Intrusion Detection with Snort

Adam Spencer Garside

Abstract

This article discusses how Snort, a mature Open Source Intrustion Detection System, can be used to monitor anomalies, attacks, and even worm infections on a network.

1. Introduction

Snort[1], a Free and Open Source Intrustion Detection System, or IDS, written by Marty Roesch (Now at Sourcefire) is one of if not the most mature IDS systems in use today. It is used to monitor network traffic and scan for protocol anomalies and packet payload signatures that represent potential attacks, worms, and unusual activities. On non-trivial networks, it helps network administrators identify potentially compromised machines, misconfigured machines, information leaks, and active or passive attacks. An IDS is an invaluable tool in the area of network security.

Snort is usually configured to listen to backbone network traffic by means of an inline hub or switch span port, though it can be used as a host based IDS as well monitoring only traffic coming to and from a single machine. Furthermore, multiple Snort scanners, or taps, can be placed on a network and integrated with a single console monitor. There are a number of monitoring consoles available for a number of platforms but the one highlighted in this article is BASE[2], a PHP based web console that works with snort's MySQL logging output plugin.

In recent years, there has been a push to not only detect but also prevent intrusion and this has spawned a number of IPS (Intrustion Prevention Systems) products. The claims made by vendors shipping IPS products indicate that IDSs are no longer useful. While this is certainly true in some situations, IDS provides data on what is happening and data is better than nothing at all. Fortunately, snort can also be configured as an IPS by using an inline mode that ties into the GNU/Linux iptables or *BSD firewall suites. Also, some firewall products now ship with a 'Deep Inspection' option which is a minimal IPS for small subsets of known attacks.

Before going any further, it must be pointed out that **security is a process, not a product**. Anyone who claims otherwise probably has a Brooklyn Bridge for sale as well. Security requires ongoing vigilance, attention to detail, and an understanding of current trends and exploits. A number of sites exist to document known issues in the Internet community[3] and mailing lists, such as Bugtraq, should be monitored constantly as both vulnerabilities and exploits are actively discussed. With these things in mind, be aware that deploying an IDS takes time, intimate knowledge of your network, and ongoing maintenance. A number of companies can help to alleviate some of the burden, such as Sourcefire[4] and Nitro Security[5]. Both of these use snort internally; the former, in fact, is the company started by the lead developer who created snort and has just been acquired by Checkpoint, a prominent firewall and security product vendor.

This article will cover a number of issues including:

- Snort Tap Placement
- Snort Rules
- BASE (Basic Analysis and Security Engine)
- A Worm propagation analysis example

2. Snort Tap Placement

One of the most critical aspects to consider when deploying snort, or any IDS for that matter, is the location of the tap(s). If critical traffic is not monitored, it cannot be analyzed. Ideal places are:

- Natural Choke Points
- Artificial Choke Points
- Intranet Trust/Untrust zone boundaries

2.1. Natural Choke Points

Natural Choke Points are areas where the network topology creates a single traffic path, such as an inbound Internet connection. Placement at points, such as firewalls, is ideal. Depending on how paranoid you want to be, an IDS before and after your firewall can help validate that your firewall policy is correct and working. Granted, an IDS outside of the firewall will generate many more false positives and alerts than one inside.

2.2. Artificial Choke Points

Artificial Choke Points exist due to the logical topology of the network. For instance, a single switch or router that separates a server core from faculty machines or from student lab machines can be an excellent place for an IDS. Such an IDS will generate more false positives and alerts than one inside your firewall but is helpful in determining network anomalies and compromised machines inside the network; an example of snort's usefulness in this situation is show in Section 5. Artificial Choke Points can also be created using network hubs and/or span ports.

2.3. Intranet Trust/Untrust zone boundaries

Intranet Trust/Untrust zone boundaries are similar to Natural Choke Points but are intra-network based within the network. Routes from labs to faculty, faculty to server core, and labs to server core are all examples of this type. Whether or not firewalls are using internally to dictate traffic policy, an IDS at such a point can help pinpoint machines inside that are compromised but not attempting to propagate out of the network.

