HAKIN9

PRACTICAL PROTECTION

IT SECURITY MAGAZINE

starterkit

Vol.2 No.1 Issue 1/2010(4) ISSN 1896-9801

SNORT® EXPOSED

Adobe® PDF
Magazine Version

ARE YOU GETTING THE MOST OUT OF YOUR IPS?

COLLECTION AND EXPLORATION OF LARGE DATA

AN UNSUPERVISED IDS FALSE ALARM REDUCTION SYSTEM –SMART

WRITING SNORT RULES

NOTES OF THE NETWORK ADMINISTRATOR

DEPLOYING SNORT AS WAF

IMPROVING YOUR CUSTOM SNORT RULES

CONTENT MODIFIERS: KEEP IT SPECIFIC

Publisher: Software Press Sp. z o.o. SK
02-682 Warszawa, ul. Bokserska 1
Phone: 1 917 338 3631
www.hakin9.org/en

Dear Readers,

As you already know Snort is the most widely deployed IDS/IPS technology worldwide. Developed by Sourcefire, Snort combines the benefits of signature, protocol, and anomaly – based inspection.

In Snort Special Issue Leon Ward, Joel Elser, Kishin Fatnani, Shivang Bhagat and Rishita Anubhai provide insight into writing Snort rules and into deployment of this IDS/IPS.

With the end of the year inevitably approaching, it's high time to briefly reflect on 2010 and enter 2011 with new solutions and ideas for the foreseeable future.

Some of them are provided by KK Mookhey in "How to get the most out of your IPS?" And annual Conference on Nagios and OSS Monitoring is to be looked forward too.

Wishing you wonderful Christmas,
Hakin9 Team

TOOLS

4 Uptime IT Systems Management Review
by Michael Munt

BASICS

6 Notes of the Network Administrator
by Doug Chick

I recently used SNORT and another program I like EtherApe to detect a major intrusion on my network. Within minutes millions of people were on my private fiber network. Once I isolated the problem I immediately connected my Internet provider. Like with many ISPs they denied it and recommended I look at my routing tables. If you are a network manager then you know in very many cases you must provide proof to your ISP before they are willing to provide you with support. In this case I recorded the event showing that there was hundreds of thousands, perhaps even a million people was passing traffic on my network. I sent the logs, and a video of my SNORT and EtherApe displays and emailed them to the ISP. I then shutdown the two interfaces on my router and waited for a return call. The call came quickly too.

8 Writing Snort Rules
by Kishin Fatnani

Though Snort can also be used for packet logging, sniffing or as an IPS, however in this article we will look more into the concept of rules by which Snort detects interesting traffic for us, basically the kind of traffic we are looking for, like a network attack, a policy violation or may be traffic from a network application or device that you are troubleshooting.

14 Collection and Exploration of Large Data
by Luca Deri

Collecting and exploring monitoring data is becoming increasingly challenging as networks become larger and faster. Solutions based on both SQL-databases and specialized binary formats do not scale well as the amount of monitoring information increases. In this article I would like to approach to the problem by using a bitmap database that allows to implementation of an efficient solution for both data collection and retrieval. NetFlow and sFlow are the current standards for building traffic monitoring applications. Both are based on the concept of a traffic probe (or agent in the sFlow parlance) that analyses network traffic and produces statistics, known as flows, which are delivered to a central data collector. As the number of flows can be pretty extremely high, both standards use sampling mechanisms in order to reduce the workload on both of the probe and collectors.

ADVANCED

18 Improving your custom Snort rules
by Leon Ward

While it is easy to create a custom Snort rule, do you know if you are actually making a good one or not? This article introduces some common mistakes I find in custom Snort rules and the potential implications of those mistakes. The Snort IPS engine has changed substantially over the last ten years. Packet processing speed has improved, IP defragmentation and stream reassembly functions have evolved, the connection and state tracking engine has matured, but there is one thing that keeps getting left behind.Custom rule-sets. With each revision of Snort new features are added that enhance the detection capability and aid in packet processing performance of the Snort engine.

24 An Unsupervised IDS False Alarm Reduction System – SMART
by Gina Tjhai and Maria Papadaki

Signature-based (or rule-based) network IDSs are widely used in many organisations to detect known attacks (Dubrawsky, 2009). A common misconception about IDSs is that they are Plug-and-Play devices that can be installed and then allowed to run autonomously. In reality, this is far from the truth.

30 Content Modifiers: Keep it Specific
by Joel Esler

Without going off the deep-end here and discussing every single Snort rule keyword, I just wanted to touch on a few modifiers that people sometimes misunderstand. These modifiers are not keywords of themselves, but rather they apply as modifiers to another keyword. That keyword is content. The content keyword is one of the easiest pieces of the Snort rules language as all it does is look for a particular string. The modifiers that I am talking about are: 1. Offset, 2. Depth, 3. Distance, 4. Within, 5. nocase, 6. http_uri 7. rawbytes.

DEFENSE

34 Deploying Snort as WAF (Web Application Firewall)
by Shivang Bhagat and Rishita Anubhai

In today's environment, web applications are becoming a popular attack point with attack agents. Attack agent can be a human attacker or an automated worm. It tries to explore vulnerabilities over HTTP(S) and exploit it for a given opportunity. The web application landscape is also changing and more complexities are getting added, it provides openings for vulnerabilities and possible exploitations. HTTP traffic is no longer restricted to name-value pairs and traditional HTML only. It has evolved with Web 2.0 and RIA, it allows JSON, AMF, XML and various other structures. It has become a platform for robust and advanced business application hosting and usage. It is imperative to secure business applications against all possible attack vectors and to maintain security of information and access. In this article we will try to understand Snort from HTTP standpoint and how we can protect applications for some of the popular attack vectors like XSS or SQL injections by using it.

40 Are you getting the most out of your IPS?
by K. K Mookhey

Picture this: a multi-billion dollar global telecom giant has invested millions of dollars into building a state-of-the-art Security Operations Center. They have a huge display screen being monitored 24/7 by a team of specialists who – so we are told – have received extensive training in the specific technologies used, as well as in the overall incident management framework. They've deployed a high-end intrusion prevention system (IPS) which feeds into their Security Incident Management (SIM) system. A review of the procedures and Service Level Agreement (SLA) of the SOC team signed with the rest of the business reveals that they are ready to respond 24/7 and have committed that within 2 hours of a serious attack they will respond to any serious attacks. On paper it all looks impressive and too good to be true.

up.time IT Systems Management Review

When it comes to the performance and availability of your IT infrastructure and applications, deep, and easy-to-use monitoring is a must.

Today, monitoring and reporting is more complex than ever, as applications and services span many environments (cloud, virtual and physical) and infrastructures (Windows, UNIX, Linux, VMware, etc). Additionally, IT infrastructure is now global and monitoring from one tool, instead of many point tools, is essential to drive down costs while increasing performance.

up.time's *Single pane of Glass* dashboard provides a deep, easy-to-use, affordable and complete IT systems management and monitoring solution designed for mid-enterprise companies. Every license in up.time's comprehensive, cross platform management and monitoring suite includes unlimited access to:

- Server Monitoring
- Virtual Server Monitoring
- Cloud Monitoring
- Co-Location Monitoring
- Network Monitoring
- SLA Monitoring & Management
- Virtualization & Consolidation
- Capacity Planning
- Application Monitoring
- Application Transaction Monitoring
- Proactive Outage Avoidance
- IT Process Automation

One of the highly beneficial capabilities of the up.time suite is access to Service Level Management. Most departments require SLA's (*Service Level Agreement*) for their equipment and applications. up.time makes it very easy to define and document agreed SLA's, then link them through to the appropriate infrastructure service. up.time also provides the ability to automate responses to issues, removing the possibility of human error while greatly decreasing the Mean-Time-To-Repair. In fact, up.time goes a step further and lets administrators proactively automate responses based on thresholds, allowing up.time to solve problems before they happen. It's not just physical, virtual and cloud based servers that up.time monitors, it also provides application and application transaction monitoring across Email, CRM, ERP, Web, and even custom applications (including any 3rd party commercial software or in-house developed applications).

$395 per Windows Server
$695 per UNIX Server
$695 per ESX Server (no charge per instance or VM)
All-in-One: No additional charges for modules or applications.

URL: http://www.uptimesoftware.com/

In addition to all of the above, up.time is extremely advance with its reporting. This area is a major asset of the up.time suite and it is immediately apparent that the reporting has had a good deal of thought and time spent on it. The reporting is both deep and very easy to create. Administrators can generate and save reports in different formats and quickly send (or set automated daily, weekly or monthly sends) via email to a single user or an entire group.

When we say up.time is easy to use, we really mean it. Installation of up.time was a dream, very simple, straightforward and easy to do. The entire process only takes a few mouse clicks. If you decide to go with the VMware appliance option, this is even easier as it comes as a pre-installed appliance that is can be imported into any virtual infrastructure.

Managing the monitored environment is achieved through a web portal which is simple, clean and easy to read (unlike other monitoring solutions that appear to have far, far too many menu's and options). Once you enter a small amount of information, the 'My Portal' home page is displayed. This page provides a summary list of the current alerts that you have configured together with saved reports and support links. All the tutorials are web based and, again, very clean and concise. The end result is that you are up and running with this product very quickly.

Everything about the product screams simplicity and yet it's extremely deep in its monitoring and reporting capabilities. Compared to other tools, up.time is very refreshing. It's certainly powerful enough for large enterprises to monitor over 5,000 servers and applications globally, and yet it's affordable for the small and mid enterprise companies to monitor between 25 and 500 servers and applications. The help documentation is included and available through your browser locally on the server.

up.time uses one of the friendliest licensing models in the industry, with its per-physical-server licensing across the board, even in virtual environments. All you need to do is count the number of physical servers you want to monitor, and that's it. Everything is included, no modules, hidden extras, application charges or management packs.

There is so much depth to this product that I can't comment on it all within the scope of this article. If this sounds interesting, up.time makes trying it for yourself incredibly easy. I suggest downloading the trial and taking it for a test drive. I think you'll be as impressed as I was. In fact, this product is so good, I'm starting to recommend that my clients review their monitoring needs and consider trialing in up.time.

MICHAEL MUNT

Doug Chick printable:

Notes of the Network Administrator

I have computer networking friends that work with various departments of the government, corporations and private companies that are very aware of the possible threats to their computers and networks.

What you will learn…

- overview of IDS
- overview of Snort

What you should know…

- basic knowledge of TCP/IP

They are and have been taking serious steps to secure their systems, despite little interest or concern from company or agency managers. Because of this lack of concern, many network security managers must take it upon themselves to secure their networks, but with little or no budget, they must rely or Open Source software to do it.

One such software I use is SNORT. SNORT is an open source network intrusion prevention and detection program (NIDS). This is a must have in any network managers security toolbox. If you are a Linux fan, then I'm sure you already know about SNORT, as it comes preinstalled with such Linux installs as Back/Track 4 and Knoppix-STD. Note there is also a Windows install.

SNORT, monitors, captures and analysis incoming packages and scans for a pattern of intrusion. In other words, it looks for specific packet signatures used by hackers, and automated hacking programs. Snort can detect attacks, probes, operating system fingerprinting, buffer overflows, port scans, and server message block scans. (In other words; all incoming network traffic.)

I recently used SNORT and another program I like EtherApe to detect a major intrusion on my network. Within minutes millions of people were on my private fiber network. Once I isolated the problem I immediately connected my Internet provider. Like with many ISPs they denied it and recommended I look at my routing tables. If you are a network manager then you know in very many cases you must provide proof to your ISP before they are willing to provide you with support. In this case I recorded the event showing that there was hundreds of thousands, perhaps even a million people was passing traffic on my network. I sent the logs, and a video of my SNORT and EtherApe displays and emailed them to the ISP. I then shutdown the two interfaces on my router and waited for a return call. The call came quickly too.

The ISP's main core router was hacked, and their routes were re-directed. Two hours later all the ISPs network engineers were called in. I stopped it on my end by shutting down the two interfaces it was coming in from, but it took them two more days to correct it on their end. I have redundant circuits from another provider, so I simply used those. The direct impact to me was minimal. Still, with the flood of hundreds of

Figure 1.

thousands of people directed to my private network with over twenty offices connected, I am still waiting to discover any long term damage. Privately one of the techs from my ISP told me later that they thought the intrusion came from China.

With the help of SNORT and EtherApe I was immediately alerted to a flood of unwanted traffic to my network. To me this reaffirmed the necessity of intrusion detection programs, and also made me research how many more types there are.

Types of Intrusion Detection Systems:

Intrusion prevention systems (ISP)

This device, also know as Intrusion Detection and Prevention Systems are network security appliances that monitor for malicious activity. This is a stand alone appliance identifies suspicious activity, isolates and logs it, and attempts to block.

Host-based intrusion detection systems (HIDS)

Host-based intrusion detection systems are installed on the computer level and monitor the actual server it is installed for suspicious activity, where ISPs operate on the network and analyze packets.

Protocol-based intrusion detection systems (PIDS)

This detection system is generally added in front of a web server and analyzes the HTTP, (and HTTPS) protocol stream and or port numbers.

Application protocol-based intrusion detection systems (APIDS)

APIDS typically are placed between servers and monitors the application state, or more accurately the protocols being passed between them. For example a web server that might call on a database to populate a webpage field.

I like SNORT because; one it is free, and two because of the support and its sheer flexibility. (I know, that is three things)

SNORT RULES…

Like with any intrusion detection device, SNORT has Rules.

```
alert tcp any any -> 192.168.1.0/24 111 \
   (content:"|00 01 86 a5|"; msg:"mountd access";)
```

The rules are actions that tells Snort what to do when it finds a packet that matches the rule criteria. There are 5 default actions; alert, log, pass, activate, and dynamic. If you are running Snort inline mode, there are additional options which include; drop, reject, and sdrop.

1. alert – generate an alert using the selected alert method, and then log the packet
2. log – log the packet
3. pass – ignore the packet
4. activate – alert and then turn on another dynamic rule
5. dynamic – remain idle until activated by an activate rule , then act as a log rule
6. drop – block and log the packet
7. reject – block the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP.
8. sdrop – block the packet but do not log it.

If you want to learn more about SNORT, I recommend you visit there site at: *www.snort.org*

I know there are other NIDS programs out there, and I'm sure they are just as good as SNORT, but as a network administrator/engineer this particular program has already proven itself to me.

EtherApe: Gorilla Virtual Warfare

As I mentioned before, another program I like is EtherApe. EtherApe is a graphical network monitor for UNIX modeled operating systems. It doesn't have the same features as SNORT, but what is does do is gives you a graphical overview, on what is going on in your network. I run EtherApe on a large screen monitor above my desk. When trouble comes, I can see an immediate flash of color that warns me that there is a possible situation on my network. This seemingly simple program has called me to action a couple of times. EtherApe has the ability to filter just the port number you want to monitor, or by default all of them. Like the name implies it works on an Ethernet network, but it also works with FDDI, Token Ring, ISDN, PPP, and SLIP.

DOUGLAS CHICK

Douglas Chick is a Director of Information Systems for a large company in the Orlando FL area. Although he introduces himself as a Network Engineer/Administrator. As with many computer people, Doug holds an MCSE and CCNA certification. Doug first became known on the Internet in May of 2000 for a series of articles about Microsoft retiring the NT4 MCSE that were published on over 30 Internet Magazines. With his humor and insightful look into the world on computer professionals, he receives a great deal of attention from other computer professionals around the world. And is proud to admit that less that one percent are in response for his many typo's and mis-spellings. For more visit: www.TheNetworkAdministrator.com

Writing Snort Rules

Snort, as you would know, is a tool used to detect intrusions on a network.

