Chapter 5

Using Snort for Network-Based Forensics

INFORMATION IN THIS CHAPTER

- IDS Overview
- Snort Architecture
- Snort Preprocessor Component
- Snort Detection Engine Component
- Network Forensics Evidence Generated with Snort

This chapter, which comprises five sections, discusses the use of Snort as a network-based intrusion detection system (NIDS) during a network forensics investigation. It is a detective-technical security control, used by organizations' security teams and network forensics examiners to monitor network and/or system activities for malicious activities or security policy violations. The first section, "IDS Overview," provides an overview of intrusion detection systems (IDSes), types of IDSes, and IDS Matrix. The second section, "Snort Architecture," provides an overview of the four phases of the Snort Architecture. The four phases are the Sniffer Component, Preprocessor Component, Detection Engine Component, and the Alert/Logging Component. In addition, in this section, Snort execution procedures are presented real time or playback analysis. The third section, "Snort Preprocessor Component," provides a description of the six categories used to group the 14 different Snort Preprocessor plug-ins and how to use them. The fourth section, "Snort Detection Engine Component," discusses the Snort rule language, the various detection engine algorithms for performance tuning the system, and how to use the Snort rules. The final section, "Network Forensics Evidence Generated with Snort," entails ensuring the three forms of Snort evidence is admissible as evidence and not classified as hearsay evidence.

IDS OVERVIEW

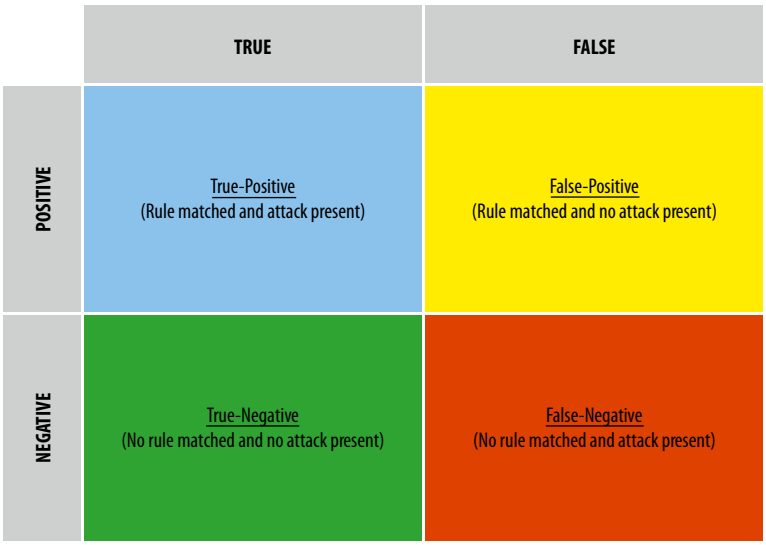
An IDS is a solution implemented by organizations to monitor networks and/or systems for malicious activities or security policy violations. Host-based and network-based IDS solutions are the most common form implemented. In a host-based intrusion detection system (HIDS), software agents monitor predefined local, remote, and network activities (files, logs, passwords) within a host for intrusions. In a NIDS, the sensors are placed at critical network locations throughout the enterprise, often in the demilitarized zone (DMZ) or at the network perimeters. The sensor captures all network traffic and analyzes the content of individual packets for malicious traffic. NIDS typically access network traffic by connecting to a hub, tapping into a network cable, or mirroring network traffic to a switched port analyzer (SPAN) port. The SPAN feature, sometimes called *port mirroring* or *port monitoring*, copies all traffic from the port or ports that it is monitoring to another port where it can be used by a network analyzer or IDS for analysis.

For the detection of network and/or system security policy violations, most IDSes use one of two detection techniques: statistical anomaly based and/or signature based. The statistical anomaly based IDS establishes a normal network traffic baseline and compares network traffic activity to the baseline in order to detect whether or not it is within the baseline parameters. If the sampled traffic is outside the baseline parameters, an alert is generated. The signature-based IDS uses preconfigured and predetermined attack patterns known as *signatures* and compares network traffic against those signatures. If the sample traffic matches a pattern, an alert is generated.

Independent of the type of IDS implemented, all IDS solutions produce one of four different responses based upon the received alert. The IDS responses are presented in the IDS Matrix diagram in Figure 5.1.

The IDS Matrix diagram is a two-by-two matrix designed to present the set of conditions that, when examined, indicate some type of intrusion event has occurred. The following is the description of the four conditions:

- True-Positive – This condition indicates that a signature was matched or an anomaly was identified, an attack actually occurred and an alert was generated. If this condition is triggered, it should result in appropriate action taken by the incident response team.
- False-Positive – This condition indicates that a signature was matched or an anomaly was identified, an alert was generated but there was no



■ **FIGURE 5.1** IDS Matrix diagram

attack present. If this condition exists, the IDS needs to be tuned to reduce this type of false indicator.

- True-Negative – This condition indicates that no attack has occurred, so no signature was matched or no anomaly was detected, and therefore, no alert was generated.
- False-Negative – This condition indicates that an attack has occurred but no signature was matched or no anomaly was detected and no alert was generated. This is the worst of the four conditions. This is typically indicative of zero-day/out-in-the-wild outbreaks.