3. Snort Rules

Snort is primarily a signature based detection engine. That is, it looks for signatures in data streams and packet headers that are known to indicate an attack, potential attack, or data leak. Some IDS products use training and anomaly detection instead which can be useful and make a good complement to snort as well. While snort is primarily signature based, shipping with over 4000 rules, it has a number of protocol pre-processors and portscan detectors that work before the rules. These pre-processors handle packet re-assembly, protocol validation, and portscan detection. Some, or all, can be turned off or tuned depending on the environment.

Snort's rules are built using a simple minimal description format. The following rule[1] triggers an alert when an external machine has acquired 'root' via telnet:

```
alert tcp $TELNET_SERVERS 23 -> $EXTERNAL_NET any (msg:"TELNET root login";
  flow:from_server,established; content:"login|3A| root";
  classtype:suspicious-login; sid:719; rev:7;)
```

In the rule, `$TELNET_SERVERS` and `$EXTERNAL_NET` are variables that are defined in the main snort configuration file. Depending on tap placement, `$EXTERNAL_NET` could be anything not on your network, anything outside your server core, or any list of machines not allowed to telnet to the machine or machines defined in `$TELNET_SERVERS`. The content being scanned for in this example is `login|3A| root` which is ASCII text and the hexadecimal definition for a colon (`|3A|`). Rule payload matching can use any combination of ASCII or hexadecimal notation (for binary data). Note also that the `flow:from_server,established;` configuration item indicates that the TCP handshake has already taken place. Some rules are only configured depending on certain TCP flags and headers.

This rule format is easy to learn making custom rules trivial to implement for any environment. Furthermore, as new exploits come out, security websites, such as SANS[3], release snort rules along with their alerts. Also, the ruleset is constantly being optimized and automated ways to keep your rules database up to date are available. For those who like to be less conservative, a cutting edge community developed ruleset is available[6] which can be used along with the standard rules.

[1] The rules are defined on a single line that is wrapped here for printability reasons.

