suppression? How can you verify that you aren’t opening yourself up to false negatives by tuning a little too much out? It is my hope to answer these and other questions with this paper.

Document Conventions

This paper uses the following conventions throughout the length of the document. Certain words are represented in different fonts and typefaces. The types of words that are represented this way include the following:

command	Operating system commands are represented in this font style. This style indicates a command that is entered at a command prompt or shell.

computer output	The results of a command and other computer output are in this style
URL	Web URL’s are shown in this style.

Introduction

The planning, implementation, and care of an Intrusion Detection Sensor (IDS) in recent years has been the result of meeting a requirement or mandate for one of the many regulations not only in corporate America but also in the government space. All too often those very systems being asked to audit the environment are not given the same care as say a router or firewall. Intrusion Detection Systems are often just a check mark for an auditor, but when that auditor would like to see the process for how an IDS alert is handled or escalated to an event and discovers there are hundreds, perhaps thousands of alerts per day that no action is being taken on, not only will the person in charge of the system look bad but the organization could be penalized as well.

So why wouldn’t the proper resources be dedicated to making sure the full potential an IDS can bring to the table be utilized? The reasons for this are varied and many. Not having a thorough understanding of IDS technology, other jobs and responsibilities that consume the time of an overburdened engineer or analyst, not knowing or having an understanding of known good traffic on a network or what might constitute bad traffic can all contribute as reasons.

These same principles can and have been used in the implementation of Intrusion Prevention Sensor’s (IPS) and are not limited to only IDS. This document will not go into the installation of the IDS but rather what to do and ideas for how to configure it once it is up and running.

Background

This is a case study of an organization tasked with implementing an IDS solution as part of an initiative to meet regulatory compliance. They in turn have asked me if I would be willing to assist in the architecture and planning, setup, and maintenance of a solution to help them meet this requirement. I agreed to assist if I could use this opportunity to write a white paper outlining some of principles and practices of tuning an IDS for the security community. I have been allowed to do this with the only request being that I not divulge the organization I am assisting or any IP addressing. For this reason, Sacrificial Lamb Inc. (SLI), will represent the organization and 192.168.0.0/24 will represent the internal addressing scheme.

The Environment

The first thing to do is get a layout of the environment. We will be monitoring and identifying services offered on the network passively. Any special considerations or specific traffic we might need to be made aware of or might need special consideration will be identified while watching the traffic.

Upon interviewing the systems and network administrator I am given the following configuration:

1 x SMTP Server
2 x Web Servers (External facing)
2 x DNS Servers
2 x Domain Controllers
2 x DB Servers
~200 addressable nodes on the segment

All web, application, and database servers are on the same segment which is never a good design from a security or performance perspective but it could provide some interesting traffic. This paper isn't about the design of security architecture. But for this environment, the web segment could be put into a DMZ and separated from the application segment which could be separated from the database segment with at least one firewall.

I was not able to audit the firewall log but SLI informed me they had done a good job with the firewall rule base in allowing only what is explicitly required through both ingress and egress filtering.

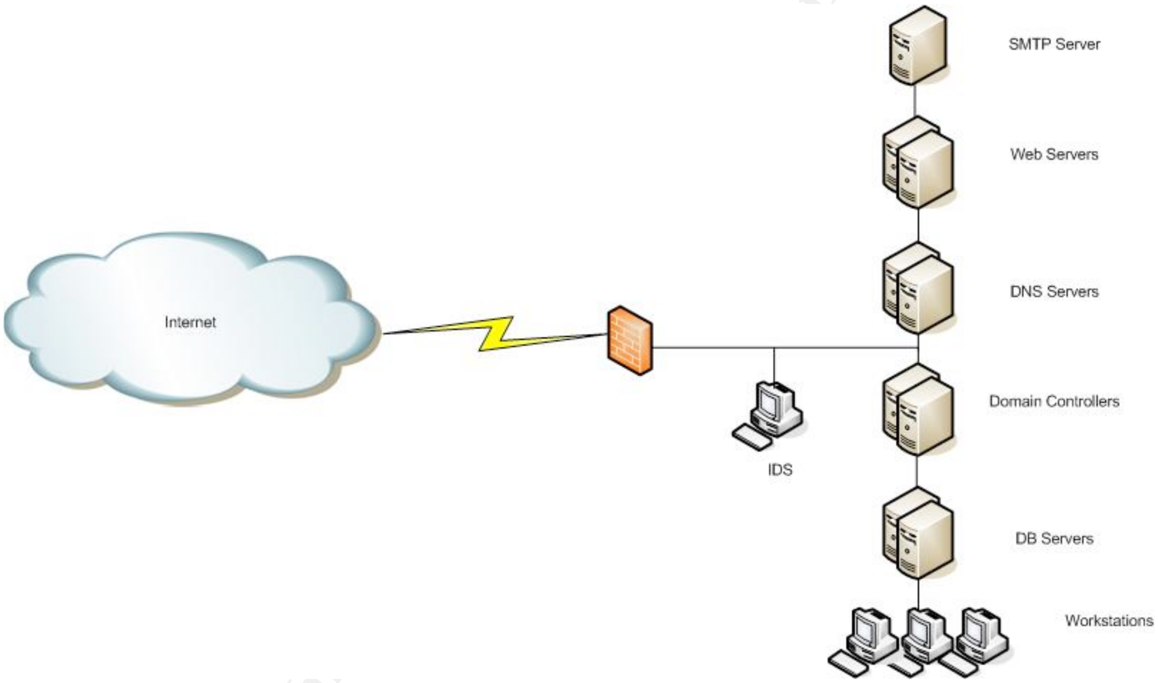

Figure 1

The IDS is deployed so it is monitoring all traffic into and out of the segment. SLI determined there would be no IDS monitoring on intra-VLAN traffic as they felt comfortable safeguards were in place so this type of analysis and it's requirements on the detection engine would not be needed. If there is any malicious type traffic communicating back out through the firewall to an unknown site/bot farm/individual, our IDS may pick up the traffic it has signatures or pre-processors available for and are not tuned out. A better solution for this type of analysis is to use techniques similar to extrusion detection or using Network

Security Monitoring (NSM). More information on this topic can be found in Richard Bejtlich's excellent book on the subject 'Extrusion Detection – Security Monitoring for Internal Intrusions'.

