Intrusion Detection with the Snort IDS

SAFETY SNORT

Search out hidden attacks with the Snort intrusion detection system.

BY CHRIS RILEY

Recently, I implemented an Intrusion Detection System (IDS) for a remotely hosted web farm. After the initial setup, I began testing and configuring to streamline the system. As soon as the system was switched on, I noticed the sort of traffic that shouldn't be internal to a DMZ. The ISP-controlled firewall had been misconfigured to allow almost all traffic.

In the short time the test was running, the IDS logged a large number of port scans and access attempts on the main servers. From these logs, it was obvious that the servers were receiving the wrong sort of attention.

The moral of the story is to keep an eye on your network. Even if you don't have a misconfigured firewall, your systems could benefit from the attention of an IDS. At the most basic level, an IDS captures network traffic. Then it compares the contents of these packets against specific rules to check for known vulnerabilities or malicious code.

When the IDS discovers something that matches a rule, it triggers a pre-configured action. The action varies depending on the configuration, however, in basic IDS mode, the system simply logs the offending traffic or sends an alert. An IDS sensor on the network perimeter keeps a watchful eye on traffic that the firewall lets through; a sensor located outside of the firewall lets you watch access attempts.

Snort [1] is an open source IDS alternative. As with many open source projects over the past few years, Snort now has a corporate arm, Sourcefire [2], but the good news is that Snort is still freely available under the GPL.

In this article, I describe how to start watching your network with Snort.

Installation

Snort is usually easy to install. Those with *.deb* or *.rpm* package managers should find Snort in the list of available software for your distros; however, it might be an older version.

At this time of writing, the latest version is 2.8.0.2. Installing from source isn't as easy as setting up Snort with *apt-get*, but you'll have many more options for customizing your system. To build Snort, you need to download the source

pcphotos, Fotolia

tarball and, optionally, the MD5 hash to run a file check.

```
wget http://www.snort.org/dl/current ➥
/snort-2.8.0.2.tar.gz
Optional:
wget http://www.snort.org/dl/current ➥
/snort-2.8.0.2.tar.gz.md5
Optional:
md5sum -c snort-2.8.0.2.tar.gz.md5
tar -xvf snort-2.8.0.2.tar.gz
cd snort-2.8.0.2
```

When the source is uncompressed and at your mercy, figure out where you want to store the logs and alerts. Also, you can always stick with the simple option of logging all output to */var/log/snort/*, or you can go the more flexible and scalable route. Snort supports a wide range of databases that allow you to centralize the data easily. The choice depends on what you want to achieve and how much traffic you expect to handle. The general rule of thumb is to take the estimated traffic level and multiply it by 10. Most of the time, the sheer amount of traffic takes people by surprise and can easily overload your logging system if you're not prepared. This example installs MySQL as the database back end. If you want to run another database, you can build in support through command switches available in *./configure*. To view a full list of the supported options, run *./configure -h*.

```
./configure --with-mysql
make
sudo make install
```

If you encounter errors running the build process, you might be missing some required headers. In particular, you need to ensure you have *pcre.h*, *pcap.h*, *pcap-bpf.h*, and *mysql.h* present in the */usr/include* directory. If these files are missing, some of the dependencies might not be installed properly (Listings 1 and 2). It's also possible to have issues with the *libpcap.so* file. On certain distros, you need to recreate this symbolic link by running *ln -s /usr/lib/libpcap.so.* < *version* > */usr/lib/libpcap.so*. Once you've run *make* and *sudo make install* cleanly, you're ready to do some final touches before heading into the MySQL database configuration. Before moving on to the database, create a new user for