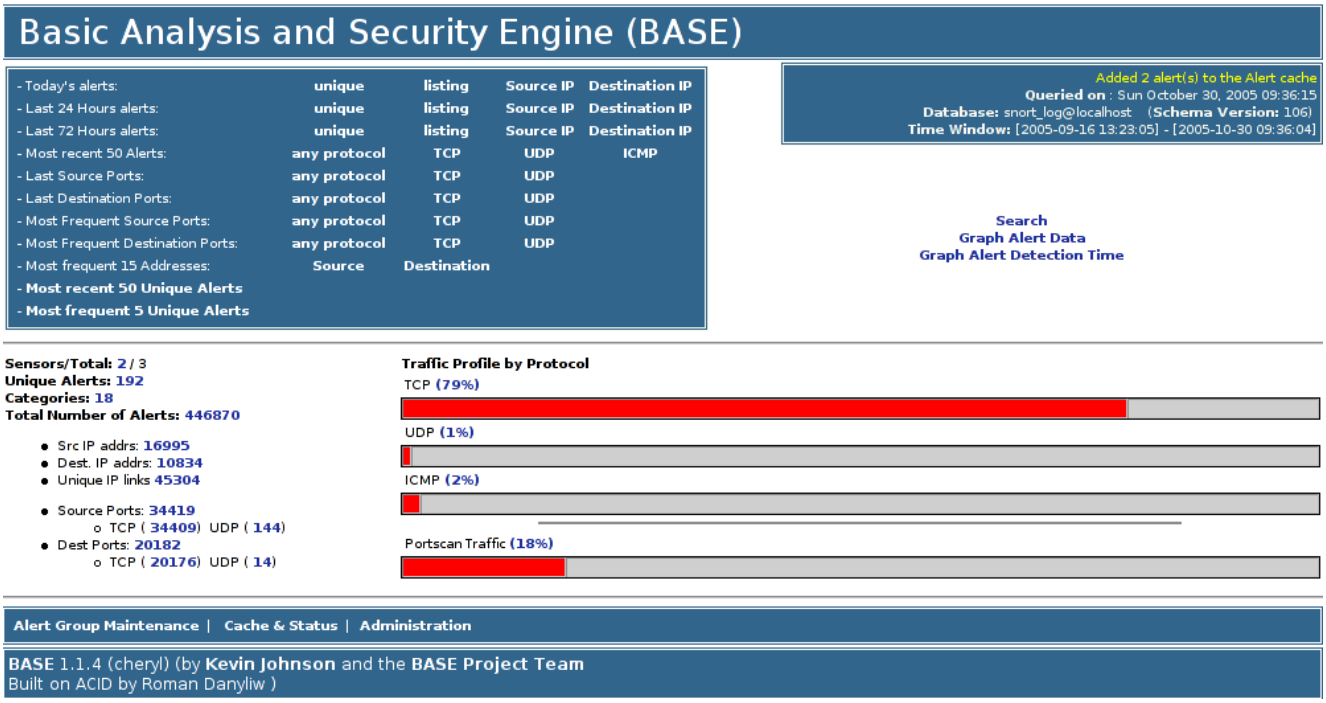

Figure 1: BASE console (Main Screen)

Rules exist for many different categories and can be disabled if the rules do not refer to your environment. Finally, Sourcefire[4] provides a more up to date ruleset for paying subscribers. These are available as updates are made to subscribers but make their way to registered users 5 days after release and are bundled into the snort codebase at every release.

As mentioned before, security is a process and the process of deploying and maintaining an IDS can take time. Many rules, while indicative of attacks, leaks, and protocol violations do generate false positives. It is critical that the rules be tuned to your environment to minimize the number of alerts so you are not deluged with too much information and can not respond effectively. For example, an untuned snort rulebase on a network of mostly Windows based machines can easily generate 100k alerts per day. Most of the false positives are just Windows machines being chatty and, once verified to be legitimate traffic, can be tuned out by narrowing the scope of the rules. Furthermore, the addition of new versions and applications can create new alert storms that have to be dealt with. Don't be discouraged, however, it isn't a hard process, just one that requires vigilance and a little time.

4. BASE (Basic Analysis and Security Engine)

BASE[2] is a PHP/MySQL web application that provides a central monitoring and analysis console for one or more snort taps. Each snort tap is configured to log to a MySQL database and that is used by BASE. The main BASE screen is shown in Figure 1. It provides a simple, easy to read overview of the number of alerts generated, unique inbound/outbound IP addresses seen, and a breakdown of alert percentages by protocol. Clicking on the links drills down into the stored alert data to provide more information. Some useful items include:

- Number of unique alerts
- Alerts ordered by category
- Today's alerts
- Most frequent src/dest ports

