Snort – Lightweight Intrusion Detection for Networks

Martin Roesch – Stanford Telecommunications, Inc.

ABSTRACT

Network intrusion detection systems (NIDS) are an important part of any network security architecture. They provide a layer of defense which monitors network traffic for predefined suspicious activity or patterns, and alert system administrators when potential hostile traffic is detected. Commercial NIDS have many differences, but Information Systems departments must face the commonalities that they share such as significant system footprint, complex deployment and high monetary cost. Snort was designed to address these issues.

Introduction

Snort fills an important "ecological niche" in the the realm of network security: a cross-platform, lightweight network intrusion detection tool that can be deployed to monitor small TCP/IP networks and detect a wide variety of suspicious network traffic as well as outright attacks. It can provide administrators with enough data to make informed decisions on the proper course of action in the face of suspicious activity. Snort can also be deployed rapidly to fill potential holes in a network's security coverage, such as when a new attack emerges and commercial security vendors are slow to release new attack recognition signatures. This paper discusses the background of Snort and its rules-based traffic collection engine, as well as new and different applications where it can be very useful as a part of an integrated network security infrastructure.

Snort is a tool for small, lightly utilized networks. Snort is useful when it is not cost efficient to deploy commercial NIDS sensors. Modern commercial intrusion detection systems cost thousands of dollars at minimum, tens or even hundreds of thousands in extreme cases. Snort is available under the GNU General Public License [GNU89], and is free for use in any environment, making the employment of Snort as a network security system more of a network management and coordination issue than one of affordability.

What is "lightweight" intrusion detection?

A lightweight intrusion detection system can easily be deployed on most any node of a network, with minimal disruption to operations. Lightweight IDS' should be cross-platform, have a small system footprint, and be easily configured by system administrators who need to implement a specific security solution in a short amount of time. They can be any set of software tools which can be assembled and put into action in response to evolving security situations. Lightweight IDS' are small, powerful, and flexible enough to be used as permanent elements of the network security infrastructure.

Snort is well suited to fill these roles, weighing in at roughly 100 kilobytes in its compressed source distribution. On most modern architectures Snort takes only a few minutes to compile and put into place, and perhaps another ten minutes to configure and activate. Compare this with many commercial NIDS, which require dedicated platforms and user training to deploy in a meaningful way. Snort can be configured and left running for long periods of time without requiring monitoring or administrative maintenance, and can therefore also be utilized as an integral part of most network security infrastructures.

What is Snort?

Snort is a libpcap-based [PCAP94] packet sniffer and logger that can be used as a lightweight network intrusion detection system (NIDS). It features rules based logging to perform content pattern matching and detect a variety of attacks and probes, such as buffer overflows [ALE96], stealth port scans, CGI attacks, SMB probes, and much more. Snort has real-time alerting capability, with alerts being sent to syslog, Server Message Block (SMB) "WinPopup" messages, or a separate "alert" file. Snort is configured using command line switches and optional Berkeley Packet Filter [BPF93] commands. The detection engine is programmed using a simple language that describes per packet tests and actions. Ease of use simplifies and expedites the development of new exploit detection rules. For example, when the IIS Showcode [IISBT99] web exploits were revealed on the Bugtraq mailing list [BTQ99], Snort rules to detect the probes were available within a few hours.

Snort vs. The World!

Snort shares commonalities with both sniffers and NIDS. Two programs that lend themselves to direct comparison with Snort, tcpdump and Network Flight Recorder [NFR97], will be examined and contrasted in this section. In many cases, Snort is financially, technically, and/or administratively easier to implement than other Open Source [OSS98] or commercially available tools.

How Is Snort Different From tcpdump?

Snort is cosmetically similar to tcpdump [TCPD91] but is more focused on the security applications of packet sniffing. The major feature that Snort has which tcpdump does not is packet payload inspection. Snort decodes the application layer of a packet and can be given rules to collect traffic that has specific data contained within its application layer. This allows Snort to detect many types of hostile activity, including buffer overflows, CGI scans, or any other data in the packet payload that can be characterized in a unique detection fingerprint.