Intrusion Detection Sensor

Now that we have an idea of what this organization believes their environment looks like and where things are located, it's time to decide on an IDS. Since SLI has limited budgetary resources to devote to this project, after looking at some of the solutions on the market, they decided SNORT (http://snort.org/) would be the best fit for the environment.

The install is a SNORT/BASE/MySQL installation on CentOS 5 Linux. The IDS was setup with a standard config (no modifications yet), and the latest set of VRT certified rules. When tuning is complete using the standard set of VRT rules, SLI will have the option of including Community and Bleeding Edge Threats (http://www.bleedingsnort.com/) rules.

Getting Started

In order to test the IDS installation and initial configuration, the sensor was turned on for a 24 hour period over a weekend so that the user base wouldn't add any more potential false positives to the alert database than what we are hoping to get out of regular network traffic. While this may not be truly normal traffic as we are not aware of what services or jobs will be or won't be running, it is better than monitoring at noon on a Friday in my opinion. No tuning of the configuration files or rules/signatures are made for the test.

```
mysql> select count(signature) as count from event where timestamp>='2007-06-30 00:00:00' AND timestamp<='2007-06-30 23:59:59';

+--------+
| count  |
+--------+
| 180590 |
+--------+
1 row in set (0.06 sec)
```

We received 180,590 alerts in a 24 hour period. This might be a daunting task for an analyst to handle depending on the specific alert and how many times it’s being generated. Let’s see which alerts and how many of each were flagged during that same period:

```
mysql> select DISTINCT sig_name as 'Signature Name', count(*) sig_id  FROM
event, signature WHERE signature = sig_id AND timestamp>='2007-06-30
00:00:00' AND timestamp<='2007-06-30 23:59:59' GROUP BY sig_id ORDER
BY sig_id DESC;
+------------------------------------------------------------------------------------------+--------+
| Signature Name
| sig_id |
+------------------------------------------------------------------------------------------+--------+
| (portscan) Open Port
|  48223 |
| ICMP L3retriever Ping                                                                     |
28467 |
| WEB-MISC /etc/passwd
|  15082 |
| NETBIOS SMB-DS IPC$ unicode share access
|   8380 |
…
…
…
| ATTACK-RESPONSES id check returned root
|       1 |
| SNMP null community string attempt
|       1 |
| DNS TCP inverse query overflow
|       1 |
| RPC mountd TCP export request
|       1 |
+-------------------------------------------------------------------------------------------+--------
+
386 rows in set, 1 warning (1.28 sec)
```

There are two approaches that can be used in dealing with alerts and investigating what is happening. The first is to edit the *snort.conf* file and customize it’s configuration to SLI’s environment. The second is to investigate these 180,590 alerts individually. With a properly tuned configuration file, the amount of individual analysis that needs to be done can be greatly reduced. So we will start by modifying the *snort.conf* configuration file.

One of the problems with looking at 24 hours worth of data is in order to see the effect of changes in the configuration you will need to wait another 24 hours to see the results. Based on the dynamic nature of network traffic, this may not reflect what a previous sample looked like. One possible solution to this issue is to generate a 2 gigabit packet capture with tcpdump from the environment during the middle of the day. (What about your weekend test?) (The weekend test was just to see what the traffic looked like from SNORT’s perspective over a 24 hour period, it wasn’t a packet capture) The reason is that typically many false positives will be generated and we can use the same data to tune the *snort.conf* file to our liking.

A full session capture file was generated using tcpdump on the sensor itself lasting roughly 2 hours before the 2G limit was reached.   It can then be determined whether the alerts are not false positives, if the traffic needs to be dropped, or perhaps just logged. This should allow us to get the low hanging fruit alerts out of the way making analysis easier as the configuration and tuning moves forward.