What you will learn…

- It will get you started with writing basic Snort rules
- Configuration of Snort is not covered here
- Preprocessors and advance rules are not covered

What you should know…

- Good knowledge about TCP/IP networks
- Packet Analysis
- Using and configuring Snort

Though the tool can also be used for packet logging, sniffing or as an IPS, however in this article we will look more into the concept of rules by which Snort detects interesting traffic for us, basically the kind of traffic we are looking for, like a network attack, a policy violation or may be traffic from a network application or device that you are troubleshooting. For instance, if someone is doing XMAS port scan to our network using nmap with the `-sX` option, Snort will give us the following alert message.

```
[**] [1:2000546:6] ET SCAN NMAP -f -sX [**]
[Classification: Attempted Information Leak] [Priority: 2]
10/15-08:51:46.970325 192.168.0.111:62202 ->
                    192.168.0.1:132
TCP TTL:53 TOS:0x0 ID:28031 IpLen:20 DgmLen:40
**U*P**F Seq: 0xD70FB1F3  Ack: 0x0  Win: 0x800  TcpLen:
                    20  UrgPtr: 0x0
[Xref => http://www.emergingthreats.net/cgi-bin/cvsweb.cgi/
   sigs/SCAN/SCAN_NMAP][Xref => http://
                    doc.emergingthreats.net/2000546]
```

If the use of P2P or IM applications is against the corporate policy, Snort can detect their use on the network and provide alerts with messages similar to these:

```
[**] P2P BitTorrent announce request" [**]
```

or

```
[**] CHAT  Yahoo IM successful logon [**]
```

To identify and alert on the appropriate events of interest, the detection engine needs to have rules for each event. A rule is what tells the engine where and what to look for in the network traffic and what needs to be done if detected. Here is a simple rule and the alert generated when that rule is triggered. Rule:

```
alert tcp any any -> 192.168.0.1 40404 (msg:"Access to
 port 40404 on server"; sid:3000001;)
```

Alert:

```
[**] [1:3000001:0] Access to port 40404 on server [**]
[Priority: 0]
10/15-14:47:19.676927 192.168.0.111:3022 -> 192.168.0.1:40404
TCP TTL:64 TOS:0x0 ID:6265 IpLen:20 DgmLen:40
***A**** Seq: 0x9197710 Ack: 0xF5F87F4  Win: 0x200 TcpLen: 20
```

Looking at this you must have got some idea about the components of a rule, but we shall go in to further depth and look at various options that can be used in the rules. So let us start with the above rule which is a quite simple one.

A rule is divided into two parts:

- Rule Header – the initial portion before the parentheses

- Rule Options – between the parentheses

The Rule Header

The header tells the engine about which packets to look into based on the protocol, IP & Port addresses and the direction in which the packet is travelling. The action to be taken upon detecting an event is also mentioned in the header. The following is the rule header format:

ACTION	PROTOCOL	IP ADDR	PORT NO.

DIRECTION	IP ADDR	PORT NO.

Let's take a look at the first field in the rule header which is -

ACTION

This field shows the action that will be taken when a packet matches the rule. Typically you would see *alert* or *log* here, which are self explanatory, but there are some more actions like *pass* to ignore the packet or *activate* to invoke another rule which has been idle due to its *dynamic* action.

PROTOCOL

Snort will match the addresses and other options only if the packet is part of the protocol mentioned here, which could be `ip`, `icmp`, `tcp` or `udp`.

IP ADDR

For the rule to be triggered, the IP address specified here must match the IP address in the packet. Each packet has an IP header which has two IP addresses, source and destination. It will depend on the direction field whether the IP address given here will be matched with the source or the destination. This field can have a specific IP, network address with CIDR notation, multiple IP addresses enclosed within square brackets and separated by comma or the word 'any' to ignore the IP address.

We can use variable representing IPs of servers or networks which makes it easier to manage. The variable are declared initially with the keyword `var` and subsequently the variable name preceded by a dollar sign can be used in the rules. Variable decalaration

```
var TEST_SERVER 192.168.0.1
```

Using the variable

```
alert tcp any any -> $TEST_SERVER 40404 ...
```

There are some predefined variable names for IPs and networks which are used in the default ruleset, some of them include

```
HOME_NET, EXTERNAL_NET, HTTP_SERVERS
```

PORT NO.

If the protocol used is TCP or UDP, there will be a respective header attached to the packet which will have 2 port addresses, source and destination. Again, for the rule to be triggered, the port nos. must also match. Whether this field will be matched with the source or destination port will depend on the direction field. A port number may be a single number or a range given with the `:` separating the lower and upper boundaries e.g. 1:200 for port numbers from 1 to 200.

Just as in IP ADDR, variables can be used to represent port numbers of services like this:

Variable declaration

```
portvar TEST_PORT 40404
```

Using the Variable

```
alert tcp any any -> $TEST_SERVER $TEST_PORT ...
```

There are some predefined variable names for ports which are used in the default ruleset, some of them include – `HTTP _ PORTS`, `SHELLCODE _ PORTS`...

DIRECTION

This field stands for the direction in which the packet must be travelling to match this rule. If the direction given is `->` then the IP and port addresses on the left side of this sign will be matched with the source IP and port while the right side will be matched with the destination. In case of the `<>` sign, the left side address will be checked in source as well as destination and if found in one, the right side will be matched with the other.

Note

The exclamation sign `!` can be used to negate the IP ADDR and PORT NO. fields

Coming back to our previous example,

```
alert tcp any any -> 192.168.0.1 40404 (msg:"Access to
 port 40404 on server"; sid:3000001;)
```

we are telling Snort to alert us if it sees a packet with any IP and any port number in the source (being left side of the direction -> field) and the IP address 192.168.0.1 and port number 40404 in the destination. Now suppose there is a service running on port 40404 on the server 192.168.0.1 and some client connected to the service and transmitted and received several messages. If this communication is captured by Snort, we surely are going to receive the alert as given above. The question here is that will Snort alert us while the connection is being established, after it has been successfully established, at the first message, or at all the messages sent and received?? To know that we need to look at the rule and its description below it which clearly says that an alert to be generated whenever any packet is seen to be going to this service. By this explanation it is very clear that we are going to receive a lot of alerts as Snort will be looking at the communication packet by packet and each packet going towards the server will trigger the alert. Hence, we are going to get an alert right from when the client requests connection to the service (SYN packet), then the acknowledgement from the client to server, each packet containing data (message) or acknowledgement from the client to server, the finish (FIN) and finish acknowledgement packets. For any packets from the server to the client, there will be no alert, hence the SYN/ACK will not give an alert, neither any messages received by the client.

So, does that serve the purpose? Well, it depends on what was the objective for writing such a rule. If you are just looking for attempts to connect to the service or successful connections, then this would create a lot of noise by giving too many alerts which are not required. However, it may be useful if you are trying to debug or troubleshoot a network application or analyze malware activities like if you want to see how frequently is the application sending a keep-alive message while being idle or if you want to check on the activity of a bot.

To reduce the noise and get a single alert on connection or when a specific message is being sent, we need to use the rule options which are enclosed within the parentheses.

The Rule Options

We have already used a couple of options in our sample rule but those options did not have any role in the detection logic. These options are called *general or metadata options* which provide information about the rule or event and do not have affect during detection. The description of the event which is displayed in the alert comes from the rule option `msg`. This option consists of the keyword `msg` followed by an argument, the message itself, separated by a `:`. Each option is then followed by a `;` to end the option and begin another if any. Some options do not require an argument hence they also don't require the `:`, however, the `;` is still required.

sid and rev

The other option that we have in our rule is the 'sid' option which uniquely identifies Snort rules. Each rule in Snort has a unique sid which is provided by the author of the rule. To ensure that it does not conflict with the sid of the rules included in the Snort distribution, the sid must be above 1000000. Other community rules may have their own series which must be avoided in custom rules. The sid is often accompanied by the `rev` option which uniquely identifies revisions of that rule. So if the rule is revised, it doesn't need to have a new sid but it will have a different rev. These numbers are also shown in the alert just preceding the alert message.

Testing the rule

Now that we have understood the rule and know what to expect from it, we need to verify the rule and see if it really works as our expectation. We can do that by crafting packets in such a way that when they are sent over the network monitored by Snort, they should trigger the rule. There are many tools available on the Internet which can help with this, some being GUI based while others command line. My preference is command line as it gives us more control and flexibility by scripting. The tools I use are `hping` and `scapy`. An explanation on the usage of these tools is out of scope, however to get the feel of the simplicity and power of these tools, we'll give you an example below. In `hping`, you can simply use the following options to trigger our sample rule:

```
hping -c 1 192.168.0.1 -p 40404
(-c 1 is to send a single packet)
```

while in scapy, the following command can be run from the scapy prompt:

```
>>> send(IP(dst="192.168.0.1")/TCP(dport=40404))
```

Making the rule more specific

We do not want alerts for each and every packet of the connection as just the knowledge about the connection attempt would be good enough. In this case we can add another option which will make the rule trigger only at the connection request from the client side. As we know that a connection request packet has only the SYN bit set, so we can instruct Snort to look for this bit. The option for this is `flags` and it takes the initial letter of the required flags as the argument like the one here:

```
alert tcp any any -> 192.168.0.1 40404 (flags:S; msg:
"Access to port 40404 on server"; sid:3000001;)
```

Here we are saying that alert us only if the TCP header in the packet has the SYN bit exclusively set. This will add to all the previous criteria of Protocol, IPs and Ports as specified in the rule header. Now we will not get all the noise we were getting earlier but there is still a problem which can give us false negatives i.e. it may not alert us even if the connection is being requested on the said service. This is because the first packet in the three-way handshake in TCP can have an ECN flag set along with the SYN flag. The two ECN flags were reserved earlier, hence they are represented by the number 1 and 2 rather than the initial alphabet. We now need to alter the flags arguments to account for the possibility of an ECN flag being set. This can be done by adding the optional flags preceded by a comma, as given below:

```
alert tcp any any -> 192.168.0.1 40404 (flags:S,12; msg:
"Access to port 40404 on server"; sid:3000001;)
```

The flags option affects the detection, hence it cannot be called a general or metadata option. The options affecting detection are *payload or non-payload options*. As the flags option refers to the information in the TCP header and not the payload, hence it is a *non-payload option*.

Payload Options

Most rules do not just depend on the header fields to identify an event of interest but they look into the payload i.e. the data passed in the packets. The headers are based on standard specifications, hence Snort can identify each field of a header and lets us refer to them by keywords (e.g. *flags*, *ttl*, *id*), however, the payload may have variable data depending on the application used to generate it. To search for a specific string in the payload, we can use the rule option *content* which lets us specify the data to be searched in ascii text or in hexadecimal.

Now, we also want to be alerted when the admin user is logging in to the service, so we first analyze the traffic using a sniffing tool which could be Snort itself or some other like *tcpdump*, or *Wireshark*. From our analysis we need to find the pattern for admin login. The following is the output from *tcpdump* when the admin login was in process (see Listing 1).

As we see from the packet capture that there is a string *usr admin* which is being used as a command to login as admin. This string can be used as our pattern to detect admin login. We can write the following rule to search the pattern:

Listing 1. *Only TCP Payload is displayed here*

```
0x0020:     0201 0075 7372 2061                             ...usr.a
0x0030:     646d 696e 0001 7565 6a34 3739 6a66 6a76         dmin..uej479jfjv
0x0040:     6e76 6a66 6a65 3433 3334 3500 7065 726d         nvjfje43345.perm
0x0050:     3a77 7777 7800                                  :wwwx.
```

Listing 2. *Only TCP Payload is displayed here*

```
0x0020:     0201 0075 7372 2061                             ...usr.a
0x0030:     646d 696e 0001 7565 6a34 3739 6a66 6a76         dmin..uej479jfjv
0x0040:     6e76 6a66 6a65 3433 3334 3500 7065 726d         nvjfje43345.perm
0x0050:     3a77 7777 7800                                  :wwwx.
```

```
alert tcp any any -> 192.168.0.1 40404 (content:"usr admin";
 msg:"user admin login detected"; sid:3000002; rev:1; )
```

This rule will trigger whenever the admin login happens, however there is also a good possibility of it triggering at other times when this string is not being used as a command but as some data. This would be termed as a *false positive*. While choosing patterns, we must keep in mind that our pattern is as much as possible unique to the required event.

Reducing False Positives

We will analyze the packets more closely looking for more patterns to reduce the possibility of false positives. Firstly we can add the constant binary data preceding and following our previous string. Binary data can be represented in hexadecimal and enclosed within the | characters.

```
alert tcp any any -> 192.168.0.1 40404 (content:"|02 01 00
|usr admin|00 01|"; msg:"user admin login detected";
                    sid:3000002; rev:2; )
```

If the binary data was not constant in all the admin logins, then we need to look at other parameters. Let's say the string *usr admin* always starts after three bytes of some variable data. We can instruct Snort to skip the first three bytes of the payload and start matching the pattern string immediately from there. For this we need to use two *content modifiers* – *offset* and *depth*. Content modifiers affect how the content option works and have to follow the required content option for e.g. (content: *usr admin*; *nocase*;) will look for the string *usr admin* but ignore the case, here the *nocase* is a content modifier.

```
alert tcp any any -> 192.168.0.1 40404 (content:"usr admin";
offset:3; depth:9; msg:"user admin login detected"; sid:
                    3000002; rev:3; )
```

With this rule, Snort will skip the first 3 bytes of the payload and start searching the string *usr admin* within the next 9 bytes. This way the rule is also optimized as the search will not be carried out for the entire payload.

Relative Content Matching

To further reduce the possibility of false alarms, we can match additional pattern if the first match is successful. We have noticed that there is a string *perm:* which always comes 24 bytes ahead of the *usr admin* string (see Listing 2).

The rule logic to match the second pattern will be same, however the keywords used here are different as now the match begins from where the previous match ended. The content modifiers used for this new pattern are *distance*, which is similar to offset, the only difference being that distance is an offset from the end of previous match whereas offset starts from the beginning of the payload, and the other one is *within* which is similar to depth but works in conjunction with distance.

```
alert tcp any any -> 192.168.0.1 40404 (content:"usr admin";
offset:3; depth:9; content:"perm:"; distance:24; within:
5; msg:"user admin login detected"; sid:3000002; rev:4; )
```

There are many more ways to enhance the rule, maybe we can cover them in the next article. If you have to say anything about this article or have a wishlist for the next one, you can write to me and I shall surely consider.

KISHIN FATNANI – CISSP, GCIH GOLD, GCFA, CCSE R70, CEH, LPT

Kishin is the founder of K-Secure (www.ksecure.net), an IT security training company. He has over 20 years experience in IT, main focus being security, and conducts many security trainings including Packet Analysis, Snort Rule Writing, Ethical Hacking, Forensic Analysis, Web Application Security, Check Point and more. He has also been a SANS local mentor for the GIAC certification GCFA and has some contributions to the development of Check Point courseware and certification exam.

His contact him at kishinf@ksecure.net
Blog: kishinf.blogspot.com
Facebook: www.facebook.com/ksecure
Twitter: www.twitter.com/ksecure

The Challenge of Compliance

Q. What are the sorts of challenges large corporations are facing with regards to compliance?

KK: The challenges are many and this is because the compliance environment is becoming more complex. Business operations are becoming dynamic by the day, regulators are becoming more watchful when it comes to data security and customers are becoming more demanding. A very simple way to know you've got a problem is to ask your CISO: *How secure are we as an organization*. If you find him hemming-hawing or giving some vague response, then you really don't have a handle on your information security.

