A quick overview of Snort

Patrik Israelsson
Jerker Karlsson
Guillaume Giamarchi

October 17, 2005

Abstract

As a network administrator, you obviously want your systems secure and up to date to avoid any security breaches. It might be reassuring to know that you've done all you can to prevent attacks - but what knowledge do you have of what attacks actually have *been made, and whether they've been successful or not? An attack, regardless of its success, is something you want to know about, and not only that - you want to know all possible details about it. Where did the attack come from? What tricks was the attacker trying to pull? What did the attacker seem to be after? And in case the attack actually was successful, what damage has been done? Has the attacker 'stolen' sensitive information of any kind?*
Enter the Network Intrusion Detection System. In the ideal case, a properly setup NIDS will help you a great deal with the answers to all your questions. Suspicious activity is recognized and logged for future reference. It's also possible to take action automatically based on the activity, though that's not entirely within the scope of the NIDS itself.

One NIDS that has gained a lot of popularity is Snort. In this essay, we will have a look at how it works, what its main advantages are but also what problems one may encounter in deploying it for use in a network.

1 Background - what is a network intrusion detection system?

The idea of a network intrusion detection system is to have a device of some sort that can 'hear' all the traffic on its part of the network. It listens to this traffic, and based on a set of defined rules, it will trigger an action of some kind on packets that match one of the defined rules. One could think of its functionality as very similar to that of antivirus software - scan content for stuff considered malicious,

and take action.
An example rule might, in common language, be: *"All packets sent on the http port to a webserver on the network, that contain the string 'cmd.exe'"*. Snort's default ruleset has an implementation of this rule - the reason we want to watch out for this is it might imply that an attacker is trying to execute a command shell on a webserver running Windows. We will look into exactly what this rule looks like in Snort's rule format later.

2 An introduction to Snort

2.1 Short history

Snort was created by Martin Roesch in 1998. As most open-source projects, it started out as a small-scale application made just for fun, as an alternative to the full-blown commercial intrusion detection systems. Today, Snort is used by many many, both commercially and privately. All in all Snort has been downloaded from the official site more than 2 million times, and spurs more than 100,000 active users. In its most basic form it can be used as a regular packet sniffer (think tcpdump), a packet logger or - most commonly - a network intrusion detection system.
Snort nowadays is developed by Sourcefire (Martin Roesch's own company) which in turn recently was bought by security giants Checkpoint.

2.2 Usage

Like most NIDSs, it's hard to talk about one specific use for Snort - there are many uses for it depending on the needs of the user, but most commonly Snort is deployed for keeping track of what's going on within a network. In itself, Snort doesn't necessarily provide a good overview as it only does one thing: trigger on specified traffic and take action in some way, where the action most often is the logging of an alert. Therefore, Snort is often used with other systems giving the user an overview of all alerts triggered, ACID[1] being one example.
It's also possible to have Snort interact with other services - for instance, it's possible to use it with iptables or similar firewall software for auto-blocking hosts that seem to be doing things they're not supposed to do.

2.3 Placement of the Snort device

An important decision when deploying a Snort device is where in the network to put it. Inside or outside the firewall? On an Ethernet tap (more on this shortly) where it will be inaccessible but also unable to react to malicious packets, or in

[1] http://www.andrew.cmu.edu/user/rdanyliw/snort/snortacid.html

a firewall/gateway position where its system have higher requirements on being 100% secure and also fast enough to both keep track of the traffic and not degrade network performance?

Placing the IDS outside the firewall will most likely generate a lot more alerts, many of which are not really dangerous since they are stopped by the firewall. None the less, being aware of what kind of attacks are directed at you, even unsuccessful, is sometimes useful.

A common technique is to have Snort run on a device that doesn't actually have an address on the network it's sniffing - a so-called 'Ethernet tap' is used. An ethernet tap is a port on a device (a switch for instance) that mirrors all the traffic passing through the device. This way, when the device running Snort is connected through an Ethernet tap, there is no way for a potential attacker to access the Snort device directly. This means that the integrity of the audit logs are protected so that a successful attacker cannot cover his tracks.

As mentioned earlier, one could also use Snort as part of an active firewall system reacting on potential attacks. In this case, apart from simply logging the packets, Snort can for instance be told to take out the IP address of the potential attacking host and pass it on to the firewall software, telling it to block the host.

3 Rules

3.1 Syntax

Snort has its own processing language used to define rules. Recently, with the release of the v2 series of Snort, regular expression processing has been added to make good rule-writing easier. However, the majority of the rules still rely heavily on the traditional syntax.