Let’s take a look at what this capture contains with a tool that allows us to extract statistical information from the packet capture; tcpdstat.

```
[brandon@SLISnort]$ ./tcpdstat ~/corp_in2.pcap

DumpFile:  /home/brandon/corp_in2.pcap
FileSize: 2048.00MB
```

```
pcap_dispatch:truncated dump file; tried to read 1434 captured bytes, only
got 1344
Id: 200706101608
StartTime: Sun Jul 1 16:08:53 2007
EndTime:   Sun Jul 1 18:01:04 2007
TotalTime: 6730.51 seconds
TotalCapSize: 1985.11MB  CapLen: 1514 bytes
# of packets: 4121261 (1985.11MB)
AvgRate: 2.47Mbps  stddev:1.69M   PeakRate: 13.91Mbps

### IP flow (unique src/dst pair) Information ###
# of flows: 5642  (avg. 730.46 pkts/flow)
Top 10 big flow size (bytes/total in %):
 57.3%  5.5%  3.3%  2.2%  2.0%  1.6%  1.3%  1.0%  0.8%  0.7%

### IP address Information ###
# of IPv4 addresses: 2075
Top 10 bandwidth usage (bytes/total in %):
 59.6% 58.9%  6.5%  5.6%  4.5%  3.9%  3.5%  3.4%  3.2%  2.3%
### Packet Size Distribution (including MAC headers) ###
<<<<
 [   32-   63]:    1158243
 [   64-  127]:    1132547
 [  128-  255]:     358481
 [  256-  511]:     154003
 [  512- 1023]:     101690
 [ 1024- 2047]:    1216297
>>>>

### Protocol Breakdown ###
<<<<
     protocol          packets              bytes          bytes/pkt
--------------------------------------------------------------------
[0] total         4121261 (100.00%)      2081542087 (100.00%)    505.07
[1] ip           4121258 (100.00%)      2081541907 (100.00%)    505.07
[2]  tcp          3445485 ( 83.60%)      1999948988 ( 96.08%)    580.45
[3]   ftp             20 (  0.00%)            1459 (  0.00%)     72.95
[3]   ssh            474 (  0.01%)           77358 (  0.00%)    163.20
[3]   smtp          5474 (  0.13%)         2676432 (  0.13%)    488.94
[3]   dns           4320 (  0.10%)          411648 (  0.02%)     95.29
[3]   http(s)     440373 ( 10.69%)       503994842 ( 24.21%)   1144.47
[3]   http(c)     300768 (  7.30%)        46650411 (  2.24%)    155.10
[3]   kerb5         1635 (  0.04%)          465539 (  0.02%)    284.73
[3]   epmap        41345 (  1.00%)         3143822 (  0.15%)     76.04
[3]   netb-se      84414 (  2.05%)         7044090 (  0.34%)     83.45
[3]   ldap          2053 (  0.05%)          850972 (  0.04%)    414.50
```

```
[3]   https         99881 (  2.42%)       62950532 (  3.02%)    630.26
[3]   ms-ds        215976 (  5.24%)       39062655 (  1.88%)    180.87
[3]   socks           522 (  0.01%)         254018 (  0.01%)    486.62
[3]   kasaa           828 (  0.02%)          69146 (  0.00%)     83.51
[3]   mssql-s        2783 (  0.07%)         205863 (  0.01%)     73.97
[3]   squid            86 (  0.00%)          36369 (  0.00%)    422.90
[3]   ms-gc          4068 (  0.10%)         959935 (  0.05%)    235.97
[3]   ms-gcs          367 (  0.01%)          34961 (  0.00%)     95.26
[3]   hotline          20 (  0.00%)           4119 (  0.00%)    205.95
[3]   icecast        1784 (  0.04%)         117015 (  0.01%)     65.59
[3]   gnu6347          21 (  0.00%)           4179 (  0.00%)    199.00
[3]   gnu6348           2 (  0.00%)            122 (  0.00%)     61.00
[3]   gnu6350           7 (  0.00%)           1697 (  0.00%)    242.43
[3]   irc6666           2 (  0.00%)            122 (  0.00%)     61.00
[3]   napster          31 (  0.00%)           5801 (  0.00%)    187.13
[3]   http-a           78 (  0.00%)          63613 (  0.00%)    815.55
[3]   other       2238153 ( 54.31%)     1330862268 ( 63.94%)    594.63
[2]  udp           661192 ( 16.04%)       80149529 (  3.85%)    121.22
[3]   dns           24703 (  0.60%)        3023736 (  0.15%)    122.40
[3]   kerb5            34 (  0.00%)           9856 (  0.00%)    289.88
[3]   ntp             954 (  0.02%)          87780 (  0.00%)     92.01
[3]   epmap           366 (  0.01%)          75592 (  0.00%)    206.54
[3]   netb-ns      359523 (  8.72%)       33438920 (  1.61%)     93.01
[3]   netb-se       45448 (  1.10%)       11056121 (  0.53%)    243.27
[3]   other        230164 (  5.58%)       32457524 (  1.56%)    141.02
[2]  icmp            5025 (  0.12%)         621402 (  0.03%)    123.66
[2]  ospfigp         4172 (  0.10%)         402036 (  0.02%)     96.37
[2]  scps            5384 (  0.13%)         419952 (  0.02%)     78.00
>>>>
```

This high level statistical overview of the information can prove very useful. Care must be taken however to look at the packet capture to verify what tcpdstat is telling us. For example: the *squid* (tcp:3128) count is at 86. The 86 represents both source and destination ports in a conversation. Shown here:

```
[brandon@SLISnort]$ sudo tcpdump -r ~/corp_in2.pcap 'port 3128' | wc -l
reading from file /home/brandon/corp_in2.pcap, link-type EN10MB
(Ethernet)
tcpdump: pcap_loop: truncated dump file; tried to read 1434 captured
bytes, only got 1344
86
```

Using tcpdump with the ‘-r’ flag specifies a file to read in instead of capturing traffic directly off of the network card. A Berkley Packet Filter (BPF) filter is also appended to look for all traffic with either a source or destination port of 3128. What a BPF filter allows for is to specify capturing or looking at traffic based on filters that are passed to tcpdump, which in this case is port 3128. Running the entire trace file through this filter it also comes up with 86. In looking at the first packet:

```
[brandon@SLISnort]$ sudo tcpdump -r ~/corp_in2.pcap 'port 3128' -nnc 1
reading from file /home/brandon/corp_in2.pcap, link-type EN10MB
(Ethernet)
17:44:01.577327 IP 66.245.31.33.3128 > 192.168.0.13.80: S
2508597576:2508597576(0) win 65535 <mss 1380,nop,nop,sackOK>
```

We see that this is the ephemeral port for a web session. Again exercise caution when reading the output of tcpdstat not to assume because it lists a certain service as in use on the network, it really is that service. Given this, it does provide useful additional information about the packet capture that can be used in conjunction with other analysis tools.

Now that we have a general idea of what packet capture contains, let’s run SNORT against the same 2 Gig capture file and look at the results in MySQL:

```
mysql> select count(sid) as Total_Event from event;

+---------------+
| Total_Event |
+---------------+
|           7088 |
+---------------+
1 row in set (0.00 sec)
```

```
mysql> select DISTINCT sig_name as 'Siganture Name', count(*) as Total
FROM event, signature WHERE signature = sig_id AND timestamp>='2007-07-
```

```
01 00:00:00' AND timestamp<='2007-07-01 23:59:59' GROUP BY sig_id ORDER

BY Total DESC;

+-------------------------------------------------------------------------------+-------+
| Siganture Name                                                                | Total |
+-------------------------------------------------------------------------------+-------+
| NETBIOS SMB-DS IPC$ unicode share access                                      |  3249 |
| (portscan) Open Port                                                          |  1421 |
| ICMP L3retriever Ping                                                         |  1206 |
| WEB-IIS view source via translate header                                      |   332 |
| NETBIOS SMB IPC$ unicode share access                                         |   215 |
| WEB-MISC WebDAV search access                                                 |   141 |
| (portscan) TCP Portsweep                                                      |   122 |
| (http_inspect) DOUBLE DECODING ATTACK                                         |    92 |
| (http_inspect) OVERSIZE REQUEST-URI DIRECTORY                                 |    57 |
| ICMP PING NMAP                                                                |    53 |
| DOS MSDTC attempt                                                             |    46 |
| (http_inspect) BARE BYTE UNICODE ENCODING                                     |    37 |
| WEB-MISC search.dll access                                                    |    22 |
| WEB-MISC handler access                                                       |    21 |
| WEB-MISC weblogic/tomcat .jsp view source attempt                             |    16 |
| WEB-CLIENT Outlook EML access                                                 |    10 |
| WEB-MISC robots.txt access                                                    |     9 |
| WEB-CGI calendar access                                                       |     6 |
| ATTACK-RESPONSES 403 Forbidden                                                |     6 |
| (http_inspect) IIS UNICODE CODEPOINT ENCODING                                 |     5 |
| SNMP missing community string attempt                                         |     4 |
| WEB-PHP friends.php access                                                    |     4 |
| MISC MS Terminal server request                                               |     4 |
| DNS SPOOF query response with TTL of 1 min. and no authority                  |     3 |
| WEB-IIS WebDAV file lock attempt                                              |     2 |
| (portscan) TCP Portscan                                                       |     1 |
| WEB-CGI redirect access                                                       |     1 |
| WEB-MISC counter.exe access                                                   |     1 |
| NETBIOS SMB-DS ADMIN$ unicode share access                                    |     1 |
| WEB-MISC RBS ISP /newuser access                                              |     1 |
+-------------------------------------------------------------------------------+-------+
30 rows in set (0.03 sec)
```