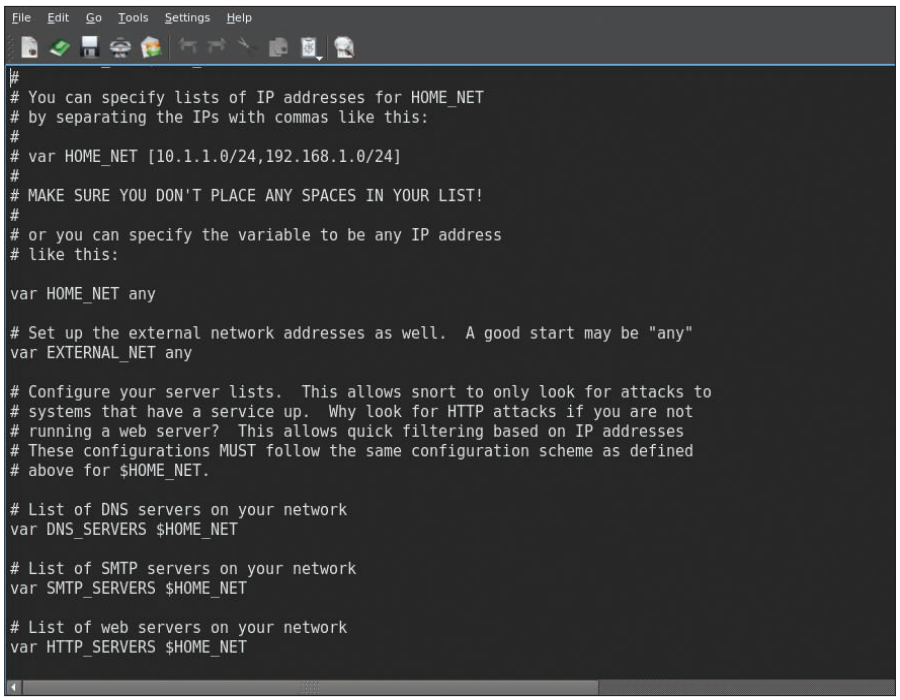

Figure 1: The snort.conf is a central point for Snort configuration.

Snort. (After all, you don't want the service running as root.) To create a new user, run the following commands:

```
groupadd snortgrp
useradd -g snortgrp snortusr
```

These commands will create the *snortgrp* group and the new user *snortusr*. Before continuing, make sure you are in the directory where you untarred Snort. The following commands create the required directories for the Snort configuration, rules, and logfiles, copying the required files to the newly created */etc/snort* directory.

```
mkdir -p /etc/snort/rules
mkdir /var/log/snort
touch /var/log/snort/snort.log
touch /var/log/snort/alert
chown -R snortusr.snortgrp ➥
/var/log/snort
cp etc/* /etc/snort/
```

To download the latest rules, connect to the Snort website [1] and register. The website offers both free and paid registration options, depending on how up to date you need your rule set. Members who have a paid subscription have access to newly updated rules 30 days before the normal registered users. It is better not to use the rules provided at version release, as these rules are quickly outdated against new attacks. Once you have registered (or subscribed), you can download the new rule set and extract it into */etc/snort/rules*. Don't forget to use md5sum to check the tar.

Preparing the Database

Now that the basic system is installed, it's time to prep the database. Once the MySQL server is running (run a *ps -A | grep mysqld* to check), the configuration can begin.

The database configuration is divided into separate steps. First, you will need to set an appropriate password, create the required database, and define the table structure. Connect to the MySQL service as root and create the database and permissions for *snortusr*. To open the MySQL command processor, run *mysql -u root -p* at the prompt. Then, you will be prompted for the root user password and given a *mysql >* prompt. Enter the following commands to complete the first phase of the setup. Because these are database commands, you need to make sure each line is terminated with a ";" character.

When creating the passwords for the user, make sure to use passwords that will stand a better chance in a brute force attack. A minimum of eight characters is suggested – with uppercase letters, lowercase letters, numbers, and special characters.

```
# database: log to a variety of databases
# ------------------------------------
# See the README.database file for more information about configuring
# and using this plugin.
#

output database: log, mysql, user=snortusr password=desired_$n0rt_pa$$word dbname=snort host=localhost

# output database: alert, postgresql, user=snort dbname=snort
# output database: log, odbc, user=snort dbname=snort
# output database: log, mssql, dbname=snort user=snort password=test
# output database: log, oracle, dbname=snort user=snort password=test
```

Figure 2: Configuring MySQL settings in snort.conf.

```
create database snort;
grant INSERT, SELECT on root.* to ➥
snort@localhost;
set PASSWORD for snort@ ➥
localhost=PASSWORD ➥
('Desired_$n0rt_pa55word');
grant CREATE, INSERT, SELECT, ➥
DELETE, UPDATE ➥
on snort.* to snort@localhost;
grant CREATE, INSERT, SELECT, ➥
DELETE, UPDATE on ➥
snort.* to snort;
exit
```

Each command should return a *Query OK* response. If not, make sure you have terminated with a *";"* character. The second stage of the installation process includes a simple script that is passed to the MySQL command processor. Make sure you are in the directory where you untarred Snort, then run the following command:

```
mysql -u root -p schemas ➥
/create_mysql snort
```

Now that both steps are complete, make sure all the pieces are in place. To confirm everything is ready, enter the MySQL command processor as the Snort user created earlier. Check the database and table structure with the following commands:

```
show databases;
use snort;
show tables;
exit
```

Once the database is ready, you're ready to begin configuring Snort.