NOTE
In today's environments where a large percentage of attacks occur over the network, having a NIDS is more of a requirement than an option. As a result, the network forensics examiner needs to include a NIDS tool in their toolkit.

The chapter discusses Snort, a NIDS designed to capture live network traffic or playback precaptured network traffic for advance intrusion analysis. The precaptured network traffic should be saved as a "de facto" standard. The "de facto" standard for network data is the libpcap library format known as *pcap* (for UNIX/Linux-based operating systems [OSes]). For Microsoft Windows-based OSes, the library format is known as *WinPcap,* but it is the same format as the UNIX/Linux-based pcap.

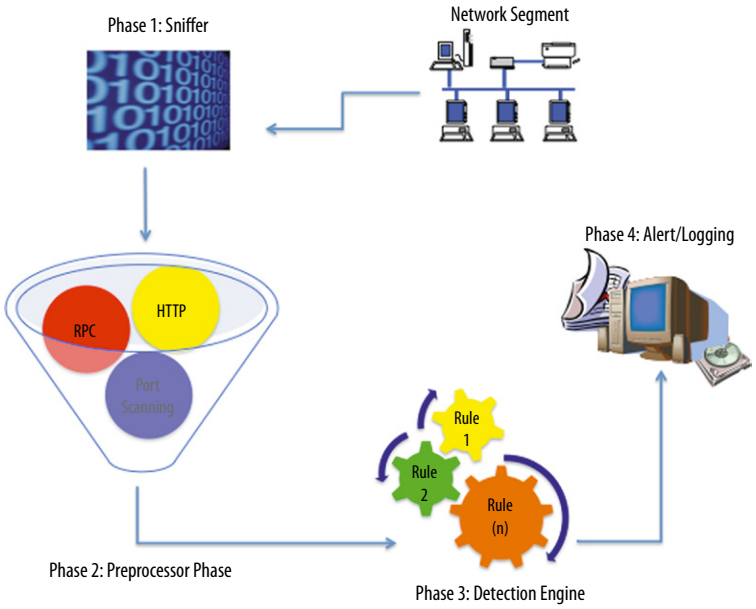
SNORT ARCHITECTURE

Snort, a free open-source multiplatform product, can be configured to run in four modes. The first mode, Sniffer, functions as a packer sniffer that reads the packets off the network. During this mode, the captured packets can be displayed in a continuous stream on a monitor. The second mode, Packet Logger, can be configured to log the packets to disk. The third mode, NIDS, allows Snort to analyze decoded network traffic against predefined preprocessors and rules and performs several different actions if a match is found. The fourth mode, Inline, allows Snort to obtain packets from iptables and drop or pass those packets based on Snort Inline-specific rule types.

The network forensics examiner should implement the Snort NIDS mode of functionality for conducting an investigation. This mode is preferred because of its noninvasive architecture. In this mode, either the network forensics examiner can attach a NIDS device to the organization's targeted subnet via a SPAN port or the network hub containing the target host(s) or obtain precaptured network binary files if a network sniffer is already deployed. While Snort can function in the other three modes very successfully, the network forensics examiner's goal should be to minimize the impact or modification to an environment or evidence.

NOTE
While the idea of a large organization having network sniffers predeployed is feasible, due to storage restrictions, most organizations will not be able to collect the vast amount of traffic or at least store the network traffic for long periods. The author has been in environments where once a security incident has been detected, preinstalled network sniffers have been activated to commence the collection of binary network traffic. In addition, the author has been in environments where organizations have implemented a distributed IDS infrastructure were multiple deployed IDS sensors transmit IDS alerts to a central IDS management console. As a network forensics investigator, if you are not sure which environment you are working with the rule of thumb is always ask!

To successfully use Snort, the network forensics examiner needs a fundamental understanding of its architecture. The Snort Architecture, presented in Figure 5.2, consists of four phases. The first phase, Sniffer Component, captures network traffic from designated network segments and decodes the protocols. The second phase, Preprocessor Component, receives the decoded protocol traffic and analyzes the traffic for a particular type of behavior using enabled plug-ins (for example, remote procedure call (RPC), Hypertext Transfer Protocol (HTTP), Port Scanning). The third phase, Detection Engine Component, receives the



■ **FIGURE 5.2** Snort Architecture

Table 5.1 Command-Line Switches for Enabling Snort to Perform Real-Time Network Analysis

Command Line Switch	Description
`-v`	Verbose.
`-l`	This option sends the Snort output to a log file.
`-d`	Dump the application layer data.
`-e`	This option puts Snort in packet sniffing mode and includes the data link layer headers.
`-c`	Instructs Snort to read the configuration file (for example, snort.conf). It can be a different file.

Preprocessor Component traffic and compares it against rules. If a rule matches the data sent via the Preprocessor Component, an alert is triggered. The final phase, Alert/Logging Component, receives the trigger alert from the Detection Engine Component and uses plug-ins to transfer the alerts to databases, log files, Syslog servers, SNMP traps, and WinPopup Messages.