Tuning

This is the most critical part of any IDS/IPS deployment. Everyone who has had the opportunity of setting up and configuring an IDS/IPS will undoubtedly have seen and had to deal with the consequences of an improperly tuned sensor.

While every option should be looked at for the particular environment, only those settings that need to be addressed by SLI are noted here.

Variables

Roughly 7100 alerts in 30 separate signature alertsand pre-processor alerts may be quite a lot in a two hour period, depending on the network. An analyst could go through these individual alerts and determine which events will need further investigation. Let's start by making the most basic of changes to the *snort.conf* file and specify the home network. The home network variable $HOME_NET can best be described as a segment or segments defined on the sensor that SNORT treats as the monitored segment. SNORT rules are configured with the HOME_NET variable typically as the network that you are trying to protect. By default, both the HOME_NET and EXTERNAL_NET variables are set to *any* in the *snort.conf*. The HOME_NET variable should not be kept as *any*. If it is set to "*any*", it *will* generate false positives, because the IDS has no idea which direction to start looking.

By changing the *snort.conf* HOME_NET variable from "*any*" to 192.168.0.0/24 and running the same pcap through SNORT again, we are given the following results:

```
mysql> select count(sid) as Total_Event from event;
+---------------+
| Total_Event |
+---------------+
|       7011 |
+---------------+
1 row in set (0.00 sec)
```

```
mysql> select DISTINCT sig_name as 'Siganture Name', count(*) as Total
FROM event, signature WHERE signature = sig_id AND timestamp>='2007-07-
01 00:00:00' AND timestamp<='2007-07-01 23:59:59' GROUP BY sig_id ORDER
BY Total DESC;
+--------------------------------------------------------------------------+-------+
| Siganture Name                                                           | Total |
+--------------------------------------------------------------------------+-------+
| NETBIOS SMB-DS IPC$ unicode share access                       |  3249 |
| (portscan) Open Port                                            |  1421 |
| ICMP L3retriever Ping                                          |  1206 |
| WEB-IIS view source via translate header                        |   332 |
| NETBIOS SMB IPC$ unicode share access                           |   215 |
| WEB-MISC WebDAV search access                                   |   141 |
| (portscan) TCP Portsweep                                         |   122 |
| (http_inspect) DOUBLE DECODING ATTACK                          |    92 |
| (http_inspect) OVERSIZE REQUEST-URI DIRECTORY                 |    57 |
| ICMP PING NMAP                                                 |    53 |
| DOS MSDTC attempt                                               |    46 |
| (http_inspect) BARE BYTE UNICODE ENCODING                     |    37 |
| WEB-CLIENT Outlook EML access                                  |    10 |
| WEB-MISC robots.txt access                                       |     9 |
| (http_inspect) IIS UNICODE CODEPOINT ENCODING                 |     5 |
| WEB-CGI calendar access                                          |     4 |
| MISC MS Terminal server request                                  |     4 |
| DNS SPOOF query response with TTL of 1 min. and no authority  |     3 |
| WEB-IIS WebDAV file lock attempt                                |     2 |
| (portscan) TCP Portscan                                          |     1 |
| SNMP missing community string attempt                             |     1 |
| NETBIOS SMB-DS ADMIN$ unicode share access                      |     1 |
+--------------------------------------------------------------------------+-------+
22 rows in set (0.02 sec)
```

The total number of alerts has dropped by 77 and the total signatures flagged dropped by 8. This is not a huge drop, however, this is the most basic step and the reduction of the number of alerts from 7088 to 7011 alerts is a good start.

It is also a good idea to move through the rest of the snort.conf and define the variables for SLI. (Sentence Fragment) The beauty of a variable is seen when a DNS server moves from say, 192.168.0.10 to 192.168.0.20, the rules pertaining to the DNS servers do not need to be individually changed. Making a change to the global DNS_SERVERS variable is all that is needed.
Additional variables can and should be added depending on the environment. If rules need to be created to define a Virtual Private Network (VPN) ranges a VPN variable can be added.

Here is an example of how we can further define several of the variables in the *snort.conf file*:

```
var EXTERNAL_NET !$HOME_NET
var DNS_SERVERS [192.168.0.10/32,192.168.0.11/32]
var SMTP_SERVERS 192.168.0.16/32
var HTTP_SERVERS [192.168.0.100/32,192.168.0.101/32]
var SNMP_SERVERS 192.168.0.33/32
var HTTP_PORTS 80
```

Let's run the same pcap back through SNORT to see what kind of effect that made:

```
mysql> select count(sid) as Total_Event from event;
+-------------+
| Total_Event |
+-------------+
|        6713 |
+-------------+
1 row in set (0.00 sec)
```

By just filling in the correct variables for existing rules, the total number of alerts has dropped by 375 from the initial alert count. Let's continue to customize the configuration file to our environment.

There are multiple steps or sections in the *snort.conf* file that break out certain functions and features as well as making it easier to read. They are...

- Setting the network variables
- Configuring dynamic loaded libraries
- Configuring the pre-processors
- Configuring the output plugins
- Configuring SNORT with global config statements
- Customizing your rule set

The next section of the network variables step is the customization of the SNORT decoder. As taken from *snort.conf* (located in '*snort_install_directory*/etc/'), "*SNORT's decoder will alert on lots of things such as header truncation or options of unusual length or infrequently used tcp options*". Depending on certain implementations of an operating system's TCP/IP stack, some of the options may prove useful, but at this time we'll leave it as the default.