Another Snort advantage is that its decoded output display is somewhat more user friendly than tcpdump's output. Snort does not currently lookup host names or port names while running, which is a function that tcpdump can perform. Snort is focused on collecting packets as quickly as possible and processing them in the Snort detection engine. Performing run-time host name lookup is not conducive to high performance packet analysis. Figure 1 shows typical Snort output for a telnet banner display, and Figure 2 shows the same packet as displayed by tcpdump.

One powerful feature that Snort and tcpdump share, is the capability to filter traffic with Berkeley Packet Filter (BPF) commands. This allows traffic to be collected based upon a variety of specific packet fields. For example, both tools may be instructed via BPF commands to process TCP traffic only. While tcpdump would collect all TCP traffic, Snort can utilize its flexible rules set to perform additional functions, such as searching out and recording only those packets that have their TCP flags set a particular way or containing web requests that amount to CGI vulnerability probes. The SHADOW IDS [SHD98] from the Naval Surface Warfare Center is based on tcpdump and uses extensive BPF filtering. SHADOW is discussed in more detail near the end of this paper.

Snort and NFR

Perhaps the best comparison of Snort to NFR is the analogy of Snort as little brother to NFR's college-bound football hero. Snort shares some of the same concepts of functionality as NFR, but NFR is a more flexible and complete network analysis tool. That said, the little brother idea could be extended in that Snort tends to fit into small places and is somewhat more "nimble" than NFR. For example, NFR's packet filtering n-code language is a serious, full featured scripting language, while Snort's rules are more one dimensional. On the other hand, writing a Snort rule to detect a new attack takes only minutes once the attack signature has been determined. See Appendix A for an example of a simple web detection rule written in n-code and the analogous Snort rule.

NFR also has a more complete feature set than Snort, including IP fragmentation reassembly and TCP stream decoding. These features are essential in any commercial product that is meant to perform mission critical intrusion detection, and NFR was the first product which could defeat anti-NIDS attacks outlined by Ptacek and Newsham [PTA98]. Presently, Snort does not implement TCP stream reassembly, but future versions will implement this capability. Snort currently addresses IP fragmentation with a rule option that sets a minimum size threshold for fragmented packets. This rule option takes advantage of

```
20:59:49.153313 0:10:4B:D:A9:66 -> 0:60:97:7:C2:8E type:0x800 len:0x7D
192.168.1.3:23 -> 192.168.1.4:1031 TCP TTL:64 TOS:0x10 DF
***PA* Seq: 0xDF4A6536   Ack: 0xB3A6FD01   Win: 0x446A
FF FA 22 03 03 E2 03 04 82 0F 07 E2 1C 08 82 04  .."............
09 C2 1A 0A 82 7F 0B 82 15 0F 82 11 10 82 13 FF  ................
F0 0D 0A 46 72 65 65 42 53 44 20 28 65 6C 72 69  ...FreeBSD (elri
63 2E 68 6F 6D 65 2E 6E 65 74 29 20 28 74 74 79  c.home.net) (tty
70 30 29 0D 0A 0D 0A                             p0)....
```

Figure 1: Typical Snort telnet packet display.

```
20:59:49.153313 0:10:4b:d:a9:66 0:60:97:7:c2:8e 0800 125: 192.168.1.3.23 >
192.168.1.4.1031: P 76:147(71) ack 194 win 17514 (DF) [tos 0x10] (ttl 64,
id 660)
                         4510 006f 0294 4000 4006 b48d c0a8 0103
                         c0a8 0104 0017 0407 df4a 6536 b3a6 fd01
                         5018 446a d2ad 0000 fffa 2203 03e2 0304
                         820f 07e2 1c08 8204 09c2 1a0a 827f 0b82
                         150f 8211 1082 13ff f00d 0a46 7265 6542
                         5344 2028 656c 7269 632e 686f 6d65 2e6e
                         6574 2920 2874 7479 7030 290d 0a0d 0a
```

Figure 2: The same telnet packet as displayed by tcpdump.

the fact that there is virtually no commercial network equipment on the market that fragments packets smaller than 256-bytes. By setting this threshold value to some reasonable value, say 128-bytes, fragmented packet probes and attacks can be logged and alerts can be sent by Snort automatically. Full IP fragment and TCP stream reassembly and analysis will be addressed in later versions of Snort.