Snort Configuration

Once the preparations are finished, it is time to dive into the Snort configuration. The main Snort configuration file, *snort.conf* (Figure 1), is located in the */etc/snort* directory. Bring this file up in the editor of your choice and take a quick look at the various sections. The *conf* file contains a wealth of helpful hints about configuring your IDS. For example, you'll need to add information about the network and servers so that Snort can map rules correctly. To ensure the system monitors the correct traffic, the *var HOME_NET* and *var EXTERNAL_NET* need to be configured to reflect the network infrastructure. On a simple network, *HOME_NET* will probably be set to a private IP range, like 192.168.0.0/24. This means that all traffic originating from the 192.168.0.1-255 IP range will be classified as internal traffic. The details will vary, depending on your configuration. If you have multiple subnets on your internal network, you can add each subnet with a comma between them. The *EXTERNAL_NET* entry is a list of specific external addresses to classify as external. The simplest method to configure this is to use the *!$HOME_NET* setting, which will be translated by Snort as all addresses except *HOME_NET*.

Specify the location of the Snort rules through the configuration. If you download the rules to */etc/snort/rules*, add this path to the *var RULE_PATH* line. The last, and important, variable to set is used by the IDS system to log data to the database. Near the bottom of *snort.conf* is a section for configuring the output plug-ins. Here the *output database: log, mysql ~* line must be uncommented and replaced with the location of the MySQL database (Figure 2).

Once the settings are completed, you have the basis of a working server. However, you're going to need to set up Snort to run on start-up and ensure it is running under the newly created *snortusr* account. At this point, you can test Snort from the command line using *snort -u snortusr -g snortgrp -c /etc/snort/snort.conf*. Snort will run under the supplied credentials and begin to log or alert on any captured traffic. When terminated, Snort displays session statistics (Figure 3). This screen report is nice, but it is not a perfect solution.

To get Snort to start on boot, insert a simple shell script into the */etc/init.d* folder. To create this script, open your favorite editor and enter the following lines:

```
#!/bin/bash
#
# Snort startup script - ➥
/etc/init.d/snortstart
#
/usr/local/bin/snort -Dq -u ➥
snortusr -g snortgrp -c ➥
/etc/snort/snort.conf
```

With this script in place, run *chmod +x /etc/init.d/snortstart* to make it executable and *update-rc.d /etc/init.d/snortstart defaults 95* to enter the necessary symbolic links into the required directories. This process might be slightly different depending on your Linux distro.

Snort Rules

Snort provides an array of rules for filtering out unwanted traffic. Snort rules are

Figure 3: Snort displays session stats at termination.

mostly easy to understand and customize. Each rule is separated into two sections: the rule header and the rule options. The header describes what message to display when the rule is triggered. The option contain keywords that tell Snort how to inspect the packet and references for research and information on the message to be displayed if an alert is triggered.

In the following rule:

```
alert tcp $EXTERNAL_NET any
->$HTTP_SERVERS
 $HTTP_PORTS(msg:"WEB-IIS
unicode directorytraversal
 attempt"; flow:to_
server,established;content:"/
..%c1%1c../"; nocase;
reference:cve,2000-0884;
reference:nessus,10537;
classtype:web-application-attack;
sid:982; rev:13;)
```

The rule header consists of the command *alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS*. In brief, this header tells Snort to alert when the rule is triggered and to examine only traffic coming from external networks (any source port) to internal http servers (on configured http ports). Although the alert statement at the beginning might seem obvious, at times you might only want to log the traffic into the database, or even do some more advanced stuff with *dynamic* and *activate* actions. If you're using Snort as an in-line IPS, you can also use the *Drop*, *Reject*, or *Sdrop* options to manage unwanted traffic. Snort can check TCP, UDP, IP, or ICMP packets, depending on your requirements. If the rule specifies TCP, and a UDP packet comes in, even if the rest of the rule header and options match perfectly, Snort will not perform any action. This rule specifies TCP, which is pretty standard for http traffic.

The next part of the header calls on some variables that are already configured into the Snort configuration file. The rule will examine traffic arriving from the *$EXTERNAL_NET* variable you set in the *snort.conf* any source port number. The -> denotes the direction of traffic. In this case, the rule applies to anything coming from *$EXTERNAL_NET* to the *$HTTP_SERVERS* on *$HTTP_PORTS*. These variables are defined in the Snort configuration file.