The next two sections of step #1 are the configuration of the detection engine itself. The machine that detection engine is running on is a dual P4, 2G of DDR RAM, a 250G HDD, and three Intel Gig NIC's which is a very good sized box for the environment and traffic requirements so there should be no need to configure the detection engine option. If the detection engine were undersized or showed signs of negative performance issues, this configuration option could help with memory requirements and pattern matchers that could assist. For example, one of the options that can be set here is to run the Aho-Corasick (ac), pattern matching algorithm. Another option is lowmem. There is a trade-off in memory, initialization time, and packet processing time when using better performing algorithms (Aho-Corasick) versus more resource conservative algorithms (lowmem).

Step #2 in the configuration script is the ability to configure "dynamic loaded libraries". This is a new functionality added to SNORT in V2.6 for the Dynamic Detection Engine (DDE). The best way to describe what this does is probably best to come from someone who probably knows more about the DDE than anyone else, Andrew Baker. This is taken from the *SNORT IDS and IPS Toolkit*. ***"This engine allows you to create dynamically loaded, shared object rules that are written in C. Shared object rules (also called dynamic detection rules) provide two key benefits to SNORT users. First, and most important, is that shared object rules allow for detection functionality that is significantly more complex than text-based rules. This allows SNORT to be updated rather quickly to detect attacks that are beyond the capabilities of the current rule detection options. Shared object rules are usually much easier to write than new detection options"***.

This dynamic detection engine was compiled for the SLI detection engine.

Pre-processors

Next is the configuration of the pre-processors. This in my opinion, is every bit as important as selecting the correct rule to run. Again, this is not a paper on the inner workings of SNORT and what every pre-processor does and how it can be configured. For information of this nature I will again recommend *SNORT IDS and IPS Toolkit* and the exceptional SNORT Users Manual at http://snort.org/docs/snort_htmanuals/htmanual_2615/. I will however go over some of SLI's pre-processor options and how they pertain to the environment.

Fragmentation is still a big part of some types of IDS evasion techniques. We can work to minimize the effects of fragmentation by the proper configuration of the *frag3* pre-processor. The way *frag3* can best be utilized is by setting a global config. The global config will encompass the *frag3* pre-processor for the detection engine and better tune the pre-processor with *frag3_engine* configurations. Here I list the *frag3* configuration for SLI and the different reassembly options that should be applied to different machines in the environment:

```
pre-processor frag3_global: max_frags 73728 memcap 6MB
pre-processor frag3_engine: policy linux bind_to   /
        [192.168.0.150,192.168.0.127,192.168.0.120,192.168.0.119,192.168.0.96
        ,   /
        192.168.0.87,192.168.0.86,192.168.0.80,192.168.0.78,192.168.0.47,   /
        192.168.0.46,192.168.0.25] detect_anomalies timeout 75
pre-processor frag3_engine: policy first bind_to [192.168.0.32] detect_anomalies
/
        timeout 75
pre-processor frag3_engine: policy windows bind_to 192.168.0.0/24
detect_anomalies    /
        timeout 75
```

By knowing the environment for an organization, an analyst can better configure the detection engine. Two tools that are highly recommended for finding out what operating systems exist in an environment are xprobe2 (http://xprobe.sourceforge.net/) and p0f (http://lcamtuf.coredump.cx/p0f.shtml).

The *frag3_engine* is more specific and can get more granular (than what?) with respect to how fragments are reassembled. Different TCP/IP stacks will handle fragmentation reassembly differently. If a detection engine is reassembling a fragment sequence in the order a Linux server would, but the target of the fragmentation based attack is a Windows box, the attack would be missed. The *frag3_engine* allows the analyst to customize the reassembly of fragmentation so the detection engine reassembles the traffic based on what has been configured for that host. There are 12 Linux boxes on this segment. Those 12 boxes are put into the '*linux*' policy by binding the IP addresses for those Linux boxes to the policy. There is also one OSX box that is of type '*first*'. *For further information on how each operating system is classified in frag3, read the README.frag3 in the SNORT /doc directory.* The final Windows policy encompasses any box in the 192.168.0.0/24 segment that is not previously defined. As SLI is mainly a Windows shop, the Windows policy is used as the catchall. Again, the default timeout is modified from 60 seconds to 75 because there is available horsepower on the engine. While this is not required, because there are resources available on the box and making the configuration as different from default as possible without impacting performance is one of the goals this has been changed.

I like to change the timeouts and memory settings in *snort.conf* where it makes sense and where I have the horsepower on the box to get away from default settings in *snort.conf*. These settings are typically never changed and most users run with them. The different policies have been developed with specific timeouts based upon the timeouts of the Operating Systems that they pertain to. Increasing time or memory options will consume more resources on the sensor and could affect performance. Malicious individuals know this and will craft their evasion techniques so as to not be flagged by the default configuration of SNORT. The downside to these customizations is the cost on the detection engine increases, sometimes dramatically, where more fragments are stored or memory is allocated.

The *stream4* pre-processor is used for stateful inspection and/or stream reassembly. Stateful inspection is the ability of keeping state in TCP sessions. Rules can be applied to a session instead as well as individual packets allowing for better analysis of the traffic and less false positives. The session reassembly portion of *stream4* allows for the reassembly of traffic over certain ports that will be inspected by the detection engine. One thing to be made aware of here is the reassembly of encrypted traffic isn't going to benefit SNORT in any way and will just waste cycles and resources so adding ports 22 (SSH) or 443 (SSL) is going to provide no benefit and will only consume resources.

```
pre-processor stream4: disable_evasion_alerts timeout 45
pre-processor stream4_reassemble: both ports < 21 25 42 53 80 110 111 135
136 137 139 143 445 513 1433 >
```

Here the stream4 pre-processor is reassembling on both server and client sessions to all of the ports listed. I like to capture both sessions for cases where for instatnce a shell may be shovelled back to or from a client. Specifying client or server specifically could miss this type of possibly malicious activirty. There is a default set of ports that this could be listed as but it contains some of those ports that are not in use in the environment at SLI. (What?)