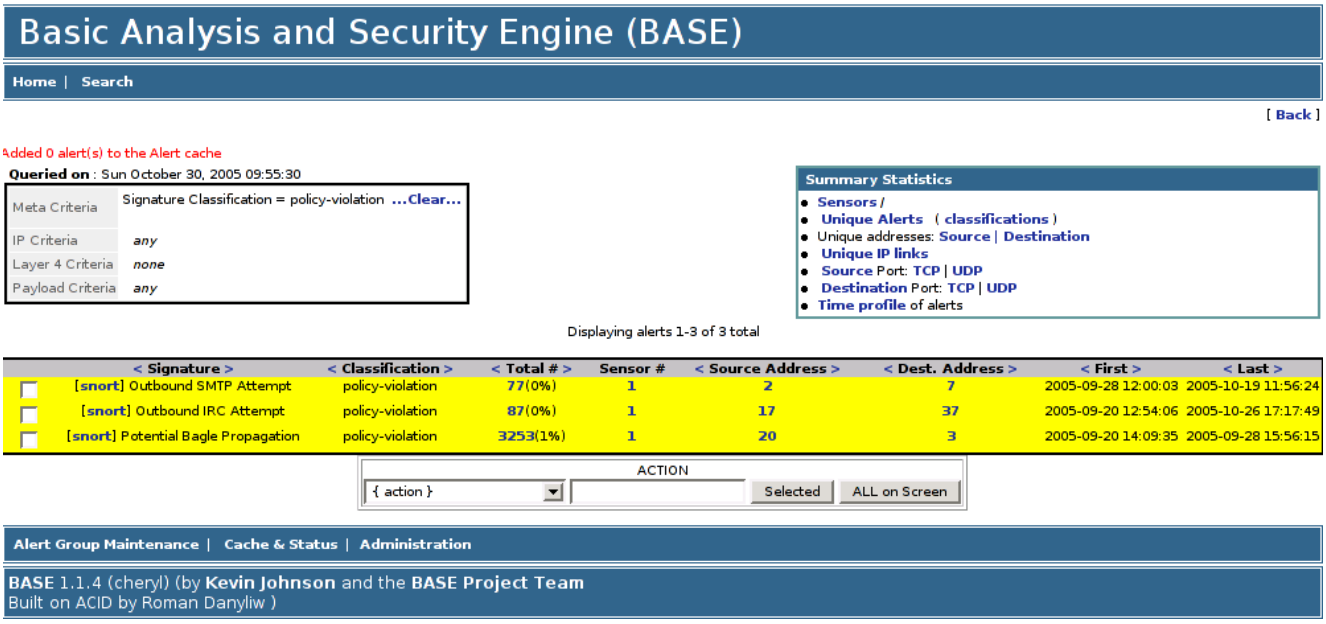

Figure 2: BASE console (Policy Violations)

Drilling down into the category for 'policy volation', you can see alerts specific to that category, including a rundown of the number of unique alerts in that category (See Figure 2). The 'policy volation' category allows for custom rules to alert based on local policy. For instance, Figure 2 shows three custom rules that generate alerts for outbound SMTP, IRC, and Attempted Bagel Worm Propagation. The outbound SMTP rule is configured to ignore authorized SMTP relays and smarthosts, only machines that shouldn't attempt SMTP connections are logged since that is often indicative of an infected machine or illegally installed software. While a good firewall policy should block the connections anyway, knowning this is helpful. For instance, a department recently installed a Xerox printer. Certain printers can email Xerox and order supplies. Snort intercepted the attempts by the printer to order its supplies and further investigation indicated a misconfiguration. Had this not been seen and the configuration updated on the printer, it would have run out of ink causing a loss in productivity. Was this a security problem. Not really but snort helped highlight a misconfiguration by seeing traffic that should not occur.

BASE can generate reports, archive alerts, email alerts to a ticketing system for followup, etc., and makes a good single view console for multiple snort taps.

5. A Worm propagation analysis example

Snort really shines in situations that are fluid. Many networks have multiple layers of Anti-Virus checkers in place: on workstations, on servers, on mailstores, and on email gateways. The problem with such checkers is that they update periodically. Our most active updating checker gets new signatures every 15 minutes which, in most cases, is sufficient. However, what if new worms were being released within that 15 minute timeframe? This happened near the end of September, 2005, when 3 bagle variants were released into the wild very quickly and some infected emails were able to traverse our gauntlet of AV scanners. We were alerted by the AV companies but knew it was too late. We knew that a number of infected messages had been received and, knowing how users are apt to click on anything, had probably infected a number of workstations. The question we asked ourselves was which ones?

Some worms when they infect a workstation make enough noise that it is simple to scan outbound firewall logs for odd connections or policy violations. These variants, however, used a two stage infection mechanism. The first stage was the email payload, the second included downloading a more robust malware system from a website. Quick analysis by security teams on the Internet indicated that the malware was being downloaded as a single .gif image with a known name from multiple URLs. It was simple, given this information, to write a quick snort rule to check for this new policy violation. The rule,