Under the Hood

Snort's architecture is focused on performance, simplicity, and flexibility. There are three primary subsystems that make up Snort: the packet decoder, the detection engine, and the logging and alerting subsystem. These subsystems ride on top of the libpcap promiscuous packet sniffing library, which provides a portable packet sniffing and filtering capability. Program configuration, rules parsing, and data structure generation takes place before the sniffer section is initialized, keeping the amount of per packet processing to the minimum required to achieve the base program functionality.

The Packet Decoder

The decode engine is organized around the layers of the protocol stack present in the supported data-link and TCP/IP protocol definitions. Each subroutine in the decoder imposes order on the packet data by overlaying data structures on the raw network traffic. These decoding routines are called in order through the protocol stack, from the data link layer up through the transport layer, finally ending at the application layer. Speed is emphasized in this section, and the majority of the functionality of the decoder consists of setting pointers into the packet data for later analysis by the detection engine. Snort provides decoding capabilities for Ethernet, SLIP, and raw (PPP) data-link protocols. ATM support is under development.

The Detection Engine

Snort maintains its detection rules in a two dimensional linked list of what are termed Chain Headers and Chain Options. These are lists of rules that have been condensed down to a list of common attributes in the Chain Headers, with the detection modifier options contained in the Chain Options. For example, if forty five CGI-BIN probe detection rules are specified in a given Snort detection library file, they generally all share common source and destination IP addresses and ports. To speed the detection processing, these commonalities are condensed into a single Chain Header and then individual detection signatures are kept in Chain Option structures.

These rule chains are searched recursively for each packet in both directions. The detection engine checks only those chain options which have been set by the rules parser at run-time. The first rule that matches a decoded packet in the detection engine triggers the action specified in the rule definition and returns.

A major overhaul of the detection engine is currently in the planning and development stage. The next version of the engine will include the capability for users to write and distribute plug-in modules and

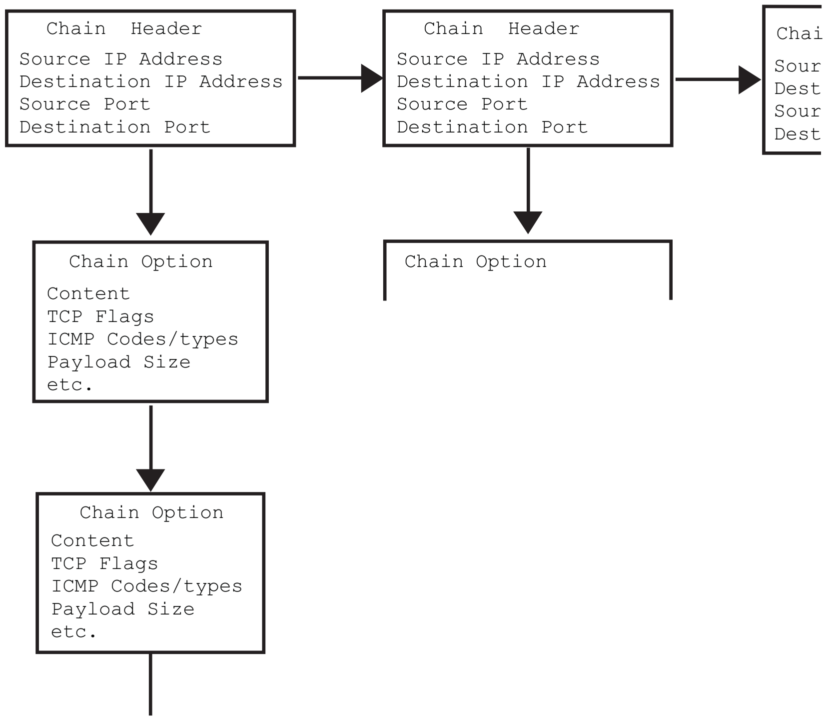

Figure 3: Rule Chain logical structure.

bind them to keywords for the detection engine rules language. This will allow anyone with an appropriate plug-in module to add significant detection functionality to Snort and customize the program for specific jobs.