The next change to the *snort.conf* file will be our *http_inspect* default settings. A quick port scan of the network (if you are authorized to do so), reveals an additional port besides the default of port 80 is being used for HTTP traffic. External traffic should not be coming to this port but it will be added to the config file just in case it is added in the future or there is a misconfiguration on the firewall that allows this traffic through.

```
pre-processor http_inspect_server: server default \
    profile all ports { 80 8080 8180 8081 } oversize_dir_length 500
```

To further customize the configuration to better reflect the web servers, the individual server itself can be called out and defined separate from the default settings as is shown in the following example.

```
---  IIS
pre-processor http_inspect_server: server 192.168.0.100 \
ports { 80 8080 8180 8081 } \
flow_depth 0 \
ascii no \
double_decode yes \
non_rfc_char { 0x00 } \
chunk_length 500000 \
oversize_dir_length 500 \
u_encode yes \
bare_byte yes \
iis_unicode yes \
double_decode yes

---  APACHE
pre-processor http_inspect_server: server 192.168.0.101 \
ports { 80 8080 8180 8081  } \
flow_depth 0 \
ascii no \
double_decode yes \
non_rfc_char { 0x00 } \
chunk_length 500000 \
non_strict \
oversize_dir_length 500 \
u_encode yes \
bare_byte yes
```

Next we come to the sfPortscan pre-processor which allows the sensor to detect uncharacteristic activity like a port scan. Modifying a memory setting on the *sfPortscan* pre-processor memcap is bumped from 10000000 to 15000000. This allocates more memory to the *sfPortscan* pre-processor which can assist among other things the slow and low types of scans by being able to hold what the pre-processor defines as possible scans a little longer. Both of these servers are listening on the ports defined for both internal and external connections.

```
pre-processor sfportscan: proto  { all } \
                    memcap { 15000000 } \
                    sense_level { low }
```

Taking a look at the alerts now provides a dramatic drop from the previous result

```
mysql> select count(sid) as Total_Event from event;
+-------------+
| Total_Event |
+-------------+
|        1976 |
+-------------+
1 row in set (0.01 sec)
```

```
mysql> select DISTINCT sig_name as 'Siganture Name', count(*) as Total
FROM event, signature WHERE signature = sig_id AND timestamp>='2007-07-
01 00:00:00' AND timestamp<='2007-07-01 23:59:59' GROUP BY sig_id ORDER
BY Total DESC;
+------------------------------------------------------------+-------+
| Siganture Name                                             | Total |
+------------------------------------------------------------+-------+
| (portscan) Open Port                                       |  1421 |
| WEB-MISC SSLv2 openssl get shared ciphers overflow attempt |   181 |
| (portscan) TCP Portsweep                                   |   122 |
| (http_inspect) DOUBLE DECODING ATTACK                      |    92 |
| (http_inspect) OVERSIZE REQUEST-URI DIRECTORY              |    58 |
| DOS MSDTC attempt                                          |    46 |
| (http_inspect) NON-RFC DEFINED CHAR                        |    38 |
| WEB-MISC robots.txt access                                 |     9 |
| (http_inspect) IIS UNICODE CODEPOINT ENCODING              |     5 |
| (http_inspect) BARE BYTE UNICODE ENCODING                  |     3 |
| (portscan) TCP Portscan                                    |     1 |
+------------------------------------------------------------+-------+
11 rows in set (0.01 sec)
```

Output plugins

This is the only step or portion of *snort.conf* that was configured prior to the tuning process. The detection engine has been configured to use Barnyard (http://www.snort.org/dl/barnyard/) for output to the MySQL database. We won't go into detail here as there are multiple resources online with regards to how to do this.

Config Statements

A majority of runtime configuration options or flags can be set at this step. Most things that you can pass SNORT on the command line can be configured here such as config order, running as a daemon, and which interface(s) will be the sensor interface(s) can all be defined here. For instance, to disable logging you would enter *'config nolog'* as a config statement. For now, this is left blank while it is still being tuned, however, don't underestimate this step.. Once the final config is set, having the command line options set here ensures the analyst doesn't forget or accidentally add something.

Customizing the Rule Set

The rules or signatures SNORT uses are nicely placed into individual files. These are files like smtp.rules for SMTP (Simple Mail Transfer Protocol) related email rules and oracle.rules for rules pertaining to Oracle. The rules should be commented out here if there is a technology or services you know isn't in your environment. There are also rule files commented out by default, like the virus.rules file. One of the reasons these are commented out is: Should the IDS be watching for virus activity? Isn't this the role of a host or gateway based AV solution? The argument could be made for defense-in-depth but again, what type of impact is the detection engine going to take for inspecting this additional traffic? For SLI, the default will be left in place as I (and even the SLI folks), am not sure of everything in the environment.

The *threshold.conf* include statement is also uncommented so if an alert needs to be suppressed or have a threshold values applied, this can be easily accomplished. Thresholds in SNORT are a way to reduce the number of alerts generated and possibly condense false alarms.

Alert Analysis

Now the *snort.conf* is tuned to the point where we can start to delve into the individual alerts and modify our rules, thresholds, or better tune the configuration file. One can typically start with the alert flagged the most and in the mysql output above it is the '(portscan) Open Port' alert.

The Basic Analysis and Security Engine (BASE http://base.secureideas.net/) is used as a front end into the MySQL database as an easier way into the database than querying the MySQL database from the command and tying the output in with the packet that caused the alert. Most of the traffic that generated this alert is internal workstations accessing a corporate partner over a site-to-site VPN tunnel. By going through the alerts file, we come across the following:

```
16:09:16.110877 IP 192.168.0.77.36602 > 10.1.1.15.80: S
2589775356:2589775356
(0) win 4128 <mss 1474>
16:09:16.114875 IP 192.168.0.77.36618 > 10.1.1.15.80: S
339739941:339739941(0
) win 4128 <mss 1474>
```

The sfPortscan pre-processor is identifying legitimate web application traffic as an alert based on the way that this specific application has been written and how sfPortscan is configured to alert. This is further verified by going through the .pcap and inspecting the sessions this host is using to talk to the server. There are multiple solutions for suppressing this alert:

- editing the *threshold.conf* configuration file and either set a threshold on this type of alert.
- or suppress it all together, which in this instance, we don't really want to do.