To enable the NIDS mode, the network forensics examiner will execute the Snort command either to analyze real-time network traffic in a noninvasive matter or to playback binary network traffic (pcap format) previously captured. The following is the syntax for real-time analysis. The command-line switches in this syntax are described in Table 5.1.

Real-Time Network Traffic Capturing

```
snort -vde -c snort.conf
```

Playback Binary Network Traffic (pcap Format)

```
snort -c snort.conf --pcap-file=<file>
```

Snort.conf is the name of the Snort configuration file. In this file, the following settings can be customized:

- Set the variables for your network
- Configure dynamic-loaded libraries
- Configure preprocessors
- Configure output plug-ins
- Add any runtime config directives
- Customize your rule set

A default snort.conf file is located in the /doc subfolder of the Snort application folder. The default snort.conf file can be modified to include and/or exclude any of the above configurations. In addition, during the execution of the Snort command, Berkeley Packet Filters (BPFs) can be used to allow packets to be filtered. Using filters, optimizes the performance by only passing Snort packets associated with the traffic that we are interested in analyzing. The following is the syntax to use Snort with a BPF and an example:

```
snort -c snort.conf --pcap-file=<file> <bpf>
snort -c snort.conf --pcap-file=<file> host 192.168.1.10
```

There are three different kinds of BPF qualifiers. Table 5.2 lists the BPF options.

For additional information (including default settings) about BPFs, a complete list is available via tcpdump filters manual page.

While each of the four components is of interest, this book focuses on the Preprocessor Component and the Detect Engine Component. In order for the network forensics examiner to monitor network and/or system activities for malicious activities or security policy violations, the Preprocessor Component and the Detection Engine Component are essential modules of the Snort Architecture.

SNORT PREPROCESSOR COMPONENT

The Snort Preprocessor Component extends the functionality of Snort by enabling the creation of modular plug-ins. The Preprocessor plug-ins are small modular software applications that provide specific protocol analysis. After the Snort Preprocessor Component receives the decoded protocol

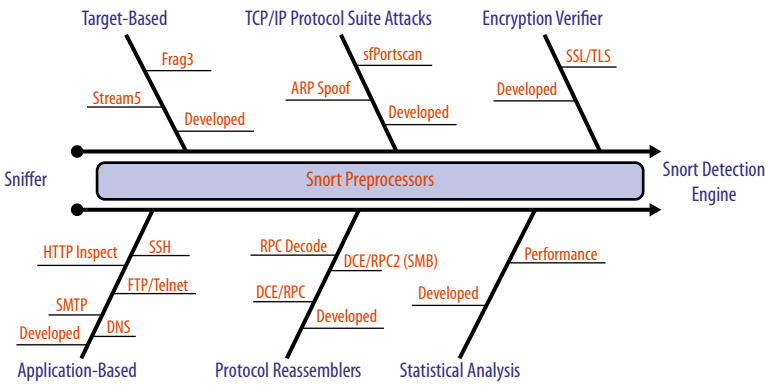
Table 5.2 BPF Qualifiers

Qualifier	Description

Type – Possible types are `host`, `net`, and `port`	This qualifier indicates what kind of entity is referenced. Examples are as follows:
	`host sundown`
	`net 128.3`
	`port 443`

Directional – Possible directions are `src`, `dst`, `src`, or `dst` and `src` and `dst`	This qualifier indicates a particular transfer direction to and/or from entity. Examples are as follows:
	`src sundown`
	`dst net 128.3`
	`src or dst port ftp-data`
	If there is no directional qualifier, `src` or `dst` is assumed.

Proto – Possible protos are: `ether`, `fddi`, `tr`, `ip`, `ip6`, `arp`, `rarp`, `decnet`, `tcp`, and `udp`.	This qualifier restricts the match to a particular protocol. Examples are as follows:
	`ether src sundown`
	`arp net 128.3`
	`tcp port 21`



■ **FIGURE 5.3** Snort Preprocessor plug-in categories

traffic from the Snort sniffer component, enabled Preprocessor plug-ins (for example, RPC, HTTP, Port Scanning) analyze the decoded traffic for a particular type of behavior.

Snort version 2.8.5.1 (the version this book is based on) has 14 predefined plug-ins. In addition to using existing Preprocessor plug-ins, the Snort Architecture provides a framework for the development of new Preprocessor plug-ins. The 14 predefined plug-ins can be grouped into six categories (see Figure 5.3).

The first preprocessor category, Target-Based Plug-ins (also known as *Target-Based IDS*), analyzes Ptacek and Newsham style attacks targeting specific OS platforms (for example, Linux, Windows, SunOS, CISCO). In the past, Ptacek and Newsham style attacks could evade the generically configured IDS. This evasiveness was because the implementation of IP stacks and Transmission Control Protocol/User Datagram Protocol (TCP/UDP) handling of overlapping data varied amongst different vendor OSes. The preprocessors described in Table 5.3 reside within this category.

The second preprocessor category, Transmission Control Protocol/Internet Protocol (TCP/IP) Suite Attack Plug-ins, detects various different types of TCP/IP port scanning techniques. This category also includes plug-ins to decode Address Resolution Protocol (ARP) and detect specific ARP attacks. The preprocessors described in Table 5.4 reside within this category.