Q. How are companies dealing with this challenging environment?

KK: What enterprises should aim to do is put in place a framework, which helps critical questions that provide a clear idea of where the organization stands with regards to information security. Our firm makes one such compliance platform called as NX27K (*http://www.niiconsulting.com/products/iso_toolkit.html*) – which helps companies comply with the multiple information security compliance frameworks and regulations, such as ISO 27001, PCI DSS, SOX, etc.

Q. What should be the capabilities of such a platform?

KK: The key feature of such a platform should be flexibility. And this is the key principle on which NX27K is built. You can modify what you see and put into your asset register, or modify your risk assessment formula, modify the audit trail, the risk register, and almost every aspect of the platform. It integrates with Sharepoint and Active Directory. Further, it adapts risk assessment guidelines such as from NIST and COBIT's Risk-IT

Q. Where can customers go for more information?

KK: Customers can email us at *products@niiconsulting.com* or visit us at *http://www.niiconsulting.com/products/iso_toolkit.html* for more information on this product and our other services and products.

Collection and Exploration of Large Data

Why the use of FastBit is a major step ahead when compared with state of the art relational database tools based on relational databases.

What you will learn…

- Basics of building traffic monitoring applications
- Data Collection and Exploration

What you should know…

- A basic knowledge of architecture and implementation of traffic monitoring tools
- A basic knowledge of TCP/IP

Collecting and exploring monitoring data is becoming increasingly challenging as networks become larger and faster. Solutions based on both SQL-databases and specialized binary formats do not scale well as the amount of monitoring information increases. In this article I would like to approach to the problem by using a bitmap database that allows to implementation of an efficient solution for both data collection and retrieval.

NetFlow and sFlow

NetFlow and sFlow are the current standards for building traffic monitoring applications. Both are based on the concept of a traffic probe (or agent in the sFlow parlance) that analyses network traffic and produces statistics, known as flows, which are delivered to a central data collector. As the number of flows can be pretty extremely high, both standards use sampling mechanisms in order to reduce the workload on bothof the probe and collectors. In sFlow the use of sampling mechanisms is native in the architecture so that it can be used on agents to effectively reduce the number of flows delivered to collectors. *This has a drawback in terms of result accuracy while providing results with quantifiable accuracy*. With NetFlow, the use of sampling (both on packets and flows) leads to inaccuracy and this means that flows sampling is very seldom used in NetFlow hence there is no obvious mechanism for reducing the number of flows records while preserving accuracy. For these reasons, network operators usually avoid sampling data hence have to face with the problem of collecting and analyzing a large number of flows that is often solved using a flow collector that stores data on a SQL-based relational database or on disk in raw format for maximum collection speed. Both approaches have pros and cons; in general SQL-based solutions allows users to write powerful and expressive queries while sacrificing flow collection speed and query response time, whereas raw-based solutions are more efficient but provide limited query facilities.

The motivation is to overcome the limitations of existing solutions and create a new generation of a flow collection and storage architecture that exploits state-of-the-art indexing and querying technologies. In the following I would like to describe the design and implementation of nProbe , an open-source probe and flow collector, that allows flows to be stored on disk using the FastBit database.

Architecture and Implementation

nProbe is an open-source NetFlow probe that also supports both NetFlow and sFlow collection and, flow conversion between version (for instancei.e. convert v5 to v9 flows). It fully supports the NetFlow v9 specification so giving it has the ability to specify flow templates (i.e. it supports flexible netflow) that are configured at runtime when the tool is started (Figure 1).

When used as probe and collector, nProbe supports flow collection and storage to either raw files or relational databases such as MySQL and SQLite. Support of relational databases has always been controversial as users appreciated the ability to search flows using a SQL interface, but at the same time flow dump to database is usually enable only realistic for small sites. The reason is that enabling database support could lead to the loss of flows due to the database overhead. There are multiple reasons that contribute to this behavior and in particularincluding:

- Network latency and multi-user database access for network-based databases.
- Use of SQL that requires flow information to be converted into text that is then interpreted by the database, instead of using an API for directly writing into the database.
- Slow-down caused by table indexes update during data insertion.
- Poor database performance when searching data during data insert.

Databases offer mechanisms for partially avoiding some of the above issues, which includinge:

- Data insert in batch mode instead of doing it in real time.
- Avoid network communications by using file-based databases.
- Disable database transactions.
- Use efficient table format optimized for large data tables.
- Not defining tables indexes therefore avoiding the overhead of index updates, though usually results in slower data search time.

Other database limitations include the complexity of handling large databases containing weeks of data, and purging old data while still accommodating new flow records. Many developers partition the database often creating a table per day that is then dropped when no longer needed.

The use of file-based databases such as SQLite offer a few advantages with respect to networked relational databases, as:

- It is possible to periodically create a new database (e.g. one database per hour) for storing flows received during that hour, this is in order to avoid creating large databases.
- According to some tests performed, the flow insert throughput is better than networked-databases but still slower than raw flow dump.
- In order to both overcome the limitations of relational databases, and avoid raw flow dump due to limited query facilities, I decided to investigate the use of column-based databases and in particular, of FastBit .

Validation and Performance Evaluation

I have used the FastBit library for creating an efficient flow collection and storage system. This is to demonstrate that nProbe with FastBit is a mature solution that can be used on in a production environment. In order to evaluate the FastBit's performance, nProbe has been deployed in two different environments:

Medium ISPs

The average backbone traffic is around 250 Mbit/sec (about 40K pps). The traffic is mirrored onto a Linux PC (Linux Fedora Core 8 32 bit, Kernel 2.6.23, Dual Core Pentium D 3.0 GHz, 1 GB of RAM, two SATA III disks configured with RAID 1) that runs nProbe in probe mode. nProbe computes the flows and saves them on disk using FastBit. In order to reduce the number of flows, the probe is configured to save flows in NetFlow v9 bi-directional format with maximum flow duration of 5 minutes. In average the probe generates 36 million flows/day. Each FastBit partition stores one hour of traffic. Before deploying nProbe, flows were collected and stored in a MySQL database.

Large ISPs

nProbe is used in collector mode. It receives flows from 8 peering routers, with peak flow export of 85 K flows/sec. The collection server is a fast machine with 8 GB of memory, running Ubuntu Linux 9.10 server 64 bit. Each FastBit partition stores five minutes of traffic that occupy about 5.8 GB of disk space. A second server running Ubuntu Linux 9.10 server 64bit and 24 GB of memory is used to query the flow data. The FfastBbit partitions are saved to a NFS mount on a local storage server. Before deploying

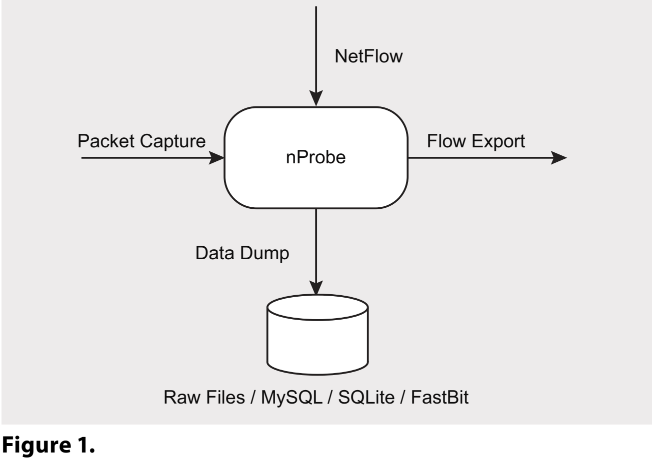


Figure 1.

Table 1. *FastBit vs MySQL Disk Usage (results are in GB)*

MySQL	No/With Indexes	1.9 / 4.2
FastBit	Daily Partition (no/with Indexes)	1.9 / 3.4
	Hourly Partition (no/with Indexes)	1.9 / 3.9

nProbe, flows were collected using nfdump and each month the total amount of flow dumps exceeds 4 TB of disk space. The goal of these two setups is to both validate nProbe with FastBit on two different setups and compare the results with the solution previously used.

FastBit vs Relational Databases

Let´s compare the performance of FastBit with respect to MySQL (version 5.1.40 64 bit), a popular relational database. As the host running nProbe is a critical machine, in order not to interfere with the collection process, two days worth of traffic was dumped in FastBit format, and then transfered to a Core2Duo 3.06 GHz Apple iMac running MacOS 10.6.2. Moving FastBit partitions across machines running different operating systems and word length (one is 32 the other is 64 bit) has not required any data conversion. This is a good feature as over-time collector hosts can be based on various operating systems and technology; hence flow archives can be used immediately without any data conversion is a desirable feature. In order to evaluate how FastBit partition size affects the search speed, hourly partitions have been merged into a single daily partition. In order to compare both approaches, five queries can be defined:

- Q1: `SELECT COUNT(*),SUM(PKTS),SUM(BYTES) FROM NETFLOW`
- Q2: `SELECT COUNT(*) FROM NETFLOW WHERE L4_SRC_PORT=80 or L4_DST_PORT=80`
- Q3: `SELECT COUNT(*) FROM NETFLOW GROUP BY IPV4_SRC_ADDR`
- Q4: `SELECT IPV4_SRC_ADDR,SUM(PKTS),SUM(BYTES) AS s FROM NETFLOW GROUP BY IPV4_SRC_ADDR ORDER BY s DESC LIMIT 1,5`
- Q5: `SELECT IPV4_SRC_ADDR, L4_SRC_PORT, IPV4_DST_ADDR, L4_DST_PORT, PROTOCOL, COUNT(*), SUM(PKTS), SUM(BYTES) FROM NETFLOW WHERE L4_SRC_PORT=80 or L4_DST_PORT=80 GROUP BY IPV4_SRC_ADDR, L4_SRC_PORT, IPV4_DST_ADDR, L4_DST_PORT, PROTOCOL`

FastBit partitions have been queried using the fbquery tool with appropriate command line parameters. All MySQL tests have been performed on the same machine with no network communications between client and server. In order to evaluate the influence of MySQL indexes on queries, the same test has been repeated with and without indexes.

Data used for testing washave been captured on Oct 12th and 13th (~68 million flows) and contained a subset of NetFlow fields (IP source/destination, port source/destination, protocol, begin/end time). The table below compares the disk space used by MySQL and FastBit. In the case of FastBit, indexes have been computed on all columns.

Merging FastBit partitions does not usually improve the search speed but instead queries on merged data requires more memory as FastBit has to load a larger index in memory. In terms of query performance, FastBit is far superior compared with MySQL as shown in Table 2:

- Queries that require access only to indexes take less than a second, regardless of the query type.
- Queries that require data access are at least an order of magnitude faster that on MySQL.
- Index creation time on MySQL takes many minutes and it prevents its use in real life when importing data in (near-)realtime, and also indexes also take a significant amount of disk space.
- Indexes on MySQL do not speed up queries, contrary to FastBit.
- Disk speed is an important factor for accelerating queries. In fact running the same test twice with data already cached in memory, significantly decreases the query speed. The use of RAID 0 has demonstrated that the performance speed has been improved.

Open Issues and Future Work

Tests on various FastBit configurations have shown that the disk is an important component that has a major impact on the whole system. I am planning to explore the use of solid-state drives in order to see if the overall performance can benefit from it.performance increases.

Table 2. *FastBit vs MySQL Query Speed (results are in seconds)*

Query	MySQL		Daily Partitions		Hourly Partitions	
	No Index	With Indexes	No Cache	Cached	No Cache	Cached
Q1	20.8	22.6	12.8	5.86	10	5.6
Q2	23.4	69	0.3	0.29	1.5	0.5
Q3	796	971	17.6	14.6	32.9	12.5
Q4	1033	1341	62	57.2	55.7	48.2
Q5	1754	2257	44.5	28.1	47.3	30.7

A main limitation of FastBit is the lack of data compression as it currently compresses only indexes but not data. This is a feature is planned to add, as it allows disk space to be saved hence to reducereducing the time needed to read the data.

This article is the base for developing interactive data visualization tools based on FastBit partitions. Thanks to recent innovations in web 2.0, there are libraries such as the Google Visualization API that allow separating data rendering from data source. Currently we are extending nProbe adding an embedded web server that can make FastBit queries on the fly and return query results in JSON format. The idea is to create an interactive query system that can visualize both tabular data (e.g. flow information) and graphs (e.g. average number of flows on port X over the last hour) by performing FastBit queries. This way the user does not have to interact with FastBit tools at all, and can focus on data exploration.

Final Remarks

The use of FastBit is a major step ahead when compared with state of the art tools based on both relational databases and raw data dumps. When searching data on datasets of a few million records the query time is limited to a few seconds in the worst case, whereas queries that just use indexes are completed within a second. The consequence of this major speed improvement is that it is now possible to query data in real time and avoid updating costly counters every second, as using bitmap indexes it is possible to produce the same information when necessary. Finally this work paves the way to the creation of new monitoring tools on large data sets that can interactively analyze traffic data in near-real time, contrary to what usually happens with most tools available today.

Availability

This work is distributed under the GNU GPL license and is available at the ntop home page *http://www.ntop.org/nProbe.html*. The nBox appliance embedded withing a pre-installed ntop and nProbe software can be requested at *www.wuerth-phoenix.com/nbox*.

LUCA DERI, FOUNDER OF NTOP

Luca Deri was born in 1968. Although he was far too young to remember, the keywords of that year were freedom, equality, free thinking, revolution. In early 70s many free radio stations had birth here in Italy because their young creators wanted to have a way for spreading their thoughts, ideas, emotions and tell the world that they were alive 'n kickin'. The Internet today represents for him what free radio represented in the 70s. He wrote his PhD on Component-based Architecture for Open, Independently Extensible Distributed Systems. Luca Deri is the founder of Ntop.

Improving your custom Snort rules

While it is easy to create a custom Snort rule, do you know if you are actually making a good one or not? This article introduces some common mistakes I find in custom Snort rules and the potential implications of those mistakes.

What you will learn…

- How to measure the performance of your custom Snort rule-set, and how to identify the "bad" performers
- How to use Snort's fast_pattern keyword to your advantage
- How to add more descriptive information to your rules to improve the analysis process

What you should know…

- The Snort rule language
- A basic knowledge of TCP/IP
- How to install and Perl modules via CPAN on a *NIX operating system

The Snort IPS engine has changed substantially over the last ten years. Packet processing speed has improved, IP defragmentation and stream reassembly functions have evolved, the connection and state tracking engine has matured, but there is one thing that keeps getting left behind. Custom rule-sets.

With each revision of Snort new features are added that enhance the detection capability and aid in packet processing performance of the Snort engine. Those enhancements not only open new avenues for detecting the latest *bad stuff* out there, they create an opportunity to improve the performance of older legacy rules you may have created many years ago. Unless your rules make good use of the current Snort language and are kept up-to-date, what once used to be a good rule could in fact turn bad.

What is a bad Snort rule anyway?

Because this article focuses on finding and fixing bad rules, before we look at any of them it would be wise for me to define what I personally call a bad rule.

- A rule that cannot or will not perform the function it's made for. I.e. it won't catch the attack/event that the rule tries to find (False negative)
- A rule that catches the wrong stuff (False positive)
- A rule that has little meaning or value to the analyst (Junk alerts)
- A rule that wastes processing time and effort to achieve its goal (performance hog)

The first two points in the list (dealing with false positives and false negatives) will always need to be addressed on a case-by-case basis, however when it comes to the last two points there are some important concepts you can follow that will substantially improve custom rules regardless of what attack or event they are designed to catch.