Editing the sfPortscan pre-processor options makes more sense here as the 10.1.1.15 server is one box that is always the destination or scanned box as SNORT sees it. The web server will be servicing multiple requests such as the one identified above so we will ignore this alert by adding the ignore_scanned option with 10.1.1.15 as the variable

```
pre-processor sfportscan: proto  { all } \
                          memcap { 15000000 } \
                          ignore_scanned { 10.1.1.15 } \
                          sense_level { low }
```

Let's take another look now specifically at this alert

| *(portscan) Open Port* | *161* |

In looking at the source and destination for these remaining alerts, they all source from eight internal IP's and are destined outside the environment. The traffic from the pcap could be analyzed or these workstations can be looked at later by SLI to see whether it could be malicious applications, P2P, Skype, valid web application traffic are some examples that might generate this alert but we will move on to other alerts.

The next highest alert count is the 'WEB-MISC SSLv2 openssl get shared ciphers overflow attempt ' at 181 alerts against an external facing web server.

[**] [1:8428:2] WEB-MISC SSLv2 openssl get shared ciphers overflow attempt [**]

```
[Classification: Attempted Administrator Privilege Gain] [Priority: 1]
07/01-16:09:05.951063 67.169.168.66:61373 -> 192.168.0.200:443
TCP TTL:113 TOS:0x0 ID:3289 IpLen:20 DgmLen:1000 DF
***AP*** Seq: 0x2114EEA  Ack: 0x83B1F85C  Win: 0xFDF7  TcpLen: 20
[Xref => http://www.openssl.org/news/secadv_20060928.txt][Xref =>
http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-3738][Xref =>
http://www.securityfocus.com/bid/20249]
```

In looking at rule 8428 and the traffic that generated this alert, it is discovered that the destination IP should have been included in our HTTP_SERVERS definition in the *snort.conf*. It turns out there is an additional external facing web server that was not included during the initial configuration. After adding this server, we take another look at the results by running the pcap back through the SNORT.

```
+----------------------------------------------------------------------+-------+
| Siganture Name                                                       | Total |
+----------------------------------------------------------------------+-------+
| (portscan) Open Port                                                 |   161 |
| (http_inspect) DOUBLE DECODING ATTACK                                |    92 |
| (http_inspect) OVERSIZE REQUEST-URI DIRECTORY                        |    58 |
| DOS MSDTC attempt                                                    |    46 |
| (http_inspect) NON-RFC DEFINED CHAR                                  |    38 |
| (portscan) TCP Portsweep                                             |    17 |
| WEB-MISC robots.txt access                                           |     9 |
| (http_inspect) IIS UNICODE CODEPOINT ENCODING                        |     5 |
| (http_inspect) BARE BYTE UNICODE ENCODING                            |     3 |
| (portscan) TCP Portscan                                              |     1 |
+----------------------------------------------------------------------+-------+
10 rows in set (0.01 sec)
```

The *http_inspect* pre-processor is flagging five of the remaining 10 alerts. First, we'll take a look at the '(http_inspect) DOUBLE DECODING ATTACK' alert. The source of these alerts are all on the SLI internal network. By analyzing the traffic and verifying with the SLI developers that the items being defined as double decode by SNORT are indeed valid as part of a web session, we can either suppress or set thresholds on this traffic

Using BASE, we gather the generator ID (identifier in SNORT that indicates which subsystem generated the alert) and the signature ID (identifier in SNORT that indicates the type of alert that was generated) so we can properly identify and suppress this alert in the *threshold.conf* which in this case are the http_inspect alerts. This file allows us to set thresholds on alerts based on quantity and time or we can suppress the alerts altogether by source or destination. This is the format that I used for the *http_inspect* pre-processor alerts119:2, 119:4, 119:7, 119:14, 119:15, and also the pre-processor alerts that we have found through analysis to be false positives 122:1, 122:3, and 122:27.

GEN:SID 119:2 Message http_inspect: Double Decode Attack

suppress gen_id 119, sig_id 2, track by_src, ip 192.168.0.0/24

### This alert has been acknowledged as an acceptable risk by SLI management due to other processes in the environment and not wanting to investigate these due to noise. -BG *Date*

Along with making these changes, it is always a good idea to comment any changes in the *snort.conf*. The comment should be able explain when a question arises about why the suppression, configuration, or rule change is there.

(SLI made a conscious effort on their part to accept the risk that any of their workstations actively involved in any of the above *http_inspect* and *sfportscan* alerts could be compromised in one way or another. SLI decided to rely on antivirus, host based IPS, and some NSM techniques.)

The next highest alert on the list is the DOS MSDTC SNORT ID 1408. The first thing to look at is the signature generating the alert. As the SNORT VRT rules are not able to be published, in order to view this signature you could download the rules and look at this specific rule to see what this specific rule is looking for.

The alert is being generated from an external source to one internal workstation. A quick scan against the workstation does not show a listener on the tcp:3372. Next, the pcap is checked to see what exactly is going on here.

```
[brandon@SLISnort]$ sudo tcpdump -r corp_in2.pcap -nn 'port 3372' | more
reading from file corp_in2.pcap, link-type EN10MB (Ethernet)
16:10:02.476162 IP 172.16.5.95.8001 > 10.2.124.56.3372: P
1927975462:1927975475(13) ack 2608862694 win 64224
16:10:02.476537 IP 172.16.5.95.8001 > 10.2.124.56.3372: . 13:1393(1380) ack 1 win
64224
16:10:02.476661 IP 172.16.5.95.8001 > 10.2.124.56.3372: . 1393:2773(1380) ack 1
win 64224
16:10:02.476786 IP 172.16.5.95.8001 > 10.2.124.56.3372: . 2773:4153(1380) ack 1
win 64224
16:10:02.476789 IP 10.2.124.56.3372 > 172.16.5.95.8001: . ack 1393 win 32768
16:10:02.476911 IP 172.16.5.95.8001 > 10.2.124.56.3372: . 4153:5533(1380) ack 1
win 64224
16:10:02.477036 IP 172.16.5.95.8001 > 10.2.124.56.3372: P 5533:6660(1127) ack 1
win 64224
16:10:02.477039 IP 10.2.124.56.3372 > 172.16.5.95.8001: . ack 4153 win 32768
…
```

What has happened is the SNORT process was started in the middle of an already established session to a corporate partner. This session just so happened to also be using an ephemeral port of 3372 and have datagram sizes larger than 1023 (1380 to be exact). These two traffic characteristics and the fact that SNORT did not see the initial session start up or three way handshake initiated by the internal client caused this rule to alert. If we wanted to clear this alert out temporarily so we could work towards no false positives for this pcap, the rule can be commented out temporarily or a local rule can be created specific to this traffic. The quickest way to temporarily turn off the alert is by commenting it out with the pound (#) symbol. A note must be made however to turn it back on when finished looking at the pcap.