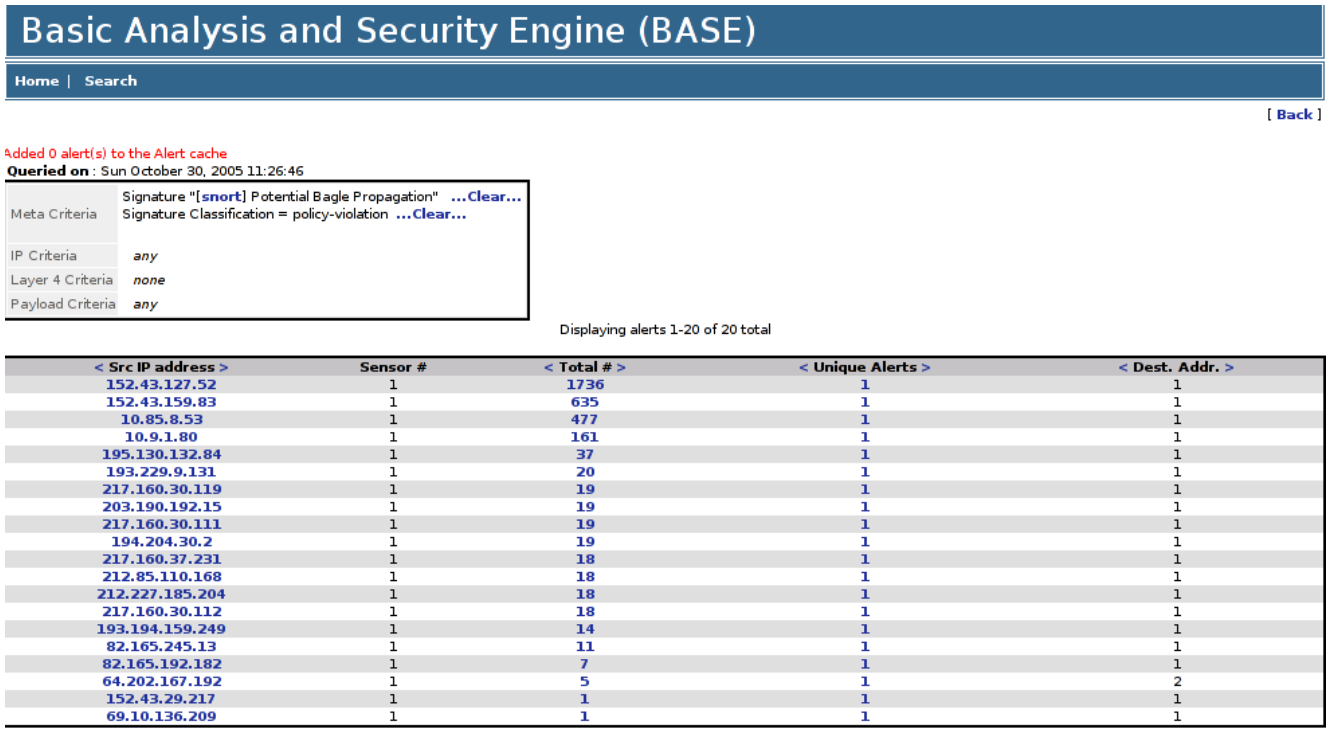

Figure 3: BASE console (Showing Internal Bagle Infections)

```
alert tcp any any -> any any (msg:"Potential Bagle Propagation";
  content:"osa6.gif"; classtype:policy-violation; sid:1000003; rev:3;)
```

was quickly put in place and snort started generating alerts (See Figure 3). The top 4 IP addresses are internal, the rest are external. While the rule could have been written to only capture requests going outbound, it was left open so that not only internal machines but the external payload delivery servers were also captured in case further analysis was needed. In this case, we identified 4 infected machines out of 5000 workstations and quickly deployed technicians to sanitize them.

This example is one of many where snort has been integral in network analysis. In this case, an IPS would not have been able to help given the fluid nature of the intrusion.

6. Conclusion

Snort provides another tool in the toolkit and can help provide information about exactly who's talking to who on the network. Its open nature provides a great platform for customization which is a key factor since no network is the same. While it takes work to deploy and maintain an IDS, the results are invaluable.

References

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

[2] *http://secureideas.sourceforge.net/*

[3] *http://www.sans.org/*

[4] *http://www.sourcefire.com/*

[5] *http://www.nitrosecurity.com/*

[6] *http://http://www.bleedingsnort.com/*