The Logging/Alerting Subsystem

The alerting and logging subsystem is selected at run-time with command line switches. There are currently three logging and five alerting options. The logging options can be set to log packets in their decoded, human readable format to an IP-based directory structure, or in tcpdump binary format to a single log file. The decoded format logging allows fast analysis of data collected by the system. The tcpdump format is much faster to record to the disk and should be used in instances where high performance is required. Logging can also be turned off completely, leaving alerts enabled for even greater performance improvements.

Alerts may either be sent to syslog, logged to an alert text file in two different formats, or sent as WinPopup messages using the Samba smbclient program. The syslog alerts are sent as security/authorization messages that are easily monitored with tools such as swatch [SWT93]. WinPopup alerts allow event notifications to be sent to a user-specified list of Microsoft Windows consoles running the WinPopup software. There are two options for sending the alerts to a plain text file; full and fast alerting. Full alerting writes the alert message and the packet header information through the transport layer protocol. The fast alert option writes a condensed subset of the header information to the alert file, allowing greater performance under load than full mode. There is a fifth option to completely disable alerting, which is useful when alerting is unnecessary or inappropriate, such as when network penetrations tests are being performed.

Writing Snort Rules

Snort rules are simple to write, yet powerful enough to detect a wide variety of hostile or merely suspicious network traffic. There are three base action directives that Snort can use when a packet matches a specified rule pattern: pass, log, or alert. Pass rules simply drop the packet. Log rules write the full packet to the logging routine that was user selected at run-time. Alert rules generate an event notification using the method specified by the user at the command line, and then log the full packet using the selected logging mechanism to enable later analysis.

The most basic rules contain only protocol, direction, and the port of interest, such as in Figure 4.

```
log tcp any any -> 10.1.1.0/24 79
```

Figure 4: A simple Snort rule.

This rule would record all traffic inbound for port 79 (finger) going to the 10.1.1 class C network address space.

Snort interprets keywords enclosed in parentheses as ‘‘option fields’’. Option fields are available for all rule types and may be used to generate complex behaviors from the program, such as in Figure 5.

The rule in Figure 5 would detect attempts to access the PHF service on any of the local network’s web servers. If such a packet is detected on the network, an event notification alert is generated and then the entire packet is logged via the logging mechanism selected at run-time.

The rule IP address and port specifiers have several features available. The CIDR block netmask may be set to any value between one and thirty-two. Port ranges can be specified using the colon ‘‘:’’ modifier. For example, to monitor all ports upon which the X Windows service may run (generally 6000 through 6010), the port range could be specified with the colon modifier as shown in Figure 6.

Both ports and IP addresses can be modified to match by exception with the bang ‘‘!’’ operator, which would be useful in the rule described in Figure 7 to detect X Windows traffic from sources outside of the network.

This rule would generate an alert for all traffic originating outside of the host network that was bound for internal X Windows service ports.

Snort version 1.2.1 has fourteen option fields available:

1. content: Search the packet payload for the a specified pattern.
2. flags: Test the TCP flags for specified settings.
3. ttl: Check the IP header’s time-to-live (TTL) field.

```
alert tcp any any -> 10.1.1.0/24 80 (content: "/cgi-bin/phf"; msg: "PHF
probe!";)
```

Figure 5: Options allow increased rule complexity.

```
alert tcp any any -> 10.1.1.0/24 6000:6010 (msg: "X traffic";)
```

Figure 6: An example of port ranges.

```
alert tcp !10.1.1.0/24 any -> 10.1.1.0/24 6000:6010 (msg: "X traffic";)
```

Figure 7: Matching by exception on the source IP address

4. itype: Match on the ICMP type field.
5. icode: Match on the ICMP code field.
6. minfrag: Set the threshold value for IP fragment size.
7. id: Test the IP header for the specified value.
8. ack: Look for a specific TCP header acknowledgement number.
9. seq: Log for a specific TCP header sequence number.
10. logto: Log packets matching the rule to the specified filename.
11. dsize: Match on the size of the packet payload.
12. offset: Modifier for the content option, sets the offset into the packet payload to begin the content search.
13. depth: Modifier for the content option, sets the number of bytes from the start position to search through.
14. msg: Sets the message to be sent when a packet generates an event.

These options may be combined in any manner to detect and classify packets of interest. The rule options are processed using a logical AND between them; all of the testing options in a rule must be true in order for the rule to generate a "found" response and have the program perform the rule action.