There are many books and online resources available that discuss the Snort rule language in depth, and a full introduction is far out of the scope of this article. However, to enable me to present some common rule problems, I need to introduce the basic building blocks of a Snort rule.

A Snort rule is made up of two parts, a rule header and a rule body. The rule body follows the rule header and is

Listing 1. *An example "bad" Snort rule*

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 \
   (msg: "0xdeadbeefbadfoo Detected"; \
   flow: established, from_client; \
   content: "0xdeadbeefbadfoo"; \
   rev:1; sid:1000001;)
```

surrounded by parentheses. The header is pretty easy to understand as it reads close to natural language. The rule header consists of an action, a protocol specification, and the traffic that is to be inspected.

The rule body (shown in Listing 1 in blue) is made up of a selection of rule options. A rule option consists of a keyword followed by one or more arguments. For example in the above rule there is a content keyword, with an argument of `0xdeadbeefbadfoo`.

This rule instructs Snort to look for the text `0xdeadbeefbadfoo` in all packets flowing out of the network to TCP port 80 that are part of an established TCP session.

Giving rules more meaning to an analyst

The above rule in Listing 1 is a simple example of a bad rule. Regardless of how much processing load it may introduce to an IDS engine, or how many alerts it could generate, just by looking at the rule source we can see it's bad. This rule lacks some of the most important information that could give any value to its existence and operation – an understanding of what's happening on the network, and what it means if it generates an alert.

When describing or teaching the Snort rule language, I like to group rule options together to describe them in three categories based on their function within the rule.

- **Detection Options:** Keywords that test for the presence of specific things in network traffic. This could be any type of text, binary data, packet header values, regular expressions, or decoded application data. These are the keywords that control if an alert is generated on a packet or not. Example Snort keywords: *content*, *pcre*, *ipopts*, *ttl*, *flowbits*, *flow*
- **Metadata Options:** These are keywords that are interpreted by the engine for alert organization. Example Snort Keywords: *sid*, *metadata*, *rev*
- **Analyst Options:** Keywords that are used by the rule writer to convey information to an event analyst who is investigating the event. Example Snort keywords: *msg*, *classtype*, *reference*, *priority*

While writing a rule it is important to understand that any event it generates may need to be understood by other people. If a security analyst is presented with an event like the below which was generated from our previous *bad rule*, what do they do to respond to the event?

```
Oct  7 15:16:30 DMZ1 snort: [1:100001:0] [**]
   0xdeadbeefbadfoo Detected  [**] [Priority: 0 ]{TCP}
     192.168.0.1:25541 -> 192.168.0.2:80
```

Well after the analyst has likely scratched their head for a moment, and then wondered what on earth `0xdeadbeefbadfoo` is, they will probably end up Googling for `0xdeadbeefbadfoo` in an effort to understand what this alert means to them.

- Is this a serious event?
- Is it somebody else's problem?
- Should I start panicking?

It is common to have a different group of people researching and writing rules vs. those will who deal with the security events they may raise, and if this isn't the case for you today it may well be in the future. At the time of rule creation only the rule writer really knows what she or he is looking for, and the implications to the network if the traffic is found. It is therefore critical for this information to be passed on to the event analyst within the rule itself. Unless a rule is correctly explained, how can a writer expect an analyst to be able to react accordingly?

Let's expand on my simple `0xdeadbeefbadfoo` example from earlier by providing some more theoretical scenario information (Listing 2).

Note the addition of three new rule options: A classification type, an overriding priority qualification, and a couple of useful references. With these extra rule options added an analyst dealing with the event now knows that `0xdeadbeefbadfoo` is in fact a low-priority Trojan, associated with CVE:2010-99999, and a related to a specific company's product.

These seemingly minor additions make massive returns in respect to the security event analysis and remediation process. Sometimes the simplest changes provide the greatest value.

Identifying and optimizing slow rules that are wasting your CPU cycles

So while fixing the analyst information problem is pretty simple, identifying suboptimal rules in terms of

Listing 2. *The bad rule, now improved with far more information to help an analyst*

```
alert tcp $HOME_NET any -> $EXTERNAL_NET 80 \
   (msg: "0xdeadbeefbadfoo Detected"; \
   content: "0xdeadbeefbadfoo"; \
   classtype: trojan-activity; \
   priority: 3; \
   reference: cve,2010-99999; \
   reference: url, http://mycompany.com/myproduct; \
   sid:100001; rev:2;)
```

computational overhead is a little more of a technical process. To make this challenge possible Snort can kindly provide us feedback of how the system functions in relation to the current configuration and network traffic being inspected.

There are a couple of useful configuration lines that can be added to your snort.conf to provide performance feedback about how the detection engine is performing. Today I will focus on the output provided by `profile_rules`.

```
config profile_rules: print 10, sort total_ticks
```

Adding this `profile_rules` configuration directive to your snort.conf will enable performance profiling of your snort rule-set. At exit, Snort will output to STDOUT a list of the top N (specified here as ten) worst performing rules categorized by the total time taken to check packets against them. This data can also be written to a text file of choice, and many other sort methods are available. Check the snort manual for full details.

Note
Snort must be compiled with `--enable-perfprofiling` to enable the performance profiling capability.

Before starting to inspect the performance output, it is vital to understand that all of the data we see is dependant on two distinct variables:

- The current configuration running (including the rule-set)
- The network traffic that is inspected by the detection engine

When testing and tweaking anything as complex as an IPS rule-set for performance, I find it imperative to isolate and work on only a single variable at a time. By focusing my tests on a large sample of network traffic stored in PCAP files that is representative to where the sensor operates, I can tweak my rules for performance against this static data-set. When I think I have optimally tweaked any rules, I can then move to test against live traffic.

An example of rule profiling output is shown in Listing 3, and each data column is explained below.

- **Num:** This column reflects this rule's position number in regard to how bad the rule performs. Here the top (number 1) reflects the rule that is responsible for consuming the most processing time (`total_ticks`)
- **SID, GID, Rev:** The Snort ID, Generator ID, and Revision number of the rule. This is shown to help us identify the rule in question in our rule-set.
- **Checks:** The number of times rule options were checked *after the fast_pattern match process* (yes, that bit is bold because it is important).
- **Matches:** The number of times all rule options match, therefore traffic matching the rule has been found.
- **Alerts:** The number of times the rule generated an alert. Note that this value can be different from *Matches* due to other configuration options such as alert suppression.
- **Microsecs:** Total time taken processing this rule against the network traffic
- **Avg/Check:** Average time taken to check each packet against this rule.
- **Avg/Match:** Average time taken to check each packet that had all options match (the rule could have generated an alert)
- **Avg/Nonmatch:** Average time taken to check each packet where an event was not generated (amount of time spent checking a clean packet for bad stuff)

The two values a rule writer has some level of control over are the number of checks, and how long it took to perform those checks. Ideally we would like to have low figures in all columns, but decreasing the *Checks* count is the first important part of rule performance tuning. To be able to tweak our rule to affect this value, we need to first understand exactly what *Checks* represents.

Introducing the fast_pattern matcher

When Snort decides what rules need to be evaluated against a network packet, it goes through two stages before starting its in-depth testing functions.

Listing 3. *Sample Snort rule profiling output*

```
Rule Profile Statistics (all rules) total sort
   Num       SID GID Rev     Checks   Matches    Alerts   Microsecs  Avg/Check  Avg/Match Avg/Nonmatch
   ===       === === ===     ======   =======    ======   =========  =========  ========= ============
     1       112   1   1        208        69        69         187        0.9        2.0          0.3
     2       111   1   1        208       208       208         151        0.7        0.7          0.0
     3       113   1   3         69        69        69          27        0.4        0.4          0.0
```

1) Protocol, Port number and service identification
Snort optimizes the rule-set into protocol, port and service (application protocol) based detection rule-buckets. *Note that service based buckets are only used when Snort is compiled with –enable-targetbase, and an attribute table is loaded*.

For example, if inbound traffic is destined to arrive at TCP:80 (HTTP), there isn't much point in running it though the rules associated with SMTP (TCP:25). The packet is assessed against the rules in the TCP:80 bucket. The same decision is also made related to source port and service metadata.

Snort also has an extra rule-bucket for the *any any* rules. These are the rules that use the value *any* as both the source and destination port numbers. All packets are also checked against the *any any* rules as well after being assessed against their particular port/service based rule bucket.

2) Packet content (fast_pattern check)
After identifying what rule bucket(s) this packet should be assessed against, a pre-screening content check known as the `fast_pattern` match is applied for all rules in the bucket(s).

For any Snort rule to raise an event *all* rule-options in that rule *must* match.

Applying a `fast_pattern` check process allows Snort to *quickly* test packets for the presence of a static content string (a single *content:* value) required to generate an event. The goal of this test is to quickly identify all packets that have *any* possibility of alerting after all of the rule options are tested. If a packet doesn't match the `fast_pattern` check, there is absolutely no point in running more computationally intense checks against it. Because the `fast_pattern` match has failed, we know that at least one of the rule options will not match, and an alert will never be generated.

Number of Checks

This brings us back to the important *Checks* value. The number of checks is the number of times a rule is assessed after both the `protocol/port/service` identification and `fast_pattern` processes are complete.

The more irrelevant packets that we can exclude with these two steps, the lower the number of checks will be, the more optimal and focused the rule will be, and the less time will be wasted performing in-depth assessment of packets that will never generate an event.

Identifying the optimal content check for fast_pattern

By default Snort selects the string from the longest content keyword (measured in number of characters) for use in the `fast_pattern` test. The design rationale behind this is simple – the longer a content check, the more unique it will likely be, therefore less packets will inadvertently match it. Although this is commonly the case, there are times when the rule writer will have a different opinion based on knowledge and experience.

Looking at the rule in Listing 4, *Example.com* is the longest content check (eleven characters), and by default will be used for the `fast_pattern` check. The other content *CST-ID-001* is however less likely to be found in network traffic, especially if your company name just so happened to be *Example.com*. It is therefore wise to tell Snort to use this better value for the `fast_pattern` check with the `fast_pattern` modifier keyword.

```
content: „CST-ID-001"; nocase; fast_pattern;
```

Following the `fast _ pattern` check, each rule option is then tested in the order that it appears in the rule. Finally the source and destination IP addresses are tested to check if they match those defined in the rule header. Only if every check made is successful is the rule action (such as alert) taken.

If any of the tests fail to match, no further checks are made on the packet against that rule, therefore it is advisable to place quick tests such as `flowbits: isset`, early in your rule options.

Automating Snort rule syntax checks

If you have a large in-house or privately shared rule-set, going back over it and inspecting each rule by hand can be a long and daunting task. To speed this process up, I created an automated tool called DumbPig. DumbPig is a simple syntax checker for Snort rules; it finds common mistakes made by new rule writers and suggests fixes for them. I created the tool a while back to automate my way out of this slow repetitive job.

Listing 4. *A rule with multiple content checks*

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS
                (\
        msg: "Potential Data Leak: Honey Token CST-
                ID-001 over http"; \
        flow: established, to_server; \
        content: "CST-ID-001"; nocase; \
        content: "Example.com"; nocase; \
        classtype: string-detect ; \
        sid: 1000001; rev: 1; )
```

Listing 5. *Dumbpig.pl in action*

```
lward@UbuntuDesktop:~/code/dumbpig$ ./dumbpig.pl -r bad.rules

DumbPig version 0.2 - leon.ward@sourcefire.com
          __,,    ( Dumb-pig says     )
        ~(  oo ---( "ur rulz r not so )
          ''''    ( gud akshuly" *    )

2 Problem(s) found with rule on line 5 of bad.rules

alert ip any any -> any 53  ( \
        msg:"DNS lookup for foo.com using IP proto with port numbers"; \
        content:"baddomain"; \
        sid:1000009; \
        rev:1; \
)
- IP rule with port number (or var that could be set to a port number). This is BAD and invalid syntax.
  It is likely that this rule head is not functioning as you expect it to.
  The IP protocol doesn't have port numbers.
  If you want to inspect both UDP and TCP traffic on specific ports use two rules, its faster and valid syntax.
- No classification specified - Please add a classtype to add a correct priority rating

Rule source sid: 1
alert ip any any -> any 53 (msg: "DNS lookup for foo.com using IP proto with port numbers"; content:"baddomain";
                   sid:1; rev:1)
-------------------------------------
Total: 1 fails over 1 rules (8 lines) in bad.rules
```

References:

- *http://snort.org*
- *http://vrt-sourcefire.blogspot.com/*
- *http://leonward.wordpress.com/dumbpig/*

DumbPig can be found at *http://code.google.com/p/dumbpig/*, it is written in Perl and uses Richard Harman's very useful `Parse::Snort Perl` module to parse a rule-set.

Listing 5 shows the tool in action. In this example dumbpig.pl is reading in a rule file called `bad.rules`, and has identified two problems with rule sid:1000009.

The first problem shown in Listing 5 is in my experience very common. A rule-writer has created a rule for DNS traffic that is commonly found on both TCP and UDP port 53. Rather than create two rules (one for TCP and one for UDP), the rule writer has used the IP protocol in the Snort header, but has also specified a port number. Because the IP protocol doesn't have ports (port numbers are a transport layer construct), this value is ignored. The end result is that every packet regardless of port will be checked for this content. This is very sub-optimal to say the least. The second problem with this rule is that it is missing extra analyst data that will provide more value to any alerts raised.

Summary

The Snort rule language is simple to pick up, and in a similar way to any other language it is easy to fall into some bad habits. Hopefully this article has introduced some simple suggestions that will improve any of your in-house IDP rules in respect to their performance and usefulness.

LEON WARD

Leon is a Senior Security Engineer for Sourcefire based in the UK. He has been using and abusing Snort and other network detection technologies for about ten years and hates referring to himself in the third-person. Thanks go to Alex Kirk (Sourcefire VRT) and Dave Venman (Sourcefire SE) for sanity checking this document.

Insecure Websites in DMZ Still Pose a Risk

Level of Trust

Normally, a website is considered to be a part of the untrusted outer perimeter of a company network infrastructure. Hence, system administrators usually put a web server in the DMZ part of a network and assume the information security risk from the website to the network is mitigated. However, several industry security standards have been imposed to protect the public infrastructure such as webs servers and name servers in addition to the services directly subjected to the standard application scope. Why is it so important to protect your website even if it is not closely connected to your critical data infrastructure?

Social Impact

Humans are the weakest link in the chain of a company's security. The experience gathered during more than 5 years of penetration testing shows that almost no large-scale companies can resist a social-engineering attack vector. In companies which have more than 30 employees, a penetration tester or a real intruder can pretext, deceive, and easily persuade at least 10% of the available employees to open an attachment or follow a link to the malicious website containing an exploit pack and a viral payload. Basic countermeasures include restricting network access to all websites but whitelisted sites, which includes your own website, or simply educating the employees. So, what happens when an intruder gains access to the website? The following list highlights what can be done with a web server located in DMZ:

- Inject an exploit pack and payload into the main page or create malicious pages
- Send spam and scam letters to the company employees inviting them to visit a malicious page at the website
- Install a rootkit and sniffer to maintain access and get all password inputs by system administrators or website maintainers
- Modify links from legitimate sites to malicious ones, for instance, to redirect *Internet bankin* link to *http://ibank.y0urbank.ru* instead of *http://ibank.yourbank.com*
- Pivot client-side payloads through the web server in the case of networks with restricted Internet access

On the 'Net

[1] OWASP Testing Guide – *http://www.owasp.org/index.php/Category:OWASP_Testing_Project*
[2] OWASP Development Guide – *http://www.owasp.org/index.php/Category:OWASP_Guide_Project*
[3] Informzaschita JSC (QSA, PA-QSA) – *http://www.infosec.ru/en*

This list includes only those risks related to successful network penetration. In addition, there are business image risks such as *defacing*, modifying sensitive public information (e.g. exchange rates at bank's website, payment credentials at some charity company's website, phone numbers etc.), or denial of service by deleting everything and bringing the web server down.