The third category, Encryption Verifier Plug-ins, decodes Secure Sockets Layer (SSL) and Transport Layer Security (TLS) traffic and determines whether Snort should stop the inspection of encrypted traffic. The Preprocessor plug-in commences the addressing of encrypted traffic. This is an area mostly ignored by IDSes due to false-positives and decrypting performance reasons. The SSL/TLS preprocessor resides within this category (see Table 5.5).

The fourth category, Application-Based Plug-ins, decodes application-specific protocols. This includes commands, client requests and server responses, and application-unique exploits. The preprocessors described in Table 5.6 reside within this category.

Table 5.3 Target-Based Plug-in Preprocessors

Preprocessor	Description
Frag3	A target-based IP defragmentation module.
Stream5	A target-based TCP reassembly module (tracks both TCP and UDP traffics).

Table 5.4 TCP/IP Protocol Suite Attack Plug-in Preprocessors

Preprocessor	Description
sfPortscan	A reconnaissance phase module alerts on Network Mapper (NMAP) scans, decoy portscans, distributed portscans, portsweeps, and filtered portscans and portsweeps.
ARP Spoof	A module used to decode ARP packets, detect ARP attacks, unicast ARP requests, and other inconsistent Ethernet to IP mapping.

Table 5.5 The SSL/TLS Preprocessor

Preprocessor	Description
SSL/TLS	A module used to analyze encrypted SSL/TLS traffic or to inspect initial SSL handshake.

Table 5.6 Application-Based Plug-in Preprocessors

Preprocessor	Description
HTTP Inspect	A module designed to decode generic HTTP-based client requests and server responses. This includes IIS and Apache server-side responses.
SMTP	A module designed to decode SMTP-based client requests and server responses.
FTP/Telnet	A module designed to decode FTP/Telnet client requests and server responses.
SSH	A module designed to detect specific SSH exploits: Challenge-Response, CRC-32, Secure CRT, and Protocol Mismatch.
DNS	A module designed to decode DNS responses and detect specific DNS exploits: DNS Client RData Overflow, Obsolete Record Types, and Experimental Record Types.

Table 5.7 Protocol Reassembler Plug-in Preprocessors

Preprocessor	Description
RPC Decode	A module designed to decode and combine RPC packets.
DCE/RPC	A module used to detect and decode SMB and DCE/RPC traffics. The SMB packets are decoded to access the DCE/RPC traffic. In addition, focuses on SMB desegmentation and DCE/RPC defragmentation.
DCE/RPC 2	A module used to perform SMB desegmentation and DCE/RPC defragmentation to technique rule evasion.

The fifth category, Protocol Reassembler Plug-ins, detects, decodes, and analyzes fragmented Distributed Computing Environment/Remote Procedure Call (DCE/RPC) traffic. These plug-ins analyze segmented server message block (SMB) traffic to access the DCE/RPC traffic. The purpose of these is to circumvent techniques used to evade IDS detection. The preprocessors described in Table 5.7 reside within this category.

The final category, Statistical Analysis Plug-ins, provides performance metrics for the IDS and the network traffic statistics. The preprocessors described in Table 5.8 reside within this category.

Table 5.8 Statistical Analysis Plug-in Preprocessors

Preprocessor	Description
Performance Monitor	A module used to measure Snort's performance. This includes, but is not limited to, the following: Alerts/sec, Time Stamp, Mbits/sec, CPU usage, Syns/sec, SynAcks/sec, Frag-Completes/sec, Closed TCP Sessions/sec, TCP Sessions initializing, TCP Sessions Established, and TCP Sessions Closing.

The Snort Preprocessor plug-ins can provide the network forensics examiner three different types of evidence. The first type of evidence, pregenerated IDS Alerts, would allow the examiner to analyze the existing IDS log files to determine if Preprocessor plug-ins were used and generated relevant evidence. The second type of evidence, binary packet capture (pcap format) files, can be imported into Snort and analyzed using the above-mentioned Preprocessor plug-ins or the examiner can develop a new Preprocessor plug-in. The final type of evidence, live packet captures, can determine if the attack is still ongoing. This form of analysis, similar to Live Digital Forensics analysis, requires the careful implementation of a Snort IDS into the environment using sound forensics procedures.

Remember that Preprocessor plug-ins run before the detection engine is called, but after the packet has been decoded. Enabling and configuring the Preprocessor plug-ins to analyze the captured traffic and generate the necessary output is done using the *preprocessor* keyword in a Snort rules file. The Snort Preprocessor plug-in syntax is as follows:

```
preprocessor <name_of_preprocessor>: <configuration_
   options>
```

Table 5.9 presents the syntactical structures of three different Snort Preprocessor plug-ins.

Regardless of the type of evidence obtained or how the evidence was produced, collected, and analyzed, it must be in accordance with sound forensics procedures and be complete, authentic, admissible, reliable, and believable. In addition, the evidence cannot be "fruit from the poisonous tree." This term is important because attackers will attempt to cover their tracks. This includes changing log files or corrupting the captured network packets.

NOTE
Fruit from the poisonous tree is a term used to describe evidence obtained with the aid of information obtained illegally or from tainted sources. The logic is if the source of the evidence (the "tree") were tainted, then anything gained from it (the "fruit") would be likewise tainted.