Next to be addressed is the robots.txt access. Typically, various search engines or spidering applications will crawl a site and make requests for a file called robots.txt (http://www.robotstxt.org/wc/robots.html). This file is typically placed on a site as a reference point for exclusion or inclusion so these spidering or crawling applications will not index or will index what is placed in the file. The robots.txt file is also a good source of information an attacker may wish to explore which is one of the reasons for the alert.

SLI does not use a robots.txt file on any of their servers nor do they wish to in the foreseeable future. Given that information, this rule will be commented out along with a commenting the rule so that anyone who might come along to work on the sensor will know why the rule was disabled, when it was disabled and who did it.

```
mysql> select count(sid) as Total_Event from event;
+-------------+
| Total_Event |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)
```

```
mysql> select DISTINCT sig_name as 'Signature Name', count(*) sig_id  FROM
event, signature WHERE signature = sig_id AND timestamp>='2007-07-01
00:00:00' AND timestamp<='2007-07-01 23:59:59'  GROUP BY sig_id ORDER
BY sig_id DESC;
Empty set, 1 warning (0.00 sec)
```

Now that there are no alerts left for this packet capture, the detection engine will be turned up. The hope is that most of the false positives and accepted risk alerts have been suppressed allowing the analyst to continue to tune alerts without being inundated with hundreds of thousands of alerts an analyst would need to go through.

Running Production

First things first, the commented-out MSDTC alert for the pcap is uncommented. After letting the detection engine run for ~34 hours, we take a look at the alerts generated to this point.

```
mysql> select count(sid) as Total_Event from event;
+----------------+
| Total_Event |
+----------------+
|              336 |
+----------------+
1 row in set (0.00 sec)
```

```
mysql> select DISTINCT sig_name as 'Signature Name', count(*) sig_id  FROM event,

signature WHERE signature = sig_id AND timestamp>='2007-07-04 00:00:00' AND

timestamp<='2007-07-7 23:59:59' GROUP BY sig_id ORDER BY sig_id DESC;

+-----------------------------------------------------------------------------------------------+--------+
| Signature Name                                                                                  |
sig_id |
+-----------------------------------------------------------------------------------------------+--------+
| (portscan) Open Port                                                                             |
79 |
| (http_inspect) OVERSIZE CHUNK ENCODING                                         |     48
|
| ICMP Destination Unreachable Communication with Destination Host is
Administratively Prohibited |     38 |
| (http_inspect) WEBROOT DIRECTORY TRAVERSAL                                    |     35 |
| (ftp_telnet) FTP traffic encrypted                                                   |     24 |
| (http_inspect) U ENCODING                                                            |     20 |
| (portscan) TCP Portscan                                                                |     16 |
| (portscan) TCP Portsweep                                                               |     14 |
| WEB-CLIENT Outlook EML access                                                     |     11 |
| MISC MS Terminal server request                                                   |      8 |
| SMTP MAIL FROM overflow attempt                                                  |      8 |
| WEB-CLIENT Microsoft wmf metafile access                                       |      5 |
| (ftp_telnet) FTP command parameters were malformed                        |      5 |
| WEB-CLIENT ShockwaveFlash.ShockwaveFlash ActiveX CLSID access       |      5 |
| WEB-CLIENT Mozilla bitmap width integer overflow multipacket attempt    |      3 |
| MISC MS Terminal Server no encryption session initiation attempt          |      3 |
| WEB-MISC login.htm access                                                           |      2 |
| (http_inspect) OVERSIZE REQUEST-URI DIRECTORY                               |      2 |
| (ftp_telnet) Invalid FTP Command                                                   |      2 |
| (portscan) TCP Distributed Portscan                                        |      1 |
| WEB-MISC SSLv2 Client_Hello with pad Challenge Length overflow attempt     |
1 |
| WEB-CGI scriptalias access                                                 |      1 |
| WEB-MISC Chunked-Encoding transfer attempt                                 |      1 |
| WEB-FRONTPAGE rad fp30reg.dll access                                       |      1 |
| WEB-FRONTPAGE /_vti_bin/ access                                            |      1 |
| (http_inspect) BARE BYTE UNICODE ENCODING                                  |      1 |
| (http_inspect) NON-RFC DEFINED CHAR                                        |      1 |
+---------------------------------------------------------------------------------------------+--------+
27 rows in set, 1 warning (0.00 sec)
```

There are now 336 individual alerts and 27 alert types in a 72 hour period. This is a lot less daunting than the initial 180,590 individual alerts and 386 alert types in just 24 hours before tuning.

In just a quick perusal through BASE at these new alerts, it looks like a majority of them are from corporate site-to-site VPN tunnels. Sometimes alerts from ‘partners’ over a VPN can be just as malicious as traffic in from the Internet so I am sure the SLI team will be spending a lot of time looking through these alerts.

Conclusion

Deploying an IDS/IPS and tuning it to a point where the alerts generated are relevant is not an easy or quick project. False positives may plague an analyst for days or weeks to come but the only thing worse than a false positive is a false negative. Knowledge of the environment is absolutely critical to save the analyst time in the long run and being comfortable with how the detection engines are set up. The proper time and resources must be allocated to maximize any return on investment an IDS/IPS provides.

The *snort.conf, threshold.conf*, and rules directory defined above can not be moved to another environment by just changing some IP’s and expect it to come up and run flawlessly. Every implementation is different. Configurations and settings will be different for every installation, but the process above is universal. While the above outline for tuning an IDS/IPS is specific to SNORT the principles and methods outlined will work for most any solution. These same and similar processes can and has been used countless times in countless organizations regardless of vendor or product.

References

Bejtlich, Richard. Extrusion Detection – Security Monitoring for Internal Intrusions. USA: Addison Wesley, 2006.

Kohlenberg, Toby, et al. Snort IDS and IPS Toolkit. USA: Syngress. 2007

The Snort Project. (2006). Snort™ Users Manual 2.6.1, from http://www.snort.org/ /docs/snort_htmanuals/htmanual_2615/