Rule Development

Snort is extremely useful for rapidly developing new Snort rules. The clear and concise manner in which the data is displayed by the tool makes it perfect for writing new rules. The general method for development consists of getting the exploit of interest, such as a new buffer overflow, running the exploit on a test network with Snort recording all traffic between the target and attack hosts, and then analyzing the data for a unique signature and condensing that signature into a rule. Figure 8 shows Snort's view of a notional "IMAP buffer overflow" that has just come into widespread use by the "script kiddie" community.

The unique signature data in the application layer is the machine code just prior to the /bin/sh text string, as well as the string itself. Using this information, a new rule can be developed quickly, such as the one defined in Figure 9.

The content field of the rule contains mixed pain text and hex formatted bytecode, which is enclosed in pipes. At run-time, this data is converted into its binary representation, as displayed in the decoded packet dump in Figure 8, and then stored in an internal list of rules by Snort. Thus, the rule contained in Figure 9 will raise an alarm any time a packet containing the "fingerprint" of the new IMAP buffer overflow is detected.

Writing High Performance Pattern Matching Rules

The current rules system lends itself to high performance under most conditions, but there are some general concepts that can be applied when writing Snort rules to keep the processing speeds as high as possible. Computationally, the content matching option is the most expensive process that can be performed in the detection engine. Accordingly, it is performed after all other rule tests. This fact can be used to advantage by specifying other rule options in combination with the content option. For example, almost all requests to web servers have their TCP PUSH and ACK flags set. Using this knowledge, it is relatively easy to write a rule which will perform a simple TCP flag test before running the far more computationally intensive pattern match test.

Other options can be combined with the content rules to limit the amount of data that must be searched. The offset and depth keywords were made specifically to fulfill this function. Using these options, the area of the packet payload to search for an exploit pattern can be localized. Care should be taken to avoid limiting the search too severely. For example, many buffer overflows use variable offsets to tune the size and placement of the exploit machine code. A Snort rule that has been tuned too tightly to key on a specific area of a packet's payload may overlook the real exploit that has been shifted to a different area within

```
052499-22:27:58.403313 192.168.1.4:1034 -> 192.168.1.3:143
TCP TTL:64 TOS:0x0 DF
***PA* Seq: 0x5295B44E   Ack: 0x1B4F8970   Win: 0x7D78
90 90 90 90 90 90 90 90 90 90 90 90 90 90 EB 3B  ..............;
5E 89 76 08 31 ED 31 C9 31 C0 88 6E 07 89 6E 0C  ^.v.1.1.1..n..n.
B0 0B 89 F3 8D 6E 08 89 E9 8D 6E 0C 89 EA CD 80  .....n....n.....
31 DB 89 D8 40 CD 80 90 90 90 90 90 90 90 90 90  1...@...........
90 90 90 90 90 90 90 90 90 90 90 E8 C0 FF FF FF  ................
2F 62 69 6E 2F 73 68 90 90 90 90 90 90 90 90 90  /bin/sh.........
```

Figure 8: Notional

```
alert tcp any any -> 192.168.1.0/24 143 (content:"|E8C0 FFFF FF|/bin/sh";
msg:"New IMAP Buffer Overflow detected!";)
```

Figure 9: Alert rule for the new buffer overflow.

the packet. On the other hand, web CGI probes and attacks generally all take place at the beginning of the packet within the first thirty to fifty bytes. This can be a great place to optimize Snort content searching.

The actual search pattern used in the content rule is another area where performance tuning may take place. Snort uses a Boyer-Moore [SEDG97] algorithm to perform its pattern matching, which is one of the best algorithms available for that task. It achieves its greatest efficiency in cases where the pattern to match consists of non-repeating sets of unique bytes. For example, the Intel x86 architecture uses the hex value 0x90 to indicate a NOP in machine code. Buffer overflows generally use large regions of NOPs to pad the actual exploit code and make the return jump calculations easier for the exploit programmer. When specifying content match patterns, it is best to avoid including any NOPs in the match pattern, which will otherwise cause the Boyer-Moore routine to complete many partial matches before actually finding the correct match pattern.

