



9 Netzwerkbasierte Intrusion-Detection-Systeme

Ein Netzwerk-Intrusion-Detection-System (NIDS) stellt einen wichtigen Teil in der Sicherheitsstruktur eines Netzwerkes dar. Sie sind in der Lage, den Netzwerkverkehr auf ungewöhnliche und möglicherweise bösartige Inhalte hin zu untersuchen.

Hierbei können unterschiedlichste Aspekte betrachtet werden. So bieten viele NIDS die Möglichkeit, die Header der IP-Pakete bezüglich nicht RFC-konformer Werte zu inspizieren. Einige NIDS ermöglichen auch die Analyse des Paketinhaltes auf verdächtige Zeichenketten. In diesem Kapitel werden Snort und ARPwatch vorgestellt.

9.1 Aufgaben eines NIDS

Netzwerkbasierte Intrusion-Detection-Systeme (NIDS) versuchen anhand eines Regelsatzes oder einer Datenbank mit Fingerabdrücken bekannter Angriffe verdächtige Vorkommnisse im Netzwerkverkehr zu erkennen. Hierbei stehen sie in einem ständigen Wettstreit mit dem Angreifer, welcher versucht, das NIDS zu täuschen, zu umgehen oder seinen Angriff so zu verschleiern, dass es ihn nicht erkennen kann.

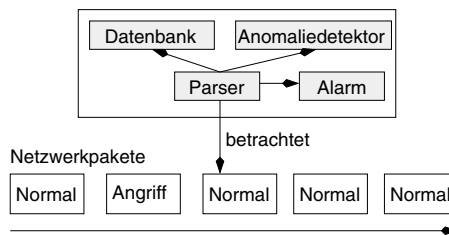


Abbildung 9.1 Funktion eines netzwerkbasierten Intrusion-Detection-Systems.

Abbildung 9.1 zeigt die Funktion eines netzwerkbasierten Intrusion-Detection-Systems. Das System besitzt einen Parser, der sämtliche Pakete aufnimmt und untersucht. Dazu greift der Parser auf eine Datenbank bekannter Angriffe und häufig einen Anomaliedetektor zurück. Wird ein Paket gefunden, welches einen Angriff darstellen kann, so schlägt der Parser Alarm.

9.1.1 Einfache Paketanalyse

Die meisten netzwerkbasierten Intrusion-Detection-Systeme, kommerzielle wie freie, begannen als einfache Paketanalytoren. Ihre Funktionalität ging nur wenig über die von *tcpdump* hinaus. Sie betrachteten den IP-Header und, wenn vorhanden, den UDP- oder TCP-Header des Paketes. Zusätzlich konnten einige den Inhalt des Paketes auf Signaturen untersuchen. So konnten sämtliche Angriffe entdeckt werden, welche Signaturen aufwiesen, die sich auf ein Paket beschränkten.

Im Folgenden sehen Sie zwei Beispiele für derartige Signaturen:

1. Ein SYN/FIN-Paket: SYN/FIN-Pakete wurden häufig bei Angriffen anstelle eines SYN-Paketes eingesetzt. Hierbei nutzt der Angreifer die Tatsache aus, dass in der Vergangenheit einige Firewall-Produkte ein SYN-Paket an der Tatsache erkannten, dass lediglich das SYN-Bit in dem TCP-Header gesetzt war. Routete nun der Router sämtliche Pakete mit der Ausnahme reiner SYN-Pakete (die einen Verbindungsaufbau von außen darstellen), so fiel auch ein SYN/FIN-Paket, in dem ja zwei Bits im TCP-Header gesetzt sind, darunter. Die meisten Betriebssysteme, welche ein SYN/FIN-Paket erhalten, reagieren, als handle es sich um ein reines SYN-Paket und bauen trotz FIN-Bit die Verbindung auf. Heutzutage werden SYN/FIN-Pakete von allen NIDS erkannt.