Ways to Protect

There are several methodologies to assess website security and mitigate risks connected with the website. One of the most popular is the OWASP Testing Guide [1], which includes more than 300 checks and addresses almost all known web vulnerabilities. The PCI Data Security Standard refers to the top 10 most widespread vulnerabilities in the software, called the *OWASP Top Ten*, and is a basic requirement for any website dealing with credit card payments. For developers, there is also a Development Guide [2], the goal of which is to prevent mistakes affecting security.

To companies willing to protect their websites and networks, Informzaschita offers the following services:

- Complete website assessment according to OWASP Testing Guide (300+ checks)
- Express assessment according to OWASP Top Ten and deployment of Web Application Firewalls for small businesses or companies falling under PCI DSS requirements
- Complete PCI PA-DSS assessment for companies developing payment applications
- Automated security web and network scanning

MARAT VYSHEGORODTSEV,
Information Security Assessment Specialist at Informzaschita JSC m.vyshegorodtsev@infosec.ru, (+7 495) 980-2345 www.infosec.ru/en

An Unsupervised IDS

False Alarm Reduction System – SMART

Signature-based (or rule-based) network IDSs are widely used in many organisations to detect known attacks (Dubrawsky, 2009). A common misconception about IDSs is that they are Plug-and-Play devices that can be installed and then allowed to run autonomously. In reality, this is far from the truth.

What you will learn…

- The limitations of IDS tuning
- The basic concepts and characteristics of SMART system
- The benefits of SMART

What you should know…

- Basics of Intrusion Detection Systems
- Basic syntax of Snort rules

The need for alarm reduction

Depending on the quality of their signatures, they can generate a significant volume of false alarms. Even when the alarms are genuine, the sheer number that can be generated with aggressive attacking traffic (e.g. Denial of Service attacks, vulnerability scans, malware propagation) can be a problem. This is especially the case if the IDS has been configured to generate an alert each time a rule matches. Tuning is often necessary to address the problems of superfluous and false alarms. However, if done recklessly, it can increase the risk of missing attacks. On another note, when attacks span multiple stages, it would be useful to have a mechanism of aggregating and grouping together all alarms relating to the same activity. This not only enhances detection, but also the efficiency of analysing and validating alarms.

In order to address the issues above, alarm reduction systems are needed. Alarm reduction is a process that analyses the intrusion alerts generated by IDS, filters the false alarms and then provides a more concise and high level view of detected incidents.

SMART (SOM K-Means Alarm Reduction Tool)

SMART is an automated alarm reduction tool, designed for Snort IDS. It extends the functionality of Basic Analysis and Security Engine (BASE) (BASE, 2009), a popular front-end for Snort, by providing a more holistic view of detected alarms. The system comprises of two stages; the first stage is responsible for removing superfluous alarms, whilst the second stage distinguishes true from false alarms, by observing trends in their occurrence. The idea behind SMART is not to replace human analysts, but to inform alarm validation and IDS tuning by identifying the most relevant candidates (alarms and rules) for review.

Figure 1 depicts the proposed classification model. Specifically, it shows that data is collected from IDS sensors, and stored in a database. The system then retrieves the data from the database and classifies them by extracting the attributes from the alerts and feeding them into the unsupervised SOM-based clustering system.

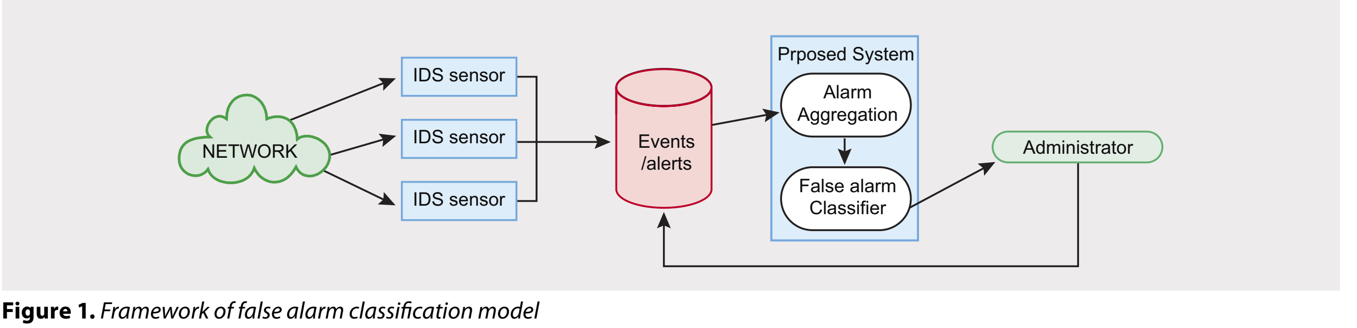


Figure 1. *Framework of false alarm classification model*

The classification process consists of the following phases:

1. Feature extraction – The system uses several attributes extracted from the alert database, which are considered effective to correlate alerts generated from a single activity. The extracted data are then normalised since the value of the data are varied depending upon the type of attributes used.
2. Alarm aggregation (first stage correlation) – Given a set of input vectors from the first phase, the system is trained unsupervised in the second phase to map the data so that similar vectors are reflected in their arrangement. The distance between two input vectors is presented on the map, not by their absolute dissimilarity (which can be calculated), but the relative differences of the data properties. The objective is to group alerts from the same attack instance into a cluster.
3. Cluster analysis – The result of the classification is further evaluated to attain a set of attributes from each cluster created in the previous phase (i.e. the first stage correlation). Building accurate and efficient classifiers largely depends upon the accuracy of the attributes, which are used as the input data for the classification. Seven alert attributes (as shown in Table 1) were chosen to represent the value of each input vector in the next classification (i.e. second stage correlation). Two out of the seven attributes, namely the frequency of alarm signatures and the average time interval between the alerts each day were computed. These features are considered to be the most relevant in terms of influencing the magnitude of the alert signatures.
4. Alert classification (second stage correlation) – The final classification is carried out based upon the attributes extracted in the third phase. The main objective of this stage is to label the alerts into true and false alarms, thus reducing the number of alerts before being presented to the administrator.

The underlying architecture of our proposed alarm classifier that illustrates the four phases of the classification process is presented in Figure 2.

Evaluating SMART

In order to evaluate the effectiveness of SMART, a set of experiments has been conducted. The dataset that was used for the experiments was *collected on a public network (100-150 MB/s) over a period of 40 days, logging all traffic to and from the organisation's web server. It contained 99.9% of TCP, and 0.1% of ICMP traffic*. The effectiveness of reducing false alarms in SMART is compared against conventional tuning methods.

False alarm rate by tuning

Tuning is a popular technique of reducing false alarms, and it is based on the adaptation of the IDS configuration to suit the specific environment where the IDS is placed (Chapple, 2003). This often involves modifying preprocessors, removing (or modifying) rules prone to false alarms, or modifying variables.

The first phase of the experiments involved running snort in default configuration and validating the generated alarms to identify the most suitable candidates for tuning. The following three rules were the most suitable, as they triggered the most false alarms:

WEB-IIS view source via translate header

This event is categorised as web application activity, which targets the Microsoft IIS 5.0 source disclosure vulnerability (Snort, 2010c). Surprisingly, this signature alone had accounted for 59% of the total alerts, in which approximately 1,970 alerts were generated per day by this event. Although the signature was created to detect a Microsoft IIS source disclosure vulnerability exploitation attempt,

Table 1 *The interpretation and data collection methods of the alarm attributes for second stage correlation*

ALERT FEATURES	DESCRIPTION
No of alerts	Total number of alerts grouped in one cluster
No of signatures	Total number of signature type in a cluster
Protocol	Type of traffic from event triggering the alerts
Port number	The service port number. Indicates if the alarm contains a well-known port number, or unknown service ports.
Alert priority	Criticality of the alerts. There are 3 types of alert priority, namely 1st, 2nd, and 3rd. If multiple signatures are found in a cluster, the priority value for each signature could be added together.
Time interval	Time interval between events* from a particular signature
No of events	The number of events1 in which a particular alert signature is triggered within a day

*One event equals to a number of alerts from a single signature, which are triggered by a particular activity.

in this case all generated alarms were related to normal traffic.

When examining the snort rule, it does not seem proficient enough to detect this type of event. It appears to be very loosely written, by searching for a particular string in the packet payload (in this case, *Translate: f*). Since the *Translate: f* is a valid header used in WebDAV application (WebDAV, 2001), the rule tends to trigger a vast volume of alarms.

If the rule gets modified to search for the *GET* command in the content, it is likely that false alarms would be reduced. The attack is launched by requesting a specific resource using HTTP GET command, followed by *Translate: f* as the header of HTTP request. In this case, a tuning can be performed by modifying the signature rule to:

WEB-IIS view source via translate header – Tuned signature rule

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-IIS view source via translate header"; flow:
to_server,established; content:"GET|20|"; content:
"Translate|3A| F"; distance:0; nocase; reference:arachnids,
305; reference:bugtraq,14764; reference:bugtraq,1578;
 reference:cve,2000-0778; reference:nessus,10491; classtype:
web-application-activity; sid:1042; rev:13;)
```

Indeed, when snort was run again with the modified ruleset, this rule had effectively eliminated 95% of the initial false alarms.

WEB-MISC robots.txt access

Although this event is raised when an attempt has been made to directly access *robots.txt* file (Snort, 2010b), it can also be raised due to legitimate activities from web robots or spiders. A spider is software that gathers information for search engines by crawling around the web indexing web pages and their links. *Robots.txt* file is created to exclude some pages from being indexed by web spiders (e.g. submission pages or enquiry pages). As web indexing is regular and structurally repetitive, this activity tends to cause a superfluous amount of alerts. *In this study, approximately 23% of total alerts (approximately 750 alarms per day) were related to this activity, and they were all false alarms*.

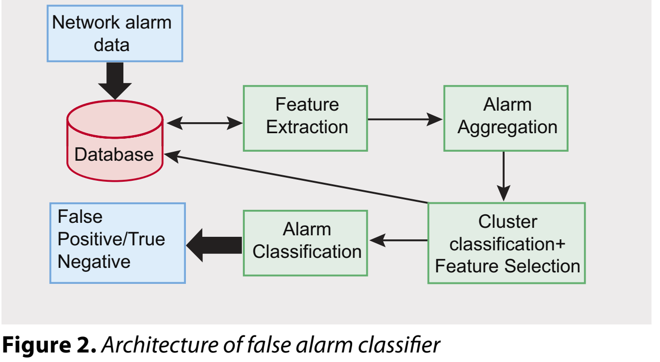


Figure 2. *Architecture of false alarm classifier*

In order to modify this rule to exclude normal web spider activity, the source IP addresses would need to be examined, in order to verify their authorisation in accessing the *Robots.txt* file. This approach, however, seems to be hardly feasible to deploy. Of course, identifying all authorised hosts from their source IP addresses is impractical and dangerous for exploitation. Specifying such a large number of IP addreses can be a problem. Also, the mere fact of allowing specific hosts to access this file could be exploited in order to bypass detection.

As such, event thresholding was used instead (Beale and Caswell, 2004). As *robots.txt* access requests generate regular and repetitive traffic, a *limit* type of threshold command is the most suitable tuning in this case. Such a threshold configuration would be as follows:

```
threshold gen_id 1, sig_id 1852, type limit, track by_src,
count 1, seconds 60
```

The rule logs the first event every 60 seconds, and ignores events for the rest of the time interval. The result showed that approximately 10% of false alarms had been effectively reduced. This indicates that tuning can only reduce a very insignificant number of false alarms from this event.

ICMP L3Retriever Ping

ICMP L3retriever Ping is an event that occurs when ICMP echo request is made from a host running L3Retriever scanner (Snort, 2010c). Quite a few alerts were generated from this event; contributing to 8% of the total alerts. This figure indicates that approximately 250 alerts were generated by this rule every day. Surprisingly, there were no malevolent activities detected following the ICMP traffic. In addition, normal ICMP requests generated by Windows 2000 and Windows XP are also known to have similar payloads to the one generated by L3Retriever scanner (Greenwood, 2007). In view of this issue and given that no suspicious output detected following these ICMP requests; these alerts were labelled as false positives.

The only method that can be deployed to suppress the number of false positive triggered from this event is by applying event suppressing or thresholding command. Instead of using *limit* type of threshold command as previous signature, this rule utilised *both* type of command to log alerts once per time interval and ignore additional alerts generated during that period:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP
L3retriever Ping";icode:0 itype:8; content:"ABCDEFGHIJKLM
NOPQRSTUVWABCDEFGHI"; depth:32; reference:arachnids,311;
 classtype:attempted-recon; threshold: type both, track
by_src, count 3, seconds 60; sid:466; rev:5;)
```

The threshold is written to detect brisk ICMP echo requests by logging alerts once per 60 seconds after seeing 3 occurrences of this event. This experiment has also proved that the event thresholding can successfully reduce up to 89% of the false alarms generated by this activity.

Overall, fine tuning has been effective in reducing false alarms. However, several limitations exist:

1. The procedure increases the risk of missing noteworthy incidents – Suppressing the number of alerts generated can also create a possibility of ignoring or missing real alerts. For example, a malicious user can hide his/her action within the excessive number of alerts generated by using a spoofed address from web spider agent. In fact, looking for an overly specific pattern of a particular attack may effectively reduce the false alarms; however, this method can highly increase the risk of missing its range. A skilful attacker can easily alter and abuse the vulnerability in various ways as an attempt to evade the IDS.
2. Tuning requires a thorough examination of the environment by qualified IT personnel and requires a frequent updating to keep up with the flow of new vulnerabilities or threats discovered.

False alarm rate by SMART

The following stage of the experiments involved analysing the snort alerts with SMART. This experiment presents the results of SMART classification, which is run every two hours, using only four hours alerts from the private dataset. This is due to increased memory requirements of running our unsupervised alarm reduction system for larger data. So, instead of running one correlation for the entire data set, SMART runs a correlation over a particular time period (e.g. every one or two hours). Figure 3 shows the maps of the correlations.

The classification reveals that about 78.8% of false alarms have been identified in the first map (left), whilst 96% of them have been detected in the second mappings (right), as shown in Figure 3. Those alarms located in the upper portion are labelled as true alarms, whilst the lower portion is for the false alarms. It is notable that our system has shown promising result in filtering all hectic and unnecessary alerts triggered by the IDS. For example, the alerts from WEB-IIS view source via translate header and WEB-MISC *robots.txt* access signatures, which had caused 82% of false alarms from the entire data.