Advanced Snorting

Snort is a flexible tool with a wide variety of uses. It is intended to be used in the most classic sense of a network intrusion detection system. It examines network traffic against a set of rules, and alerts administrators to suspicious network activity so that they may react appropriately. There are many other areas where Snort can be useful as well.

Shoring Up Commercial IDS's

Snort can be used to fill holes in commercial vendor's network-based intrusion detection tools, such as when a new attack makes its debut in the hacker/cracker community and signature updates are slow to come from the vendor. In this case, Snort may be used to characterize the new attack by running it locally on a test network and determining it's signature. Once the signature is written into a snort rule, the BPF command line filtering may be used to limit the traffic that Snort analyzes to the service or protocol of interest. Snort can be used as a very specialized detector for a single attack or family of attacks in this mode.

The recent IRDP denial of service attack [IRDP99] revealed by the L0pht provides a good example of this concept. The same day that the attack was announced, Snort rules were made available by the user community and these attacks were detectable.

Passive Traps

Another application to which Snort is very well suited is as a Honeypot monitor. Honeypots are programs or computers that are dedicated to the notion of deceiving hostile parties interested in a network. Most honeypot systems, for example Fred Cohen & Associates Deception Toolkit [DTK98], record their data at the server level, with a fake ''service'', such as an FTP server actually recording the data sent to it. The problem with that concept is that the services doing the recording have to be started before they will record anything. This means that events such as stealth port scans or binary data streams will be missed or garbled on honeypots that don't perform packet level monitoring. Another problem is that the data generated by such a system will tend to be complex by its nature.

The data coming out of a honeypot requires a skilled analyst to properly interpret the results. Snort can be a great help to the analyst/administrator with its packet classification and automatic alerting functionality. With these capabilities a honeypot can be erected as a stand alone intrusion detection mechanism. It requires no other monitoring or maintenance because Snort can be set to record and generate event notification on the first packet that arrives at the honeypot.

Snort can be used to implement another concept that is being advocated today; that of ''passive traps'' [MJR99]. A passive trap uses the ''home field advantage'' that network administrators enjoy when securing their networks. One aspect of this concept is that administrators know which services are <B>not</B> available on their networks. Snort rules can be written that watch for traffic headed for these non-existent services. Packets which are found to be using these ports may be an indication of port scanning, backdoors, or other hostile traffic. For example, a network that is not using TFTP can be configured with Snort alert rules for all packets headed to or from any node on the network bound for port 69. This can be a good method for detecting covert communications channels such as Loki or backdoors like Back Orifice. Another easy concept to implement to set up pass rules for all of the services known to be running on a network and log inbound connections to other ports or port ranges.

Shining Some Light on SHADOW

SHADOW is designed to be a cheap alternative to commercial NIDS. As an aside, SHADOW was probably the first true lightweight intrusion detection system. tcpdump is used as the sensor in these systems, which are configured using often extensive BPF commands. All traffic that is not filtered out with these BPF rules is collected into a single file that can become quite large over extended periods of time. Once the data is collected by the sensor, it is post-processed using a variety of external third party tools. There are some limitations to this system, including a complete lack of real-time alerts and a lack of good data classification tools to aid the analyst in identifying the data produced by the sensor.

Snort uses the same BPF filter language rules as tcpdump, and can be used as a complete replacement for tcpdump sensors in environments where SHADOW is the IDS of choice. The advantages of using Snort as a replacement sensor include real-time automatic traffic classification as it is collected and real-time alerting. This allows security events to be detected and acted upon by the administrative staff in a more timely manner and log file sizes to be reduced significantly. At the same time, Snort can record the data it collects to tcpdump formatted files so that the data generated by the system can be post-processed for in depth analysis with existing tools that analysts are comfortable using.

Focused Monitoring

"Focused monitoring" is the concept of watching a single critical node or service on a network for signs of hostile activity. For example, the Sendmail [ALMN99] SMTP server has an extensive and well known list of vulnerabilities and exploits. A single Snort sensor could be deployed with a rule set that covers all known Sendmail attacks and would provide highly focused monitoring of that specific traffic on the network. These rules could even be extended to provide a running narrative of all of the commands and responses into and out of SMTP servers on the defended network. This can make the network security analysts job somewhat easier by letting the collection engine (Snort) describe the normal flow of commands and responses as well as the attacks.