Table 5.9 Sample Snort Preprocessor Plug-ins

Preprocessor (Examples)	Description
`preprocessor frag3_global:` `preprocessor frag3_engine: policy windows`	The frag3 preprocessor requires at least two preprocessor directives. The frag3_global preprocessor provides global configuration options. The frag3_engine instantiates the desired OS engine (for example, MS Windows).
`preprocessor stream5_global: track_udp no` `preprocessor stream5_tcp: policy windows`	The stream5 preprocessor requires at least two preprocessor directives. The stream5_global preprocessor provides global configuration options. The stream5_tcp, stream5_udp, or stream5_icmp instantiate the desired OS engine (for example, MS Windows).

`preprocessor sfscanport: \`
`proto { all } \`
`scan_type { all } \`
`sense_level { high }`
	The sfportscan preprocessor provides protocol, type of scan, and the alert severity level.

After the Snort Preprocessor Component has completely analyzed the captured packets, the Detection Engine Component receives the captured packet next. The next section, "Snort Detection Engine Component," discusses this component.

SNORT DETECTION ENGINE COMPONENT

The Snort Detection Engine Component extends the functionality of Snort using a very flexible and powerful rules language (a form of predefined signatures). After the Detection Engine Component receives the packets (including packets reassembled) from the Snort Preprocessor Component, the Snort Detection Engine Component examines the packets for content that matches the rule criteria.

The Snort Detection Engine is a customizable component. It can be configured to use either the Aho–Corasick algorithm or the Trie structure. The Aho–Corasick is a string-searching algorithm invented by Aho and Corasick. This algorithm functions similar to a dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. The Trie structure is an ordered data structure modeled like an upside-down tree that stores keys in the nodes with values below them.

The decision to use either the Aho-Corasick algorithm or the Trie structure is based on system memory and traffic performance parameters inserted into the Snort configuration file using the following syntax (for descriptions of the search method "syntax" see Table 5.10).

```
config detection: search-method [Search-Method]
```

Table 5.10 Search Method Syntax

Search Method	Description
`ac`	Aho–Corasick Full (high memory usage, best performance)
`ac-std`	Aho–Corasick Standard (moderate memory usage, high performance)
`ac-bnfa`	Aho–Corasick NFA (low memory usage, high performance)
`acs`	Aho–Corasick Sparse (low memory usage, moderate performance)
`ac-banded`	Aho–Corasick Banded (low memory usage, moderate performance)
`ac-sparsebands`	Aho–Corasick Sparse Banded (low memory usage, high performance)
`lowmem`	Low Memory Keyword *Trie* (low memory usage, low performance)

The Snort rules criteria is determined by separating Snort rules into two sections, the rule header and the rule options. The Rule Header section contains the rule's criteria based on action, protocol, the source and destination IP addresses and netmasks, the source and destination ports information, and direction operator. The Rule Option section contains alert messages and information on which parts of the packet should be inspected to determine if a match occurs and the rule action mentioned in the above rule header section is taken.

Snort version 2.8.5.1 (the version this book is based on) provides the network forensics examiner with various options for obtaining predefined Snort rules (for example, SourceFire Vulnerability Research Team Subscriber services, SourceFire Vulnerability Research Team Registered Users services, and Third-Party sources). In addition to using the above-presented options, the Snort Architecture provides a framework for the development of Snort rules.

The *Snort Rule Headers* section specifies the action Snort should perform if a match with a predefined signature occurs. The five default (noninline mode) actions are alert, log, pass, activate, and dynamic. The following is a description of each action:

- `Alert` – This action sends a predefined message, and then records the packet.
- `Log` – This action records the packet.
- `Pass` – This action instructs the system to ignore the packet.
- `Activate` – This action sends a predefined message, and then enables a dynamic rule.
- `Dynamic` – This action is idle until activated by an activate rule, and then act as a log rule.

Next, the Snort rule headers specify the remaining information that applies to the following:

- Protocol – Snort supports `TCP`, `UDP`, `ICMP`, and `IP`.
- IP addresses/ports – This portion deals with the IP address information for a given rule. The keyword `any` may be used to define any source and/or destination IP addresses and CIDR/netmasks.
- Port numbers – This portion deals with the port information for a given rule. The keyword `any` defines any source and/or destination ports.
- Direction operator – This operator can be either the one-way source to destination operator "`->`" or the bidirectional operator "`<>`."

Table 5.11 presents the three different examples of Snort rules headers.

The Snort Rule Option section is comprised of four categories. This section contains alert messages and information from which parts of the packet are inspected to determine if a match occurs. Figure 5.4 presents the four categories.

Table 5.11 Examples of Snort Rules Headers

Snort Rules Headers	Description
`Log tcp any any -> 192.168.1.0/24 23`	Log tcp traffic going from any source IP address and port number to the IP subnet 192.168.1.0 using port 23 (Telnet).
`Log tcp any any <> any any`	Log any bidirectional tcp traffic going from any source IP address and port number to any destination IP address and port number.
`Alert tcp 192.168.1.10 any -> any 443`	Alert on tcp traffic going from IP address 192.168.1.10 and any port number to any IP address using port 443 (SSL).