In addition to the private data set, the system has also been tested using publicly available data set, DARPA 1999. The experiment also shows promising results,

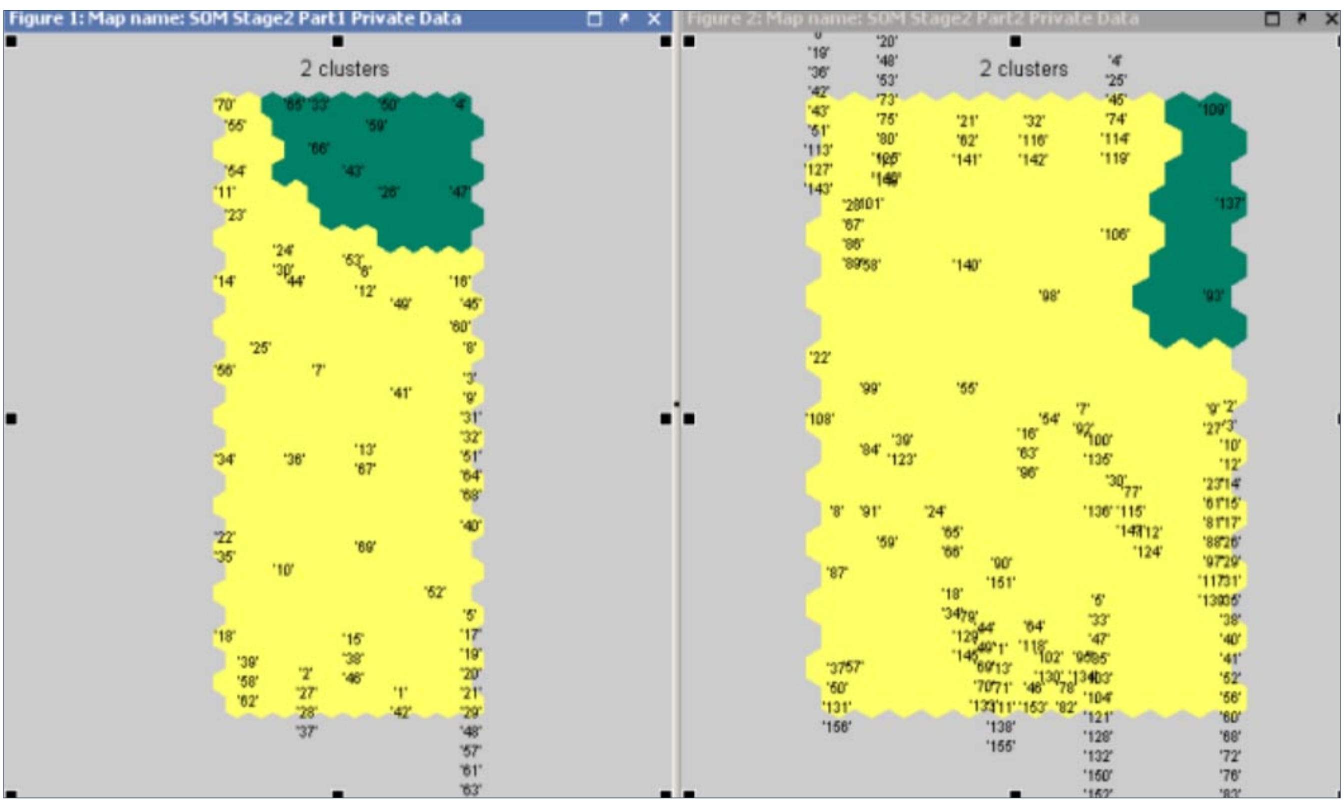


Figure 3. *SMART classification result using private data set*

References

1. BASE (2009), Basic Analysis and Security Engine (BASE) Project, *http://base.secureideas.net/*
2. Beale, J. and Caswell (2004), Snort 2.1 Intrusion Detection, 2nd edn, Syngress, United State of America, ISBN 1931836043
3. Chapple, M. (2003), Evaluating and Tuning an Intrusion Detection System, Information Security Magazine, *http://searchsecurity.techtarget.com/tip/1,289483,sid14 gci918619,00.html*
4. Dubrawsky, I. (2009), CompTIA Security+ Certification Study Guide: Exam SYO-201 3E: Exam SYO 201, Study Guide and Prep Kit, 3rd edn, Syngress, United States of America, ISBN 1597494267
5. Greenwood, B. (2007), Tuning an IDS/IPS From The Ground UP, SANS Institute InfoSec Reading Room, 2007, *http://www.sans.org/reading_room/whitepapers/detection/tuning-ids-ips-ground_1896*
6. Snort (2010a), WEB-IIS view source via translate header, *http://www.snort.org/search/sid/1042?r=1*
7. Snort (2010b), WEB-MISC robots.txt access, *http://www.snort.org/search/sid/1852?r=1*
8. Snort (2010c), ICMP L3Retriever Ping, *http://www.snort.org/search/sid/466?r=1*
9. WebDAV (2001), WebDAV Overview, Sambar Server Documentation, *http://www.kadushisoft.com/syshelp/webdav.htm*

reducing up to 95% and 99% of false alarms in the first and second classification respectively. The system appears effective in filtering the false alarms triggered by a noisy traffic such as ICMP traffic (ICMP Ping and Echo Reply) and web-bug alerts, which have formed the highest number of false alarms.

Overall, SMART has been effective in detecting false alarms, such as the redundant and noisy alerts raised by ICMP traffic. In fact, it is also proved that the system outperforms the traditional tuning method in filtering the WEB-MISC *robots.txt* access alerts. In other words, the issue of subjective rule suffered by common tuning method can be addressed using the proposed system. More than 90% of false alerts from WEB-MISC *robots.txt* access can be identified by SMART.

Summary

Performing a fine-tuning to reduce false alarms is not a straightforward task. If not done properly, there is a possibility that the system might miss real attacks. SMART is an automated alarm reduction system, which helps filtering false alarms generated by Snort IDS. It has advantages over the conventional tuning method. Unlike tuning, the system does not require any prior knowledge of the network and protected systems. In addition, it provides a higher level of alert information to the administrator by aggregating alerts from the same attack instance and validates the accuracy of Snort IDS.

GINA TJHAI

She holds a BSc Computer Science from the University of Wollongong, Australia (2005), and an MSc in Information System Security from the University of Plymouth, UK (2006). She is currently a PhD candidate in the Centre of Security, Communications & Network Research at University of Plymouth, UK. Her current research interests include network intrusion detection and prevention, pattern classification, neural network and data mining.

MARIA PAPADAKI

Maria Papadaki is a lecturer in Network Security, at University of Plymouth, UK. Prior to joining academia, she was working as a Security Analyst for Symantec EMEA Managed Security Services (MSS), UK. Her postgraduate academic studies include a PhD in Intrusion Classification and Automated Response (2004), and an MSc in Integrated Services and Intelligent Networks Engineering (2000), University of Plymouth, UK. Her research interests include intrusion prevention detection and response, network security monitoring, incident prioritisation, security usability, and security education. Dr Papadaki is a GIAC Certified Intrusion Analyst, and is a member of the GIAC Advisory Board, as well as the British Computer Society. Further details can be found at www.plymouth.ac.uk/cscan.

Acknowledgments

The authors would like to acknowledge the contribution of Prof Steven Furnell and Dr Nathan Clarke in the work presented in this paper.

Content modifiers: Keep it Specific

Without going off the deep-end here and discussing every single Snort rule keyword, I just wanted to touch on a few modifiers that people sometimes misunderstand.

What you will learn…

- wrting better Snot rules
- content modifiers
- how to improve the analysis process

What you should know…

- Good knowledge about TCP/IP networks
- Packet Analysis
- Using and configuring Snort

They aren't difficult, and hopefully after this explanation and a few examples, I can clear some of the air around these modifiers.

The modifiers that I am talking about are:

1. Offset
2. Depth
3. Distance
4. Within
5. nocase
6. http_uri
7. rawbytes

These modifiers are not keywords of themselves, but rather they apply as modifiers to another keyword. That keyword is *content*.

The content keyword is one of the easiest pieces of the Snort rules language as all it does is look for a particular string. So for instance if I wanted to look for the word *joel* within a packet, a simple:

```
content:"joel";
```

would allow me to do that. The interesting part comes into play when you want to specify where inside of a particular packet you want the string *joel* to be looked for. If you are running just a plain content match with a simple string, and not specifying where in the packet to look for that string, your Snort instance will receive a ton of alerts, and then you, the analyst, are stuck looking through all of those alerts to try and pick out the alert that is needed. While a content match for *joel* might be pretty unique on most networks, it will occur a bunch on mine.

Offset

Offset is defined in the Snort manual as:

The offset keyword allows the rule writer to specify where to start searching for a pattern within a packet.

So, given a certain packet, Offset tells the content match it's modifying where to start looking, given an offset from the beginning of the data payload of the packet (see Figure 1).

In the above example, if I wanted to find the word *GET* (highlighted). I would write:

```
content:"GET"; offset:0;
```

Meaning, start at the beginning of the data payload of the packet (*offset:0;*) and find the word GET. Now, in this example, the word *GET* is at the very beginning of the packet making the search very easy. However, if I wanted to match on the word *downloads* that is found a bit later in the above screenshot, I could still start my content match at the beginning of the payload (*offset: 0;*) but the content match would be more accurate and less computationally expensive if I were to make the offset more accurate.

```
content:"downloads"; offset:13;
```

Would tell Snort to start looking for the word *downloads* at the 13th byte in the data portion of the packet. So, what if I chained these two together?

```
content:"GET"; offset:0; content:"downloads"; offset:13;
```

In other words, start looking for *GET* at the beginning of the data payload of the packet, and start looking for the word *downloads* at the 13th byte of the packet. Now, why would I do this? This example tells Snort, after the first content match, go back to the beginning of the packet, move over 13 bytes and then start looking again for a second content match. There are several things wrong with this example, -that I did on purpose.

First off, if you are at the first content match in a Snort rule, or a content match you want to start at the beginning of the packet, you don't have to write *offset:0;*. Any content match that doesn't have a modifier after it automatically starts at the beginning of the data payload portion of the packet by default. *Offset:0;* is implied for this type of match. Second, and a:

```
>Common Misconception<
```

Some tend to think that if they stack two contents next to each other, that Snort will look for those contents in the order they are provided. For example, if I were to write:

```
content:"GET"; content:"downloads";
```

Some people generally think that in the above example, that the word *downloads* will have to occur after the word *GET* in the packet. This is wrong. If no modifiers to contents are specified then the order of the matches within a given packet (or stream for that matter) doesn't matter. *downloads* could be first, then *GET*, and the rule will still fire.

So given the above exampled screenshot, if I wanted to force the word *downloads* to occur after the word *GET*, I could use a distance modifier, which I will touch on a bit later.

Depth

Depth is defined in the Snort manual as:

The depth keyword allows the rule writer to specify how far into a packet Snort should search for the specified pattern.

So, given the above example again:

I want to match on *GET* but ONLY if it occurs at the beginning of the packet. Notice when I was describing offset above I said that offset tells Snort where to start looking. Not where to stop. If I don't tell Snort where to stop using a content match, Snort will search the entire packet. If I want to tell Snort where to stop looking for a content match, I have to use something like depth.

So for the above example, if I want to match on *GET* but only at the beginning of the data portion of the payload:

```
content:"GET"; depth:3;
```

Notice some things.

1. I didn't start with *Offset:0;*. Remember, if I am beginning a content search at the beginning of the data payload of the packet, *offset:0;* is implied.
2. Depth counts in positive integers. While offset starts counting at *0* bytes, depth counts in positive integers, *GET* is three bytes long, so my depth is *3*.
3. Depth starts counting from the offset point. Not from the beginning of the packet. While, in the above *GET* example, the offset point IS the beginning of the packet, don't get confused by this.
4. By telling Snort to only look in the first three bytes, if Snort is analyzing millions of 1500 byte packets, only matching on the first three bytes is a significant CPU saver.

Distance

Distance is defined in the Snort manual as:

The distance keyword allows the rule writer to specify how far into a packet Snort should ignore before starting to search for the specified pattern relative to the end of the previous pattern match.

Distance says to us, *okay, relative to the end of the previous content match, when should I start searching*

```
0000  00 18 01 b6 c1 4d 00 23  6c 88 ac 08 08 00 45 00   .....M.# l.....E.
0010  05 dc 5b 4c 40 00 40 06  97 2f c0 a8 01 06 41 c7   ..[L@.@. ./....A.
0020  3f 2b e5 d0 00 50 65 dd  02 73 5e b1 ca 22 80 10   ?+...Pe. .s^.."..
0030  ff ff de 42 00 00 01 01  08 0a 22 14 0a 13 f9 c2   ...B.... .."....
0040  d7 cb 47 45 54 20 2f 63  6f 6e 74 65 6e 74 2f 64   ..GET /c ontent/d
0050  6f 77 6e 6c 6f 61 64 73  2f 35 37 2f 30 38 2f 30   ownloads /57/08/0
```

Figure 1.

for the second content match? So bringing back my previous example:

```
content:"GET"; depth:3; content:"downloads";
```

If I were to do this:

```
content:"GET"; depth:3; content:"downloads"; distance:0;
```

That by itself would force the content match *downloads* to occur after the *GET* content match. Doesn't matter where (*distance:0;*), just as long as the pattern match is AFTER the first one. However, if I wanted to be more specific and more specifically match on the screenshot that I provided above:

```
content:"GET"; depth:3; content:"downloads"; distance:10;
```

This says to the Snort engine, *match on GET, in the first 3 bytes of the data payload of the packet, then move 10 bytes relative to the end of GET and start looking for downloads*.

Notice I said start looking. Not limited to. Kinda like putting an offset without a depth there… so we have within.

Within

Within is described in the Snort manual as:

The within keyword is a content modifier that makes sure that at most N bytes are between pattern matches using the content keyword.

Within allows you to specify a range between content matches, it also allows you to tell a second (relative) content match where to stop.

So, using the content matches we've built already:

```
content:"GET"; depth:3; content:"downloads"; distance:10;
```

The only problem here is *downloads* is being searched for in the entire packet, except for the first 13 bytes, essentially. How can we search for *downloads* only in that specific spot? Within.

```
content:"GET"; depth:3; content:"downloads"; distance:
                    10; within:9;
```

Match on GET, in the first 3 bytes of the data payload of the packet, then move 10 bytes relative to the end of GET and start looking for downloads, however, downloads must occur wholly within the next 9 bytes.

Could I say *within:10;*? Yes, I could, and then *downloads* could be found in it's present position, or if there was another byte in front of the actual content match.

Also notice that within, like depth, works in positive integers (distance starts counting at *1*).

nocase

Finally, let me discuss nocase. nocase, or *No case*, simply tells Snort to ignore case sensitivity for the content match specified. nocase doesn't make the Snort engine work any harder in the grand scheme of things, and it's very handy for being able to make sure your rules do not get bypassed.

Example?

Let's say I wanted to match the above screenshot, no matter what. Well, if I was an attacker, and I came to your webserver trying to access your *downloads* directory, as the rule is written, I could pass my *GET* string as lowercase *get* or mixed case *GeT*, and depending upon your webserver, it might accept it, and I have effectively bypassed your rule.

The easiest thing to do with this type of evasion is to use a nocase; statement.

```
content:"GET"; depth:3; nocase; content:"downloads";
                   distance:10; within:9; nocase;
```

So, I want you to notice a few things:

1. We went from very generic to very specific, your use case will vary.
2. Modifiers to contents come AFTER the content match and not before, they won't work, don't try it.
3. Offset goes with Depth, distance goes with within. Don't mix them.
6. http_uri

http_uri is described in the Snort Manual as:

The http_uri keyword is a content modifier that restricts the search to the NORMALIZED request URI field.

http_uri, if you are a long time Snort user, is the same as uricontent. uricontent and http_uri both read from the output of the http_inspect preprocessor. The http_inspect preprocessor is responsible for the decoding and normalization of http traffic within the parameters specified in the configuration of that preprocessor itself. Configuration of this preprocessor is just as important as any other preprocessor, as it can either save you a lot of time, or it can cause you to have a lot of false positives.