The rule we talked about earlier, about catching all packets in web traffic containing 'cmd.exe', would look something like this in Snort's rule format:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-IIS
cmd.exe access"; flags: A+; content:"cmd.exe"; nocase;
classtype:web-application-attack; sid:1002; rev:2;)
```

As this is a very basic rule, most of this is fairly straight-forward. The rule tells Snort to generate an alert and log the packet, if the following condition is met:
A packet is sent from $EXTERNAL_NET (a variable set in the Snort configuration containing a definition of the network outside our own) on any port using

TCP, to $HTTP_SERVERS (again a configured variable) on port 80. The packet has TCP ACK set and contains 'cmd.exe', case insensitive.
If this is true, the packet is logged using the remaining specifications that we see in the rule:
The title for the log entry is "WEB-IIS cmd.exe access", the class is "web-application-attack" and the sid (Snort rules ID) is 1002, revision 2.
The SID, along with the "rev" identifier, is used to identify the triggered rule easily.

Most of the default rules make use of more complex mechanisms, but the syntax remains the same. More advanced options include checking raw binary data, using offsets to check binary values in specific positions in packets, investigating TCP checksums, and much more.

3.2 The art of choosing good rules

Snort is distributed with quite a large amount of rules, ranging from rules checking for shellcode being transferred to a host on the network, to rules taking note of stuff that isn't necessarily harmless by itself but could be worth noting, like anonymous FTP access. The key thing here is of course to make sure one has relevant rules. In some environments, internal FTP access might be highly suspicious and consequently it's a good idea to employ a rule for it, while in other environments FTP access is something that happens all the time - a rule triggering on this will just generate useless alerts that divert the attention from other potential attacks.

Also, it's far from true that just because one has a rule checking for, say, shellcode transfers, one will only get relevant alerts.
For instance, Snort has a default rule called *"SHELLCODE x86 setuid 0"* that simply triggers on the sequence of the hexadecimal values "b017 cd80" in a packet. In an environment where hosts on the network issue large file transfers on a regular basis, this rule will most likely trigger loads and loads of times - because in a sufficiently large binary file, the mentioned sequence is bound to occur sooner or later. We get a highly unwanted noise in the generated alerts. Simply removing the rule isn't a good solution either - a setuid 0 call being transferred over the network is in itself highly suspicious, when it actually is a call. We learn from this that the art of writing a good rule isn't as easy as it might seem at first. The person deciding what rules to use and/or write needs to think about what should be considered normal traffic on his/her network, try to filter out as many false positive cases as possible, and decide whether the hosts on the internal network should be trusted not to do any mischief.

3.3 The importance of being up to date

It's not enough to have Snort (or any other NIDS) in use on the network and leave it at that when the installation is done. Like with any system, the administrator needs to make sure that the NIDS and its rules are relevant and up to date. A new exploit might spread quickly, and without corresponding rules Snort won't even notice it. Again, there are obvious similarities to antivirus software. Also like in the antivirus software case, much effort has been put into inventing new ways of evading the system, exploiting this fact that Snort (like AV software) can only look for what it's told to look for. This 'dumbness' of Snort is a drawback.
Sourcefire, the company developing Snort, are constantly adding, deleting and modifying rules. It's possible to subscribe to updates to Snort rules - it's free for private use but costs quite some money for commercial use.

4 Conclusions

When deploying Snort, it's important to make sure the used rules are relevant and up to date, otherwise the system will be much less efficient - due to low signal-to-noise ratio in the case of a bad choice of rules, and due to Snort missing attacks completely in the case of a Snort system with rules not being updated properly.
Apart from the challenge of choosing/writing good rules for Snort, there is a related disadvantage - since Snort only looks for things defined in its ruleset, it doesn't have the ability to tell what traffic is considered to be normal from each host on the network, and what traffic seems to be out of place. This way, 'normal' behaviour but from the 'wrong' computer on the network isn't noticed unless rules are setup on a host-by-host basis. There are a few systems who have started to deal with this problem, called 'anomaly based intrusion detection systems', for instance ASDIC[2] which is developed in Uppsala.
However there are obvious advantages of using a NIDS, such as Snort, in a network. Properly configured, it gives a good overview of what's going on in the network, and provides a way of automatically logging packets from potential attacks for future reference. With some careful thinking, it can even be used for reacting directly to attacks as they occur.

References

[1] Brian Caswell and Jeremy Hewlett. *Snort Users Manual* (available from http://www.snort.org/docs/)

[2] Brian Caswell, Jay Beale, James C. Foster, Jeremy Faircloth. *Snort 2.0 Intrusion Detection*

[2] http://www.ping.se/eng/asdic.html