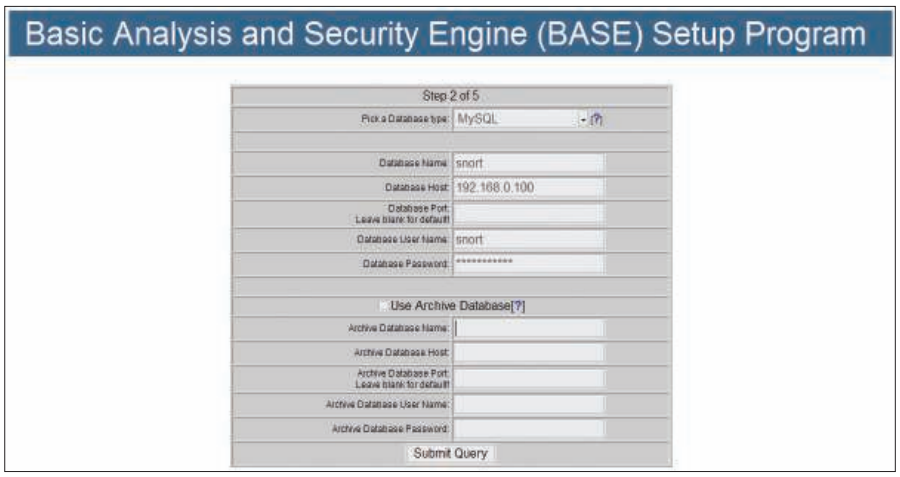

Figure 4: Setting up access to MySQL through the BASE configuration page.

The direction of traffic is very important. In this case, any replies coming back from the *$HTTP_SERVERS* will be ignored since they won't match the direction of the rule.

The remainder of the example forms the rule options. The options section starts by telling Snort what message to display in the alert. In this case, the rule tells Snort to output *"WEB-IIS unicode directory traversal attempt"* into the log/database and alert. Following this command is the most important part of the rule options: the part dealing with matching desired traffic. The *flow* tag tells Snort to only examine packets sent to the target server once a session is established. This requirement prevents Snort from examining the SYN, SYN-ACK, ACK 3-way handshake that initializes the connection. On a busy IDS, eliminating the handshake traffic from the rule check could significantly improve the performance of the system.

The *content* section is where the real meat of the rule is formed. In simple terms, Snort will take the value specified in the content tag and compare it against requests sent to the server. The preceding example rule is searching for the string *"/..%c1%1c../"*. This string uses Unicode to hide a directory traversal attempt on a web server. Most systems are now immune from this kind of attack, but you'll still find a number of attempts to exploit this vulnerability on well-traveled sites. The *nocase* command following the *content* tag tells the rule to ignore case when matching the contents. The final tag to consider is the *classtype* tag. The information within this tag tells Snort how severe the event is. In this case, the *web-application-attack* class type matches a high priority level. These levels are better explored and configured in the *classifications.config* file.

To fine tune the list of active rules, take a look into */etc/snort/snort.conf* again and examine the list of rules near the bottom. The default configuration enables a range of rules that provide a general overview of traffic coming into the network.

To trim the alerts to a manageable level, and to ensure you're monitoring the correct services, you can modify the list of rules. Creating a more focused list will reduce packet loss and ensure better performance. The rules you leave active are dependent on your network infrastructure and overall requirements. If your network doesn't use specific services or protocols, disable any unnecessary rules to reduce overhead.

By default, a number of rules are disabled. Many of the disabled rules occasionally cause false positives, but you might want to enable some of them to use for specific purposes. Once you have tailored the list to your requirements, take some time to provide information on specific services.

As you have already seen, Snort offers built-in variables to simplify the task of configuring rules. Without variables, if your company had a number of http servers listening on a port other than the standard port 80 (port 8080 for example), you would have to edit every Snort rule to alter the http port from 80 to 8080. Not only that, but you would have to change every rule with every update of the rule set. Instead, you can use

Snort's built-in variables to set the value of *$HTTP_PORTS* to 8080 instead of the standard 80. Then you can run the servers on whatever ports you want without having to always edit the rules to match the environment. To change the value of *HTTP_PORTS* to *8080*, edit the *snort.conf* file as follows:

```
var HTTP_SERVERS ➥
[10.10.10.100/32,10.10.10.111/32]
var HTTP_PORTS [80,8080]
```

If you want to specify a range of ports instead of a long list (i.e., 8000 through to 8080), use a colon (*8000:8080*).

snort.conf includes built-in variables for HTTP, AIM, and Oracle services. Also, you can add your own variables for other services if you plan to reference them in custom rules. Providing Snort with information on where and how the services on your network are configured lets the IDS reduce overhead by restricting traffic checks to a limited range of destinations. Why check SMTP traffic destined for a system that only supports SSH or FTP services? On large networks, you can never be 100% sure what traffic is passing over the wires. You might discover an SMTP service is running on an obscure server that's been in the company since the dawn of time.