So, for instance, using the above example again:

If I wrote a rule as above using the keywords and distances I have already laid out: see Figure 4.

```
content:"GET"; depth:3; nocase; content:"downloads";
                   distance:10; within:9; nocase;
```

This effectively looks for the word *GET* and the word *downloads* within the packet, completely skipping over the word *content*. However, if I wanted to match the uri string within the rule, I could write the rule as such:

```
content:"/content/downloads"; nocase; http_uri;
```

Notice two things:

1. I didn't do a content match for the word *GET*. *GET* will occur in most http traffic and thusly is rather pointless. Try to avoid matching on *GET*, *POST*, etc, unless there is an absolute need. Say, for instance to avoid false positives.
2. I don't have any distance, within, offset, or depth statements. The http_inspect preprocessor already knows which portion of the packet is the uri string, so by using the http_uri keyword, not only does Snort only have to match on the uri string portion of the http request, but it also looks for the normalized version of it.

What do we mean by normalized, you may ask yourself. Normalized means, out of all the different types of encoding that http_inspect can decode, or normalize (as of the writing of this document, the number of them is 11). Without the http_inspect preprocessor's normalization, a rule would have to be written for every possible permutation of an uri request string.

Okay, so you take one rule and split it into 11, big deal right? The biggest advantage of http, and it's greatest disadvantage is that for all of the different versions of http encodings, they can be stacked on top of each other! HTTP can stack two unicodes on top of each other and then encode it with UTF-8. That could theoretically turn the amount of rules you would have to write to catch every permutation of the above content match into the billions!

http_uri takes care of all of that for you. For instance, if you want to look for:

```
content:"/content/downloads"; nocase; http_uri;
```

and the malicious user on your network encodes his uri as:

```
"%2fcontent%2fdownloads"
```

Without http_uri as a content modifier, the above method wouldn't normalize the unicode `%2f` into the ascii / and would be able to bypass the rule you wrote.

So when looking for a particular URL string, use http_uri as a content modifier. Not only does it limit Snort on where it has to look, but it makes your rule harder to bypass. Take a look at http_uri and the other 9 http specific content modifiers in the Snort manual, test and use them.

rawbytes

rawbytes is defined in the Snort manual as:

The rawbytes keyword allows rules to look at the raw packet data, ignoring any decoding that was done by preprocessors.

This is a rather simple modifier. The purpose of the rawbytes keyword is to *undo* anything the preprocessors may have done to the encoding or decoding of a packet. If the http_inspect preprocessor was to normalize out `%2f` into /. As a rule writer you might want to specifically search for `%2f` as a content match, to see if someone was attempting to bypass any content filtering systems such as an IPS on the network.

```
content:"%2fcontent"; rawbytes;
```

This example would only match on that particular string, if it occurred on a http_inspect normalized port. Normally this is not something you'd like to do, however, the keyword does exist in the off chance that you'd want to be able to negate any normalization.

Hopefully this helped clear up any confusion surrounding these keywords. For further information, please refer to the Snort Users manual. *http://www.snort.org/start/documentation*

JOEL ESLER

Joel Esler is a Senior Security Consultant at Sourcefire. At Sourcefire, Joel travels the world installing and configuring customer Sourcefire and Snort deployments, performing public speaking engagements, and teaching Sourcefire and Snort classes. Having visited over 100 customers, Joel has configured gear in many of the Fortune 50 companies in several different industries, including Banking, Government, Travel (Airline and Rail), Manufacturing, and SCADA (Power) Environments.

Deploying Snort as WAF (Web Application Firewall)

In today's environment, web applications are becoming a popular attack point with attack agents. Attack agent can be a human attacker or an automated worm.

What you will learn...

- Securing web applications with Snort

What you should know...

- Basic knowledge of Snort
- The Snort rule language
- A basic knowledge of TCP/IP

It tries to explore vulnerabilities over HTTP(S) and exploit it for a given opportunity. The web application landscape is also changing and more complexities are getting added, it provides openings for vulnerabilities and possible exploitations. HTTP traffic is no longer restricted to name-value pairs and traditional HTML only. It has evolved with Web 2.0 and RIA, it allows JSON, AMF, XML and various other structures. It has become a platform for robust and advanced business application hosting and usage. It is imperative to secure business applications against all possible attack vectors and to maintain security of information and access. In this article we will try to understand Snort from HTTP standpoint and how we can protect applications for some of the popular attack vectors like XSS or SQL injections by using it.

Problem Domain

Web applications are having set of different entry points and these entry points are attacked and vulnerabilities are discovered by an attacker. It is possible to access applications using HTTP with many different ways and entry poin ts to the application can be of different types as shown in the Figure 1.

These entry points can be mapped to internal execution of the code and if validations are not in place then it leads to a potential vulnerability as shown in figure 2.

Examples of vulnerabilities

URL for SQL Injection – *http://192.168.100.50/details.aspx?id=1.* In above case *id* parameter is vulnerable to SQL injection so it is possible to attack this parameter and gain access to back end database.

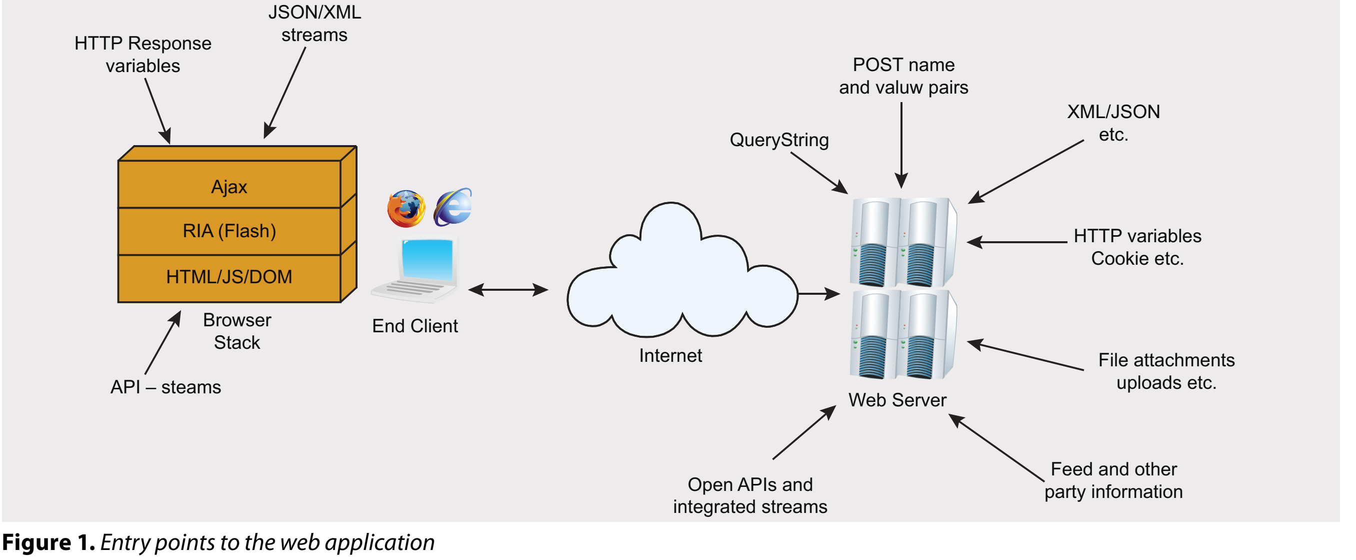


Figure 1. *Entry points to the web application*

URL for XSS – *http://192.168.100.50/search.aspx?search=%3Cscript%3Ealert(%27hey%27)%3C%2Fscript%3E&submit=Search.* In the above URL where we have a parameter called *search* which is vulnerable to XSS, it's possible to inject a script tag and make it executable at browser's end. Both of the above vulnerabilities are exposed since input validations are not in place and malicious content can be injected. Let's try to see some solutions to fix this type of vulnerabilities to secure the application.

Solution to address validation problem

It is possible to resolve some of these vulnerabilities by following two ways.

a) Long term and permanent fix – if developer fixes the problem by providing proper input validations then it's a permanent fix to the problem.
b) Short term and quick patch – it is possible to filter http traffic and dropping the requests with vulnerable payloads, this is not a fix in the source code but does not allow an attacker to inject malicious payload into the vulnerable parameters.

Solution (b) can be implemented by *Web Application Firewall* (WAF). There are different ways WAFs can be put in place, it can run as device or implemented on host. Let's see how Snort can be used to filter http traffic and can act as first line of defense for some of the popular attack vectors.

Solution Deployment Scenario

The deployment for this solution, as expected, requires SNORT which works in combination with the Linux operating system. Before we discuss SNORT used as a Web Application Firewall, Therefore there are two main alternatives: If the application to be protected is hosted on a distinct system, we can set up a tunnel to the hosting system via the Linux system where our customized SNORT-based Web Application Firewall can be run. This provides a mechanism of the web application and its security to be set up across more than one physical system. This can be depicted as follows for purposes of clarity:

The tunnel depicted above has been shown to be created by the proxy HTTP server, SQUID so as to facilitate modularization of the firewall from the actual system where the application is hosted. SQUID can be made to run using the following for configuration:

```
Path: /etc/squid/squid.conf
http_port 192.168.100.8:80 accel defaultsite=192.168.100.50:80
cache_peer 192.168.100.50 parent 80 0 no-query originserver
                    name=myAccel
acl our_sites dstdomain 192.168.100.50
http_access allow our_sites
cache_peer_access myAccel allow our_sites
```

This causes the reverse proxy to be set up because of which our target system 192.168.100.50 is communicated via the current system where SNORT sits – 192.168.100.8 using the port 80 since the requests here are HTTP requests.

The second scenario is evidently much simpler in which the web application is itself hosted on the same system as SNORT. For example when hosted on the Apache, a tunneling proxy system as described above may not be required and the SNORT Firewall can be integrated with the application on the same system.

SNORT

Snort came about primarily as an Open Source *Intrusion Detection System* (IDS). The criteria by which SNORT can successfully flag potential intrusions are signature matching, protocols and anomalies in regular working of a network. *Rules* are the basic requirements for any such specification of criteria to be done for flagging intrusions.

An added functionality to this has been that of an *Intrusion Prevention System* (IPS) which not only informs about potential intrusions but also prevents them. This feature comes under SNORT-Inline integrated with *IPTables*. *IPTables* queues the messages in the traffic through the network that have matching protocol or other signatures according to rules. After this, SNORT-Inline fetches these selected queued packets based on the configuration and Rules files. These are then processed by actions such as packet dropping after

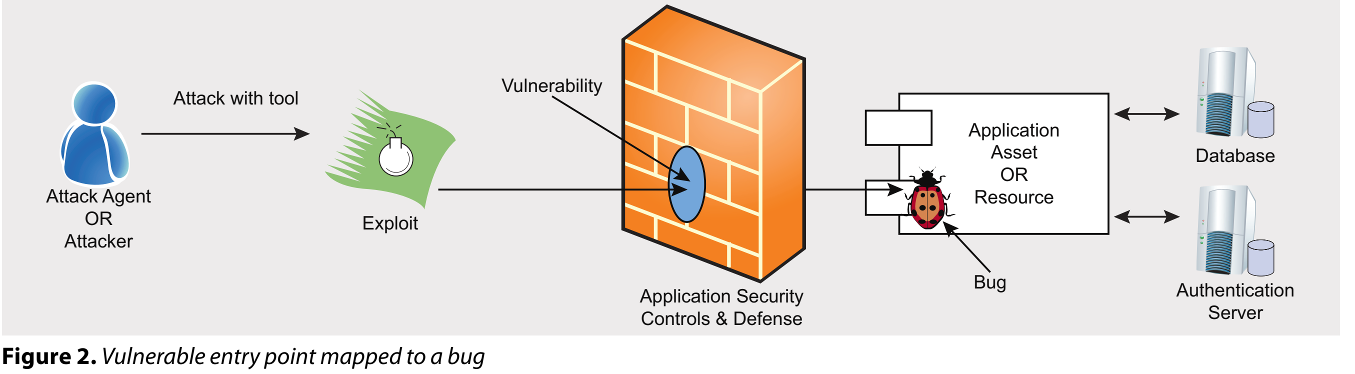


Figure 2. *Vulnerable entry point mapped to a bug*

alerting the user. Thus, intrusions here are not just detected but prevented too.

SNORT-Inline has an HTTP Preprocessor which essentially inspects HTTP requests and responses. The preprocessor can be set up to allow SNORT-Inline IPS for HTTP traffic as follows:

```
preprocessor http_inspect: global \
    iis_unicode_map unicode.map 1252
preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 8180 } oversize_dir_length 500
```

This code informs HTTP inspect about the Unicode mapping to be performed and the files to be used for the same. Moreover it defines default profiles for Web Servers to respond on the ports 80, 8080 and 8180 TCO or to alert on size exceeding. Once this is done the SNORT Web Application Firewall can be customized by adding rules as required for the target application to be secured.

Rules

Rules to guide the SNORT Inline IPS to tackle intruders may be included by setting them up in snor.config and also by including them as rule files by:

```
include $RULE_PATH/sql-injection.rules
Add rules in etc/snort/rules as sql-injection.rules
```

Rules for SQL Injection Attacks

Here, *any any -> any any* ensures that any traffic through the network (any source to any destination) is handled by this particular rule. Furthermore, the *msg* defines the alert to be used to flag the success of this rule. Proceeding, the content fields are required to define the target area in the web application where this rule must be checked out. For example, as mentioned above the parameter *id=* in the page */details.aspx* is the one that is vulnerable to SQL Injection and hence this area has been defined by the content fields here. Finally the Perl Compiled Regular Expressions are used to specify the pattern that must be matched for this alert. The rule above and those given below can consequently be used to prevent the SQL Injections by using the payload for the vulnerable parameter as escaped characters like % or %25 or SQL commands like insert, delete, update. The 'classtype' identifies the type of attack addressed by this rule and the *sid* and *rev* fields are unique identifiers for the rule (including the revising and editing performed on the rules).

Rules for XSS Attacks

For the rules treating XSS below, the significance of the various parts of the rules remains the same as that described above while treating the rules for SQL Injection. The difference here is in the content that the rule works on and the patterns that are matched to safeguard against XSS Attacks. At the vulnerable parameter *search* here, the packets that are picked out are ones with <, > and all its escaped or encoded forms along with other JavaScript event handler function calls such as mouseover, mousemove, change, etc.

Starting SNORT-Inline

Finally, having set up the rule files, the SNORT IPS is all set to be started up. This can be done by:

```
snort -Q -c /etc/snort/snort.conf -l /var/log/snort -v
```

Where,

- -Q is to enable inline mode
- -c is to set the path for the config file
- -l is to set the path for log file
- -v is to enable the verbose mode

A Demonstration Run

An example run after the integration of all the steps mentioned above was then done on the target web application discussed. This successful run (log and drop) can be reviewed as under:

```
[**] [1:9000:5] Cross-site scripting attempt [**]
[Classification: Web Application Attack] [Priority: 1]
10/23-18:51:25.793080 192.168.100.8:34591 -> 192.168.100.50:80
TCP TTL:64 TOS:0x0 ID:12085 IpLen:20 DgmLen:657 DF
***AP*** Seq: 0x1CFC03AE  Ack: 0x8D3DEF67  Win: 0x6B9  TcpLen: 32
TCP Options (3) => NOP NOP TS: 3725432 90169
```

Conclusion

Thus in light of the discussion above, SNORT has emerged as a viable Web Application Firewall. This application for SNORT essentially involves placing an SNORT Inline IPS as an individual component for the Web Application Firewall (SQUID can be used to configure a Reverse Proxy scenario). This positioning is done such that all the traffic for communication with our target application is tunneled via this firewall at all times. The firewall can then be equipped with the 'Web Application Security' aspects by setting up rule files to check the traffic for various categories of exploits at their corresponding vulnerable points.