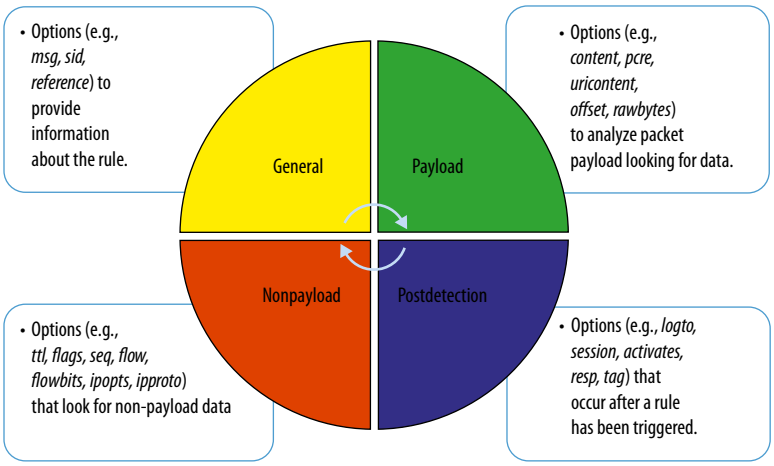


FIGURE 5.4 Snort detection rule categories

General, the first category, provides the Snort Rule Option section with information about the rule matched. For this section, eight different option keywords are available. The eight keywords are as follows: `msg`, `reference`, `gid`, `sid`, `rev`, `classtype`, `priority`, and `metadata`. Tables 5.12 and 5.13 describe two keyword examples.

Payload, the second category, provides Snort Rule Option section with 25 keywords to analyze a packet payload to search for data. The 25 keywords are as follows: `content`, `nocase`, `rawbytes`, `depth`, `offset`, `distance`, `within`, `http_client_body`, `http_cookie`, `http_header`, `http_method`, `http_uri`, `fast_pattern`, `uricontent`, `urilen`, `isdaaat`, `pcre`, `byte_test`, `byte_jump`, `ftpbounce`, `asn1`, `cvs`, `dce_iface`, `dce_opnum`, and `dce_stub_data`. Tables 5.14 and 5.15 describe keyword examples.

Nonpayload, the third category, provides Snort Rule Option section with 21 keywords to analyze nonpayload data. This typically is the metadata associated with a packet. The 21 keywords are as follows: `fragoffset`, `ttl`, `tos`, `id`, `ipopts`, `fragbits`, `dsize`, `flags`, `flow`, `flowbits`, `seq`, `ack`, `window`, `itype`, `icode`, `icmp_id`, `icmp_seq`, `rpc`, `ip_proto`, `sameip`, and `stream_size`. Tables 5.16 and 5.17 describe keyword examples.

Table 5.12 Sample Snort Rule Using `msg` Keyword

Keyword	`msg`
Syntax	`msg: "<message text";`
Example	`log tcp any any -> any 443 (msg: "TCP port 443 traffic log";)`
Description	This rule tells Snort to log any TCP destined for TCP port 443 that reaches the IDS and includes the message "TCP Port 443 trafficlog" with the log entry.

Table 5.13 Sample Snort Rule Using `reference` Keyword

Keyword	`reference`
Syntax	`reference: <id system>,<id>;`
Example	`alert tcp any any -> any 80 (msg: "WEB-IIS Microsoft IIS 5.1 and 6.0 WebDAV password bypass attempt"; content: "GET /..%c0%af/protected/protected.zip HTTP/1.1" reference:cve,2009-1535;)`
Description	This rule tells Snort to alert on any TCP source traffic destined for TCP port 80 that attempts a WebDAV password bypass and to include a URL reference link to the Common Vulnerabilities and Exposures (CVE) page for that vulnerability.

Table 5.14 Sample Snort Rule Using `content` Keyword	
Keyword	`content`
Syntax	`content: [!] "<content string>";`
Example	`alert tcp any any -> any 80 (content: "Password";)`
Description	This rule tells Snort to alert on any TCP source traffic destined for TCP port 80 that contains the word "password." Additional options about the `content` keyword is listed below: • The `content` keyword uses the Boyer–Moore pattern-matching algorithm. A string-search algorithm was developed by Boyer and Moore in 1977. • It is case sensitive. • It can contain mixed text and binary data. • The binary data, enclosed within the pipe (|) character, represented as hexadecimal numbers. • Multiple content rules can be specified in one rule. • If proceeded by an exclamation mark (!), the alert will be triggered on packets that do not contain this content.

Table 5.15 Sample Snort Rule Using `pcre` Keyword	
Keyword	`pcre`
Syntax	`pcre:[!]"(/<regex>/|m<delim><regex><delim>)[ismxAEGRUBPHMCO]";`
Example	`alert ip any any -> any any (pcre:"/BLAH/i";)`
Description	This rule tells Snort to perform a case-insensitive search for the string BLAH in the payload from any IP source address traffic destined for any destination address. The Perl Compatible Regular Expressions is used by the `pcre` keyword. For more detail pertaining to pcre regular expressions, check out the PCRE Web site: www.pcre.org.