Snort also lets you create custom rules. The best way to learn about creating rules is to look at an existing rule. Once you have examined a few of the rules, you can select one that most closely matches your requirements and play around with it. It is common to add these custom rules to the *local.rules* file for testing purposes. Adding your custom rules to *local.rules* also protects them from being overwritten when you update to a new rule set.

Figure 5: Old school ASCII art: See the Snort pig mascot on the left.

Custom rules come in handy in situations when the patch for a known vulnerability isn't available yet. Adding a custom rule to your IDS/IPS can give you an added layer of protection, or at least an early warning system if the problem appears.

Snort Preprocessors

Preprocessors, which you can enable and disable through *snort.conf*, let Snort manipulate inbound traffic. Snort automatically enables a number of preprocessors to deal with fragmented traffic, stateful stream inspection, performance monitoring, RPC traffic decoding, ftp/telnet/SMTP/DNS/SMB traffic monitoring, and port scanning. You'll even find a preprocessor especially designed for the Back Orifice trojan. Each preprocessor has its own set of options and settings. The defaults should be fine as a starting point, but spend some time with the preprocessor configuration if you want to get the most out of your IDS. In particular, the sfPortscan preprocessor can create false positives if set incorrectly. If you begin to receive false positives, you can easily disable sfPortscan through *snort.conf*.

If your server is low on RAM, you might need to tweak some of the preprocessor memory settings. As an example, the frag3 preprocessor uses 64MB of RAM as a default for storing and reassembling fragmented traffic. Although 64MB doesn't seem like much on today's servers, you can see how the addition of several preprocessors could start to make a dent in server performance. Inversely, if you have more than enough RAM, you can increase the available memory to ensure fragmented traffic doesn't become a problem in high-load environments.

Logs and Alerts

Your IDS system is ticking away, logging traffic, and alerting a MySQL database. Having a database full of alerts and logged traffic is a great thing. However, receiving an alert at your desktop when somebody is port scanning your systems is even better.

Unfortunately, Snort doesn't offer a built-in solution for delivering alerts to a remote desktop. As with many *nix projects, however, Snort is easy to interface with other utilities. Two possible candidates are Swatch and Logsurfer.

Other products are available on the front end to display your Snort data as nice graphs and statistics. One of the more popular systems is BASE (Basic Analysis and Security Engine) [3]. Download BASE version 1.3.9 from the project website. To get started with BASE, you'll need to set up Apache and PHP on your server. BASE also relies on ADOdb [4] to provide access to the database through PHP. For security and performance reasons, it is a good idea to set this up on a system separate from the Snort sensor system.

A management console is no good if you can't get access to it while the IDS is busy dealing with other traffic. Sometimes having a second NIC in the Snort sensor especially for monitoring and management is a good idea. Once you have the Apache, PHP, and ADOdb packages installed, you will need to untar the BASE code into */var/www/base*. For the time being, change the permissions on the */var/www/base* directory to make it world writable (*chmod 777*). This practice is terrible for security, but you'll only need this capability for the duration

Spec'ing the Server

If you are planning a Snort server, it is important to consider the traffic you're expecting to handle. Snort will run on almost any kind of hardware. However if you want a fast, reliable IDS without high packet loss, Snort needs a reasonably fast processor. Needless to say, you'll also need storage space for the logs and alerts. The most critical thing, however, is a good NIC. Where possible, ensure the NIC is a separate card and not built into the motherboard. Most major NIC manufacturers now offer a server class network card with an onboard processor especially designed for network traffic processing.

Listing 1: Snort Dependencies

- Libpcap
- Libpcap-dev
- PCRE
- PCRE-dev
- Libnet-1.0.2.a
- MySQL-Server-5.0 (for MySQL support)
- MySQL-Client (for MySQL support)
- MySQL-dev (for MySQL support)

of the configuration process. You can then go to the BASE web page, *http://Your_Server/base*, to configure access to the database (Figure 4). You'll need to enter the path to ADOdb files, as well as your MySQL server name, logon, and password into the web form.