SHIVANG BHAGAT & RISHITA ANUBHAI (BLUEINFY APPSECLABS)

Shivang Bhagat & Rishita Anubhai (Blueinfy AppSecLabs) – Shivang and Risita are web security researcher and consultant at Blueinfy. Both are involved in web application pentesting and development of tools in the area of web security assessment tools. Both of them have authored a few tools for AppSecLabs at Blueinfy.


Get the free Mobio® app:
www. getmobio.com
market
Available on the iPhone
App Store
Make a Purchase
Chiefs Hockey Game
YOUR PERSONALIZED OFFER
Jerseys
TICKETS
Game Tickets
Make a Purchase
Concession Stand
FOOD
Nachos
Fries
Cheese Burger
Burger
Peanuts
BEVERAGES
Mobio
SECURE IDENTITY MANAGEMENT
MOBILE PAYMENTS
EXCLUSIVE PROMO ACCESS
EMPOWERING IDENTITY
Find us on @MobioID
www.mobioid.com

Nagios founder Ethan Galstad speaks

at the Conference on Nagios 2011

The yearly edition of the Conference on Nagios and Open Source-Monitoring, hosted by Würth Phoenix will take place in Bolzano/Italy on the 12th of May 2011

Following the great interest in 2010, the successful series of an international Conference on Nagios and OSS Monitoring will continue also in 2011. The organization team of Würth Phoenix spares no efforts to top last year's agenda and bring international Nagios and OSS Monitoring experts to Italy. The father of Nagios, Ethan Galstad, and ntop founder Luca Deri have already confirmed their participation. Contacts with other nameable Open Source Monitoring experts are in progress. The sessions are intended to go deep into actual business trends and best practices without missing out the technical hard facts.

Call for papers: Submit your proposal until the 31st of January 2011

If you've got something to share, to tell, to introduce or to show regarding Open Source Monitoring please take the time and submit a presentation to the organization team. You are invited to come up with a brief abstract of your presentation. You should also provide some information about yourself, your experience with Open Source software and the specific aspects of monitoring solutions based on Nagios. Just write an E-Mail at *info@wuerth-phoenix.com*.

Why should you attend

You´re invited to share experiences and hear some of the most current OSS developments. Beside the keynote speeches also a panel discussion will be part of the agenda to highlight the changing market relevance of Nagios and other well known OSS projects. Last but not least you will meet a lot of friendly and like-minded people there.

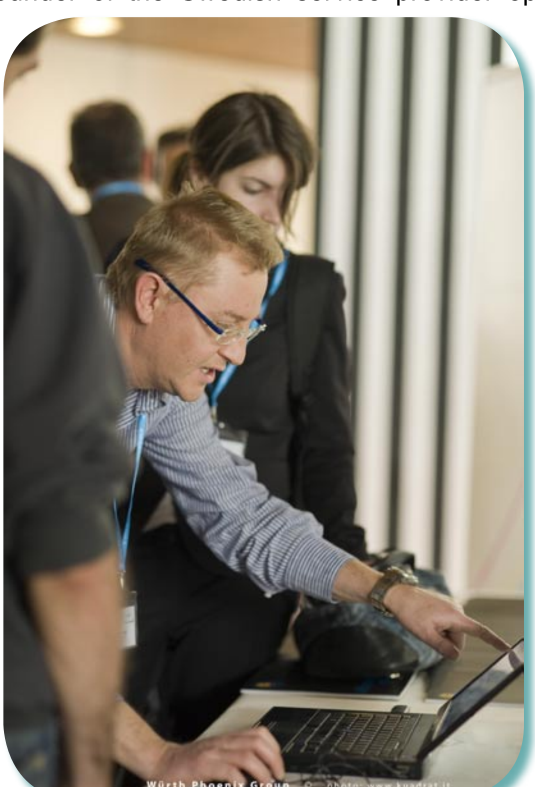
Subscribe as participant

Also in 2011 the attendance at the Conference including networking lunch will be for free. We welcome every passionate of Open Source, from sys admins to developers, from programmers to IT managers and entrepreneurs. The number of participants is limited. Just confirm your participation and register already now at *www.wuerth-phoenix.com/nagios*.

Last year´s edition: A brief look back

In 2010 400 visitors from several countries gathered together in Bolzano/Italy to closely experience the new developments and the future directions of different monitoring solutions.

Renowned international speakers ensured a many-sided agenda, which was mainly directed to a broad audience of businessmen and engineers. Ethan Galstad introduced the developments of his worldwide established monitoring standard Nagios. Reliable solutions for more system stability in the IT business should not engulf massive budgets for licensing costs. Jan Josephson highlighted the internationally growing request of Open Source software solutions in the business area. The founder of the Swedish service provider op5 referred mainly to the strong market position in the Scandinavian countries. The mix of Open Source projects and commercial services has long stood the test. Finally Luca Deri from the University of Pisa suggested some precious tips for practicing IT monitoring. He spoke of the application areas of his worldwide used monitoring program ntop. Insights from the development community were given by Michael Medin from Sweden and Reinhard Scheck from Germany. They provided a review into the development focus of Nagios and Cacti directing their speech mainly to advanced users. Speakers of famous industrial companies, like the Savio ITEMA Group, talked about choice and implementation of Open Source solutions in worldwide acting enterprises.



In addition to the lectures which were followed by direct feedbacks by the audience, a panel discussion followed. Experts as well as visitors interchanged opinions on possibilities and future scenarios of a stronger cooperation between Open Source Projects and proprietary producers like Microsoft, HP or IBM.

Watch the interview with Ethan Galstad:
http://www.youtube.com/watch?v=GtvpE3Ug-KQ
Get some impressions of this year´s edition:
http://www.youtube.com/watch?v=9UOveAFH1Ns&feature=related

Are You Getting the Most out of your IPS?

Picture this: a multi-billion dollar global telecom giant has invested millions of dollars into building a state-of-the-art Security Operations Center.

What you will learn...

- why problems with IPS may occur
- how to assess the performance of your IPS
- how to optimize your IPS

What you should know...

- basic knowledge of TCP/IP
- basic knowledge of IPS

They have a huge display screen being monitored 24/7 by a team of specialists who – so we are told – have received extensive training in the specific technologies used, as well as in the overall *incident management framework*. They've deployed a high-end *intrusion prevention system* (IPS) which feeds into their *Security Incident Management* (SIM) system.

A review of the procedures and *Service Level Agreement* (SLA) of the SOC team signed with the rest of the business reveals that they are ready to respond 24/7 and have committed that within 2 hours of a serious attack they will respond to any serious attacks.

On paper it all looks impressive and too good to be true.

Putting it to the test

For starters, we decide to launch an unannounced, but highly noisy scan on their public IP addresses. But just to make it a little interesting, we do this over the weekend. The SLA assures us that the SOC is ready to respond on a 24/7 basis within 2 hours to all high criticality events.

When we come in to their office on Monday, we expect to see a flurry of activity having happened at the SOC. But to our not-so-great surprise we find that there's not even an incident recorded over the weekend. And this is after we've launched Nessus scans with all plugins selected and making no attempts at IPS evasion or other forms of stealth.

To give them further benefit of doubt, we also run internal scans on critical servers and wait to see if this raises any alarms at the SOC. But nothing seems to really shake the SOC out of their reverie.

Some tough questions

At this stage, it is pertinent for us to confront the customer with the ground reality, and begin an in-depth investigation as to why the SOC is unable to justify its existence? In spite of the millions of dollars sunk into it in terms of capital expense, and the monthly expenditure on a supposedly well-trained team of people, why are they not reacting to obvious internal and external intrusion attempts?

The answers that emerge from this audit reveal issues that exist with a number of companies that deploy Intrusion Detection and Intrusion Prevention Systems and build their SOCs, but fail to get the maximum value out from them.

Problem #1: IPS is not properly tuned

Many organizations think that once the *system integrator* (SI) has come in and deployed an IPS, that's about it, and everything should work perfectly fine. A lot of organizations quickly learn that this is not the case, and discover that their IPS is now just a garbage collector. It is being swamped with thousands of alerts per minute and it is simply impossible to make any sense of what is actually happening on the network.

When they now seek to tune the IPS the system integrator is no longer in the picture or the IPS is an out-dated version or there's a new on-going project to implement a SIM, which will solve all these problems, or some story or the other.

What is important to realize is that the SI should've been asked to not just implement an IPS that works, but an IPS that works properly. One that has been fine-tuned to weed out as many false positives as is practical, where the dashboards have been tuned to reflect as true a picture of the network as is possible, and where the reports show the actual trends of attacks on the network.

Even more importantly, IPS tuning is not a one-time job – it is a constant effort and the organizational team should take over this responsibility and expend effort towards ensuing the IPS is constantly being tuned.

A weakly configured IPS is the #1 reason why a lot of IPSs continue in monitor-only mode many months or even years after deployment.

Problem #2: Lack of trained resources

One of the more humorous events happened when we decided to evaluate the capability of the SOC team members. The guys in the main shift were well-trained and certified on the specific IPS, and we came away duly impressed. On a hunch, we requested to speak with the SOC team guys in the off-peak hour shifts. During one such interview, the person we were interviewing admitted that he wasn't aware of an IPS or SIM. His job description according to him was as a NOC team member monitoring the up-time of the links and escalating calls to engineers if he saw any problems appearing on the screen. When we asked him his experience on the IPS, his response was that he had not been asked to look at the IPS alerts or been trained on what to do if he saw a series of red-line alerts cropping up.

This lack of trained resources was further evidenced by the lack of any responses to our weekend scans. A deeper investigation into the contract between the company who had supplied the resources and the Telco revealed that the off-peak resources needed to be paid much more if the Telco wanted SOC-level guys. So the Telco had settled for security incident analysts during normal working hours, and NOC-level guys for the other shifts.

Therefore, to cut down on the operational expenditure, the huge capital expenditure on the IPS and the SOC was actually being wasted.

Very often we find SOC team members do not even understand the basics of TCP/IP. They are essentially trained to look out for red/amber/green and react as per a fixed standard procedure. The very least that a SOC analyst should know is how attackers operate, what a reconnaissance phase consists of, how web applications are compromised, etc.

Problem #3: The SIM Implementation/IPS Upgrade/XYZ Project will solve our problems

This is a typical ostrich-head-in-the-sand approach, wherein all problems related to the IPS and the capability of the SOC team to respond are brushed away under the next new big project that the organization is about to implement. We have seen numerous instances where a *Security Incident Management* (SIM) implementation fails (and this constitutes enough material for a completely different article, but here are some of them):

1. The SIM project itself takes so long to implement, during which time the organization is living with an open risk of not being able to detect and prevent attacks on its network
2. The IPS doesn't integrate with the SIM due to various reasons, or if it integrates, it ends up overwhelming the SIM drowning out alerts from other systems
3. Correlation rules on the SIM don't get created properly – thus the SIM is nothing but a glorified log collector
4. If the IPS it not tuned in the first place, it is simply going to turn out being a case of Garbage-In-Garbage-Out with an additional layer of SIM added on top of it

So whatever is the next big project – it will surely not miraculously clean up an IPS mess. In fact, a messy IPS implementation will only result in a messier SIM implementation!

Problem #4: Weakly configured reports

There are some basic questions that an IPS should be configured to answer in order to show its effectiveness and prove *Return on Investment* (ROI):

1. What are the sorts of trends we are seeing in terms of attacks? For instance, for attacks from the Internet:
   a. Which ports are under attack more than others?
   b. Which IP addresses are under attack more than others?
   c. Overall are we facing more attacks or less?
2. Is our blocking capability effective enough, i.e. how many attacks are being blocked in comparison to the total number of confirmed attacks being detected
3. Of the total number of high-criticality attacks, how many is the SOC team responding to?
4. What is the time duration between attack detection and response?

The answers to these questions reflect the ROI for an organization. The lack of accurate answers reflects that either the IPS is mis-configured or the reporting capabilities of the IPS have not been fully utilized.

Problem #5: Not squeezing the maximum out of the IPS

Most IPS's offer many features which remain unexplored. Snort is a great example of this. Not only is Snort highly flexible, but writing your own rules for Snort is a breeze. Snort also comes with bleeding-edge rules, which are worthwhile keeping a watch for. The commercial version of Snort – Sourcefire comes with a highly impressive (but equally costly) learning engine, which reduces the number of manual resources required to monitor and fine-tune the IPS.

A lot of organizations choose to ignore many of the advance features that their IPS offers, and System Integrators are happy doing a plain vanilla deployment, although the marketing team of the same vendor may have pitched all of the advanced features as *Unique Selling Propositions* (USPs)!

Problem #6: Weak incident management processes

A large number of organizations deploy IDSs and IPSs, yet do not build strong processes and policies around the process of incident management. Some of the critical aspects that get missed out are:

1. What is an incident that requires action to be taken?
2. What action is to be taken for reacting to different types of incidents:
   a. A virus/worm infection
   b. An internal hacking attempt
   c. An internal policy violations
   d. An external hacking attempt
   e. An external port-scanning attempt
3. What is to be done when the investigation into the incident concludes that it is a false positives?
4. What records are to be maintained during and after investigation?
5. Who are the owners of various systems (a list of IP addresses and the corresponding administrators is either missing or not updated)?

Problem #7: Weak metrics and measurement of IPS success

The final weakness present in a lot of IPS implementations is the lack of metrics to measure how well the IPS has been implemented and whether things are getting better or getting worse. In the absence of metrics, there is no way to conclude if the organization's capabilities to detect and block attacks is getting better or not. Some suggested metrics for an IPS and the associated incident management processes are:

1. The ratio of false positives to total alarms
2. The number of high criticality events responded to
3. The amount of time taken between an alert appearing on the console and the response being triggered
4. The number of changes made to the IPS filters or rules
5. The number of unannounced security scans detected by the IPS and the monitoring team

Conclusion

At the end of the day, an IPS is a fairly expensive investment for any organization, not only in terms of hardware and software, but also in terms of the skilled resources that need to be deployed to manage and monitor the IPS. Most system integrators are keen to implement the solution to show that it works, and not in a way that makes it work well. Getting the maximum out of a news Rolls Royce in your garage requires you to not only be trained in driving one, but also in the mechanics of checking its parameters regularly and tuning it. It is not just a simple plug-and-play mechanism. To conclude, the key points that any IPS implementation should bear in mind are:

1. Tuning the IPS to filter out as many false positives as possible
2. Ensuring well-trained resources are deployed to manage and monitor it
3. Not assume that the next big security project will magically resolve all current IPS issues
4. Ensuring the reports and dashboards are properly configured
5. Using all the features of the IPS to the maximum, especially since you've paid for them
6. Putting in place strong incident management procedures around the IPS technology
7. Putting in place an effective metrics framework to ensure you get the maximum bang for your buck!

K. K MOOKHEY

K K Mookhey is the Founder and Principal Consultant at NII Consulting. From a startup capital of USD 5000 in 2001, he has built it into a USD 500,000 practice providing services in IT Audits, Risk Management, Compliance, and Computer Forensics. NII has offices in India and the Middle East, and a client list that includes the United Nations, Saudi Telecom, Dubai Stock Exchange, Atlas Air, Royal Sun & Alliance, and many others. He has written numerous articles on information risk management and these can be accessed at http://www.niiconsulting.com/innocation/articles.html. He has also presented at conferences such as Blackhat (Las Vegas 2004), Interop (Las Vegas 2005), IT Underground (Prague 2005). He is also the author of two books - one on Linux Security and Auditing, and the other on the Metasploit Framework by Syngress Publishing. He blogs at http://www.everydayentrepreneurs.blogspot.com