Table 5.16 Sample Snort Rule Using `flow` Keyword	
Keyword	`flow`
Syntax	`flow: [(established|stateless)] [,(to_client|to_server|from_client|from_server)] [,(no_stream|only_stream)];`
Example	`alert tcp any any -> any 21 (msg:"Incoming FTP Change Directory command detected"; \ flow:from_client; content:"CWD incoming"; nocase;)`
Description	This rule tells Snort to trigger on any client request destined for TCP port 21 containing the content "CWD incoming." This keyword is used in conjunction with TCP stream reassembly.

Table 5.17 Sample Snort Rule Using `sameip` Keyword	
Keyword	`sameip`
Syntax	`sameip;`
Example	`alert ip any any -> any any (sameip;)`
Description	This rule tells Snort to trigger on any traffic where the source IP and the destination IP addresses are the same.

Table 5.18 Sample Snort Rule Using `logto` Keyword	
Keyword	`logto`
Syntax	`Logto:"filename";`
Example	`alert ip any any -> any any (sameip; logto:"SAME_IP_ADDRESS.txt";)`
Description	This rule tells Snort to trigger on any traffic where the source IP and the destination IP addresses are the same and log the results into a file named "SAME_IP_ADDRESS.txt." Snort does not support this option in binary logging mode.

Table 5.19 Sample Snort Rule Using `session` Keyword	
Keyword	`session`
Syntax	`session: [printable|all];`
Example	`log tcp any any <> any 23 (session:printable;)`
Description	This rule tells Snort to extract all printable strings in a Telnet packet. The printable keyword only prints user entered or readable data. The `session` keyword is best suited for postprocessing binary (pcap) log files.

Postdetection, the fourth category, provides the Snort Rule Option section with ten keywords used to perform actions after a rule is triggered. The ten keywords are as follows: `logto`, `session`, `resp`, `react`, `tag`, `activates`, `activated_by`, `count`, `replace`, and `detection_filter`. Tables 5.18 and 5.19 describe keyword examples.

The Snort Detection Engine can provide the network forensics examiner with evidence-based alerts describing the intrusion or security policy violation. To enable the Snort rules, which run within the Detection Engine Component, the `include` keyword must be used in the Snort configuration (snort.conf) or rules file indicated on the Snort command line (using the `-c <filename>` option). Multiple `include` keywords can be added to the file to allow multiple rules to be processed by the Snort Detection Engine. The `include` keyword instructs the Snort Detection Engine to read the contents of the named file and add the contents in the place where the include statement appears in the file. An example for using Snort rules is available in the default snort.conf file downloaded during the installation of Snort. The syntax for the Snort rule to be included in the Snort configuration (snort.conf) or rules file is as follows:

```
include <include file path/name>
```

As stated earlier at the beginning of this section, various predefined Snort rules (for example, SourceFire Vulnerability Research Team Subscriber

services, SourceFire Vulnerability Research Team Registered Users services, third-party sources) are available for the network forensics examiner. The rules are available from multiple sources or the network forensics examiner can develop their own rules. The Snort Architecture provides a framework for the development of Snort rules. To obtain precreated rules for various malicious software attacks, the examiner should visit the following Web site: www.snort.org/snort-rules/?#rules.

Just like in the case of the evidence obtained during the Snort Preprocessor Phase, the type of evidence obtained in the Detection Phase and how the evidence was produced, collected, and analyzed must be in accordance with sound forensics procedures and be complete, authentic, admissible, reliable, and believable. After the Detection Engine Component has completely analyzed the captured packets, the Alert/Logging Component receives the alert or log data.

NETWORK FORENSICS EVIDENCE GENERATED WITH SNORT

A network forensics investigation that entails the use of Snort involves three forms of data which the network forensics examiner must address within the court of law. The first form of data is the capturing or captured binary network sniffer data. During this stage, the network forensics examiner or the organization must prove that the gathered data was obtained using business record procedures (which include nontainted equipment). The second form of data, which occurs during the Preprocessor and Detection Engine Components stages, is the preprocessor and detection rule criteria used to identify the security intrusion or security violation. The final form of data is the IDS alerts generated and saved as a log file or in a database.

The various forms of Snort generated evidence collected during network forensics investigations require the network forensics examiner to teach organizations how to produce and handle digital or electronically generated evidence before the organization experiences a security incident, if possible. The teaching process entails making sure the organization understands the requirements for having the court accept evidence obtained during an investigation. As a result, the network forensics investigator must plan for and address this issue early on, before the collection of any must network-based evidence within the organization. The network forensics examiner must ensure organizations are familiar with the four principles of network forensics evidence. The following is a list of the four principles:

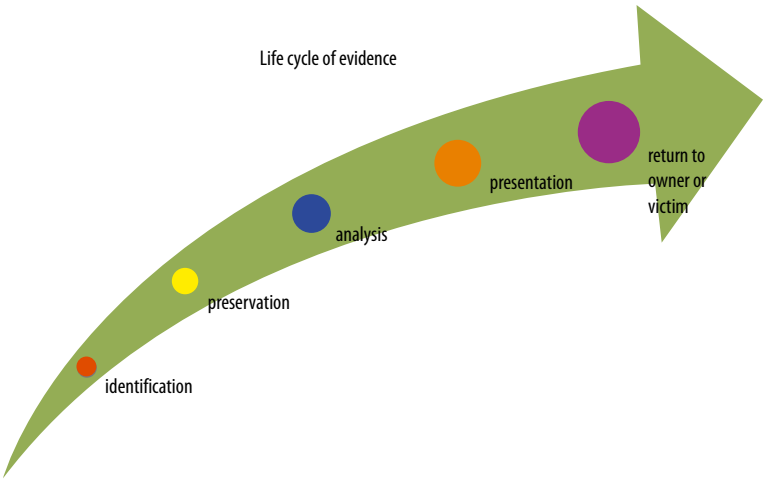
- Understanding the Life Cycle of Evidence
- Adhering to the Rules of Evidence Criteria

- Knowing the Uniqueness of Digital Evidence
- Submitting of Computer Records

The first principle, Understanding the Life Cycle of Evidence, requires all parties involved in the investigation understand that evidence has different life cycle phases and everyone must properly follow each phase in accordance with sound forensics procedures. Figure 5.5 presents the five phases of the life cycle of evidence.

The second principle, Adhering to the Rules of Evidence Criteria, requires organizations to collect and submit both inculpatory and exculpatory evidences. Inculpatory evidence is evidence that supports a given theory (for example, there is child porn on the hard drive). Exculpatory evidence is evidence that contradicts a given theory (for example, the access time for various files proves the suspect did not commit the crime). Regardless whether the evidence is inculpatory or exculpatory, all evidence should be treated equally and consistently. Organizations should apply the same security and accountability controls for evidence to comply with state's rules of evidence or with the Federal Rules of Evidence. Figure 5.6 presents the five stages of Rules of Evidence Criteria. The stages are described in Table 5.20.

The third principle, Knowing the Uniqueness of Digital Evidence, emphasizes that digital or electronic evidence, unlike other physical evidence, can be changed more easily. The only way to detect these changes is to compare the original data, maintained using a Chain of Custody form, with a duplicate using a court accepted Cryptographic Hash Integrity Algorithm (for example, MD5 and Secure Hash Algorithm).



■ **FIGURE 5.5** The life cycle of evidence

■ **FIGURE 5.6** The stages of rule of evidence criteria

Table 5.20 The Stages of Rule of Evidence Criteria

Stage	Title	Description
1	Admissible	Evidence must be able to be used in court or elsewhere.
2	Authentic	Evidence relates to incident in relevant way and accurate.
3	Complete	Inculpatory and exculpatory evidences must be presented.
4	Reliable	There should be no doubts or questions about authenticity and veracity of the evidence.
5	Believable	The evidence must be clear, easy to understand, and believable by a jury and/or judge.

The fourth principle, Submitting of Computer Records, requires the organization to ensure collected evidence can be admissible. Most courts have interpreted computer records as hearsay evidence. This trend is changing and computer records are being accepted as direct evidence. However, for the network forensics examiner and the organization, the hearsay rule is a very important hurdle to crossover. Computer records are divided into two groups: computer-generated records and computer-stored records. Most courts consider computer-generated records as admissible if they qualify as a business record exception. However, if the network forensics examiner wishes to submit computer-stored records as authentic, the person offering the records must demonstrate that the individual who created the data and the data itself is reliable and trustworthy.

NOTE

Direct evidence is any statement or entity introduced to prove a fact that stands on its own merit and does not need any supportive or backup information to refer to.

Hearsay evidence is any statement heard out-of-court and presented in court to prove the truth of an allegation. However, there are court admissible exceptions to the general rule against hearsay.

Computer-generated records are data the system automatically or manually can generate and maintain, such as system log files and proxy server logs. The records must be output generated from computer applications/processes. It, usually, is not data an individual inputs or generates.

Computer-stored records are electronic or digital data that an individual inputs or generates and saves using electronic media on a computer, such as a spreadsheet or word processing document.

You can find additional information at the U.S. Department of Justice Web site (www.cybercrime.gov) or the Searching and Seizing Computers and Obtaining Electronic Evidence Manual (www.cybercrime.gov/ssmanual/05ssma.html)

SUMMARY

In summary, five sections were discussed regarding the use of Snort as a network forensics investigation tool. It commenced with an overview of the various security controls. Specifically, it indicated that Snort is a detective-technical security control, used by organization's security teams and network forensics examiners to monitor network and/or system activities for malicious activities or security policy violations.

Second, this chapter provided an IDS overview, the main type of IDSes and the IDS Matrix diagram. After the IDS overview, the four phases of the Snort Architecture was provided. The four phases are the Sniffer Component, Preprocessor Component, Detection Engine Component, and the Alert/Logging Component. In addition, in this section, Snort execution procedures were presented for real time or playback analysis.

The next two sections, "Snort Preprocessor Component" and "Snort Detection Engine Component," provided descriptions of the Snort Preprocessor plug-ins and the Snort rule language, how to use the Snort plug-ins and rules. The final section, "Network Forensics Evidence Generated with Snort," entailed ensuring the three forms of Snort evidence is admissible as evidence and not classified as hearsay evidence.