If the BASE web page says the configuration files aren't writable, check the *chmod* you just performed. BASE will add content to the MySQL database for reporting, and once completed, the setup is finished. If you experience problems, you might need to uncomment the *mysql.so* extension in your *php.ini* file. Don't forget: You will need to reset the permissions on the */var/www/base* directory to something readable by your Apache server. It is important to note that BASE doesn't provide any built-in security for the web front end. So if possible, enabling SSL and ensuring that there's an *.htpasswd* on the BASE directory is a step in the right direction.

Aside from the database, you will also find text logs and alerts in */var/log/snort*. These logfiles contain the complete log data in *tcpdump* format. Should you want, you can easily write a script to inform you when a new alert is logged. To work with these files, use *snort -r* to process the *tcpdump* file into something easier to read. The *-vd* switches provide verbose information. To make things a little easier, Snort also supports the use of BPF (Berkeley Packet Filter) [5] to filter output from the command line.

```
snort -vd -r snort.log.1206804587 ➥
tcp and src port 22
snort -vd -r snort.log.1206804587 ➥
not host 192.168.0.1
```

What is MD5?

MD5 is a cryptographic hash function that provides a 128-bit hash based on the contents of file. When downloading a program or document, you can use the *md5sum* command to ensure the download you have is the same as the original. Md5sum compares the hash value of the download to an MD5 hash of the trusted version. Many software projects now provide an MD5 hash of binaries. The MD5 is usually found on the project website in the download section. Running this check against your download can help you avoid installing corrupted or malicious software.

Listing 2: BASE Dependencies

- apache(-ssl)
- php5
- php5-mysql
- php5-gd
- libphp-adodb

Prevention or Detection

Snort provides several options for *preventing* (and detecting) intrusion. The three main modes for preventing intrusion are inline filtering, cooperation with an existing iptables-based firewall, and TCP-RST mode.

When Snort is working as an inline filter, all traffic must pass through the Snort system before it passes to the internal network. If the traffic triggers a rule in the Snort system, the packets are dropped. The inline solution offers advanced firewall-style security with a regularly updated rule set. However, the IPS can also prevent access to systems through false positives and will slow down your network if you have more traffic than the Snort sensor to handle. For inline mode, you'll need to add *--enable-inline* to your *./configure* command.

If you already have an existing iptables-based firewall, you can configure Snort to provide dynamic rule changes. The iptables option reduces some of the lag on inbound traffic, but as a trade-off, your system will be slower to respond to attacks. Once the malicious traffic triggers an alert, Snort sends a command to the iptables system to block the attacker. This style of IPS, if not correctly configured, can be manipulated by a creative attacker to force a denial of service on your own systems.

If an attacker spoofs malicious traffic from your ISP's router or DNS server, you could end up blacklisting services you need to maintain a reliable network presence. To combat this, use a whitelist of addresses you never ban. However, an attacker who discovers the address of your whitelist can spoof attacks from this address without fear of being blocked.

The final option is to allow Snort to disconnect unwanted connections through the use of TCP-RST packets (through the use of the *flexresp2* patch). This option can terminate an unwanted connection from both ends. However, this solution causes a race condition between your IPS and the malicious traffic. The IPS attempts to close the connection before the attacker can complete the attack. The attacker will already have an advantage in this case, because the malicious traffic is already inside your network before Snort can act. This mode of operation helps prevent certain attacks, but it might be less reliable than the other techniques.

How you configure your IDS/IPS is dependent on your security requirements. If you intend to set up Snort as an IPS, test the server in IDS mode until you've correctly tuned the configuration and reduced false positives.

Once you're happy with the configuration, move Snort to its new role as a prevention system.

Conclusion

Snort has many other features to discover. For example, I never got to mention the retro ASCII art pig (Figure 5).

Numerous books and online resources will help you get started with the Snort intrusion detection system. The Snort project website offers a great number documents that can help solve problems. Snort's website also offer a community forum that provides user assistance and news. ■

INFO

[1] Snort homepage: *http://www.snort.org*

[2] Sourcefire: *http://www.sourcefire.com*

[3] BASE: Basic Analysis and Security Engine: *http://base.secureideas.net*

[4] ADOdb database abstraction library for PHP: *http://adodb.sourceforge.net*

[5] BPF (Berkeley Packet Filter): *http://tcpdump.org*

THE AUTHOR

Chris Riley is an IT Security Analyst living and working in Austria. After 12 years working as a server administrator in the UK and Germany, he is now spending a majority of his time performing penetration tests in and around Vienna.