Focused monitoring can be especially useful in instances where existing NIDS provide inadequate coverage. For example, a set of rules that monitor SQL database queries to a web or database server could be developed. This would provide more complete coverage of CGI and ODBC SQL attacks and probes than any commercial NIDS on the market today. This concept can be extended to any network communications technology that is under represented by commercial NIDS.

Conclusions

Snort was designed to fulfill the requirements of a prototypical lightweight network intrusion detection system. It has become a small, flexible, and highly capable system that is in use around the world on both large and small networks. It has attained its initial design goals and is a fully capable alternative to commercial intrusion detection systems in places where it is cost inefficient to install full featured commercial systems.

Availability and Requirements

Snort will run on any platform where libpcap will run. The current version of Snort is 1.2.1, and libpcap is required to compile and run the software. Snort is known to run on RedHat Linux 5.1/5.2/6.0, Debian Linux, MkLinux, S/Linux, HP-UX, Solaris 2.5.1-2.7 (x86 and Sparc), x86 Free/Net/OpenBSD, M68k NetBSD, and MacOS X.

Information about snort may be acquired directly from the author's web site at http://www.clark.net/~roesch/security.html .

Snort may be downloaded from the author's web site at http://www.clark.net/~roesch/snort-1.2.1.tar.gz .

There is a slowly growing library of Snort rules available at http://www.clark.net/~roesch/snort-lib .

Acknowledgements

Snort originally used Mike Borella's ipgrab program as a development template and example for how to properly code libpcap programs and packet decoders. ipgrab can be found at http://www.borella.net. Mike's code is an excellent starting point for any libpcap-based project.

Ron Gula of Network Security Wizards http://www.securitywizards.com provided valuable advice on logging methodologies and some of the initial program logic, as well as contributing example rules to the system.

Ken Williams <jkw@frey.rapidnet.com> has been fantastically supportive throughout the development of Snort, providing encouragement and ideas for additional features as well as providing a friendly forum for the distribution of Snort.

The Snort user community has been especially enjoyable to work with, providing bug reports, ideas for new development directions, and new rules for the library since the program's initial release. Their support and enthusiasm has kept this a vital and growing collaborative project far past what I had imagined was possible!

References

[SHD98] *SHADOW*, Steven Northcutt et al., Naval Surface Warfare Center Dahlgren Laboratory, 1998, http://www.nswc.navy.mil/ISSEC/CID/ .

[TCPD91] *tcpdump*, Van Jacobson, Craig Leres and Steven McCanne, Lawrence Berkeley National Laboratory, 1991, http://www-nrg.ee.lbl.gov/ .

[PCAP94] *libpcap*, Van Jacobson, Craig Leres and Steven McCanne, Lawrence Berkeley National Laboratory, 1994, http://www-nrg.ee.lbl.gov/ .

[DTK98] *Deception Toolkit*, Fred Cohen & Associates, 1998, http://all.net/dtk/dtk.html .

[GNU89] *GNU General Public License*, Richard Stallman, 1989, http://www.gnu.org/copyleft/gpl.txt .

[BPF93] "The BSD Packet Filter: A New Architecture for User-level Packet Capture," Steven McCanne, Van Jacobson, *USENIX Technical Conference Proceedings*, 1993.

[ALE96] "Smashing the Stack for Fun and Profit," Aleph1, *Phrack #49*, 1996, http://www.phrack.com .

[BTQ99] Bugtraq Mailing List, archives and vulnerability data base are available at Security Focus, http://www.securityfocus.com .

[IISBT99] "NT IIS Showcode ASP Vulnerability," *Bugtraq ID #167*, Parcens/L0pht, May, 1999, http://www.securityfocus.com .

[OSS98] *The Cathedral and the Bazaar*, Eric S. Raymond, 1998, http://www.tuxedo.org/~esr/writings/cathedral-bazaar/ .

[FYD97] "The Art of Port Scanning," Fyodor, *Phrack #51*, 1997, http://www.insecure.org/nmap/p51-11.txt .

[SWT92] "Centralized System Monitoring With Swatch," Stephen E. Hansen and E. Todd Atkins, *USENIX Seventh Systems Administration Conference*, 1993, http://www.stanford.edu/˜atkins/swatch/lisa93.html .

[SEDG97] *Algorithms in C: Fundamentals, Data Structures, Sorting, Searching,* Robert Sedgewick, Addison-Wesely Publishing Company, 1997.

[IRDP99] *L0pht Security Advisory*, Silicosis and Mudge, August 1999, http://www.l0pht.com/advisories/rdp.txt .

[ALMN99] *Sendmail*, Eric Allman, 1999 http://www.sendmail.com .

[PTA98] *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*, Thomas Ptacek and Timothy Newsham, Secure Networks Inc, 1998, http://www.nai.com/services/support/whitepapers/security/IDSpaper.pdf .

[MJR99] *Burglar Alarms for Detecting Intrusions*, Marcus Ranum, NFR Inc, 1999, http://www.blackhat.com/html/bh-usa-99/bh3-speakers.html .

Author Information

Martin Roesch is a Network Security Engineer with Stanford Telecommunications Inc. He holds a B.S. in Computer Engineering from Clarkson University. He has extensive experience with intrusion detection systems and has developed several systems professionally. He was a primary software engineer during the development of GTE Internetworking's Global Network Infrastructure IDS, and designed and developed GTE's new commercial honeypot/deception system "Sentinel". He is also a member of the Trinux Linux Security Toolkit distribution development team. Snort is his first Open Source Software project, and has been an excellent learning experience for him. Contact him at <roesch@clark.net>.

Appendix A

Sample NFR rule to detect web server CGI probes (n-code sample excerpted from the L0pht's NFR IDS Modules web page at http://www.l0pht.com/NFR).

```
badweb_schema = library_schema:new( 1, ["time", "int",
                                        "ip", "ip", "str"], scope());

# list of web servers to watch. List IP address of servers or a netmask
# that matches all. use 0.0.0.0:0.0.0.0 to match any server

da_web_servers = [ 0.0.0.0:0.0.0.0 ] ;

query_list = [ "/cgi-bin/nph-test-cgi?",
               "/cgi-bin/test-cgi?",
               "/cgi-bin/perl.exe?",
               "/cgi-bin/phf?"
                ] ;

filter bweb tcp ( client, dport: 80 )
{
        if (! ( tcp.connDst inside da_web_servers) )
                return;
        declare $blob inside tcp.connSym;
        if ($blob == null)
                $blob = tcp.blob;
        else
                $blob = cat ( $blob, tcp.blob );
        while (1 == 1) {
          $x = index( $blob, "\n" );
          if ($x < 0)          # break loop if no complete line yet
            break;
          $t=substr($blob,$x-1,1);      # look for cr at end of line
          if ($t == '\r')
            $t=substr($blob,0,$x-1);    # tear off line
          else
            $t=substr($blob,0,$x);

          $counter=0;
          foreach $y inside (query_list) {
            $z = index( $blob, $y );
            if ( $z >= 0) {
              $counter=1;
              # save the time, the connection hash, the client,
              # the server, and the command to a histogram
              record system.time, tcp.connHash, tcp.connSrc, tcp.connDst,
                                                        $t to badweb_hist;

            }
          }
          if ($counter)
            break;
        }
        # keep us from getting flooded if there is no newline in the data
        if (strlen($blob) > 4096)
                $blob = "";

        # save the blob for next pass
        $blob = substr($blob, $x + 1);

}

badweb_hist = recorder ("bin/histogram packages/test/badweb.cfg",
        "badweb_schema" );
```

Appendix B: Snort rules to detect the same web CGI probes.

```
alert tcp any any -> any 80 (msg:"CGI-nph-tst-cgi";
                content:"cgi-bin/nph-test-cgi?"; flags: PA;)
alert tcp any any -> any 80 (msg:"CGI-test-cgi";
                content:"cgi-bin/test-cgi?"; flags: PA;)
alert tcp any any -> any 80 (msg:"CGI-perl.exe";
                content:"cgi-bin/perl.exe?"; flags: PA;)
alert tcp any any -> any 80 (msg:"CGI-phf";
                content:"cgi-bin/phf?"; flags: PA;)
```