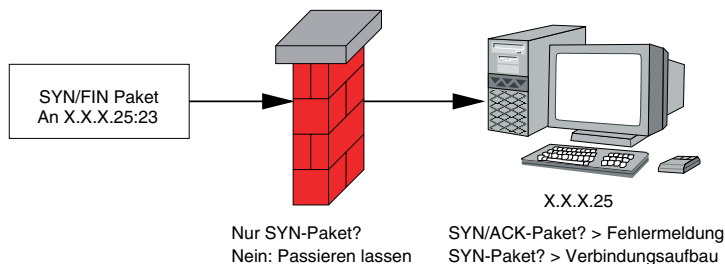


Abbildung 9.2 SYN/FIN-Angriff

2. Phonebook-Angriff phf: Mehrere Versionen des originalen NCSA-Webservers und frühe Versionen des Apache-Webservers enthielten zu Demonstrationszwecken ein Telefonbuch-CGI-Programm. Dieses in C geschriebene CGI-Programm verwendet eine Funktion `escape_shell_cmd()`, um gefährliche Metazeichen (`&;`'|*?~<>^()[]{}$`) aus der URL filtern, bevor UNIX-Befehle aufgerufen werden. Da diese Funktion das Newline (`\n, %0a`) ignorierte, bestand die Möglichkeit, jeden UNIX-Befehl auszuführen und alle UNIX-Dateien anzuzeigen. In der Abbildung 9.3 wird das in dem Angriff auftretende Paket gezeigt. Der Angriff kann recht gut an dem Vorkommen der Zeichenkette `/cgi-bin/phf?` erkannt werden.

```
GET /cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd
```

0x0000	4500	01fe	6fb5	4000	4006	cb42	7f00	0001	E...o.@..B....
0x0010	7f00	0001	831c	0050	9a5c	c9e5	9a71	41f7P.\...qA.
0x0020	8018	7fff	d3c0	0000	0101	080a	0018	a657W
0x0030	0018	a657	4745	5420	2f63	6769	2d62	696e	...WGET./cgi-bin
0x0040	2f70	6866	3f51	616c	6961	733d	7825	3061	/phf?Qalias=x%0a
0x0050	2f62	696e	2f63	6174	2532	302f	6574	632f	/bin/cat%20/etc/
0x0060	7061	7373	7764	2048	5454	502f	312e	310d	passwd.HTTP/1.1.

Abbildung 9.3 Phonebook-Angriff

9.1.2 Defragmentierung

Sehr früh wurde von den Angreifern erkannt, dass ein NIDS, welches auf einem einfachen Paketanalysator beruht, mithilfe der Fragmentierung des Paketes umgangen werden konnte. Das Paket, welches den Angriff durchführte, wurde so in verschiedene Fragmente aufgeteilt, dass die Signatur nun auf mehrere Fragmente verteilt vorlag (weitere Informationen weiter unten in diesem Kapitel und im Anhang). Sämtliche NIDS, die nicht in der Lage sind, diese Fragmente vor der Untersuchung zu sammeln und zu defragmentieren, werden nun den Angriff nicht erkennen können. Ein Werkzeug, welches diesen Angriff durchführen kann, ist *fragrouter*.

Als Beispiel soll nun das eben gezeigte Paket zum Angriff des *phf.cgi*-Skripts in seiner fragmentierten Form gezeigt werden. Es führt immer noch zur Ausnutzung der Sicherheitslücke. Jedoch muss das NIDS nun in der Lage sein, die Signatur über mehrere Fragmente zu erkennen.

```
GET /cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd
```

0x0000	4500	001c	e612	2004	4006	f112	c0a8	01fd	E.....@.....
0x0010	c0a8	0065	4745	5420	2f63	6769			...eGET./cgi

0x0000	4500	001c	e612	2005	4006	f111	c0a8	01fd	E.....@.....
0x0010	c0a8	0065	2d62	696e	2f70	6866			...e-bin/phf

0x0000	4500	001c	e612	2006	4006	f110	c0a8	01fd	E.....@.....
0x0010	c0a8	0065	3f51	616c	6961	733d			...e?Qalias=

0x0000	4500	001c	e612	2007	4006	f10f	c0a8	01fd	E.....@.....
0x0010	c0a8	0065	7825	3061	2f62	696e			...ex%0a/bin

0x0000	4500	001c	e612	2008	4006	f10e	c0a8	01fd	E.....@.....
0x0010	c0a8	0065	2f63	6174	2532	302f			...e/cat%20/

Abbildung 9.4 Phonebook-Angriff mit fragmentierten Paketen (erzeugt mit *fragrouter -F1*)

9.1.3 TCP-Streamreassemblierung

Einige Angriffe lassen sich nur erkennen, wenn das Paket in seinem Kontext der gesamten Verbindung gesehen wird. Es ist jedoch im Fall von TCP ohne weiteres möglich, die Informationen Buchstabe für Buchstabe in jeweils einem TCP-Segment und damit auch in einem Paket zu übertragen. Ein Mechanismus zur Defragmentierung hilft dem IDS in diesem Fall nicht, den Angriff zu erkennen. Im Folgenden wird ein drittes Mal der *phf*-Angriff bemüht. In der Abbildung 9.5 werden die Pakete gezeigt, die entstehen, wenn jeder Buchstabe in seinem eigenen Segment versendet wird. Die Nutzdaten sind fett markiert. Der empfangende Rechner wird in seinem TCP-Stack die komplette Session reassemblieren und an die Applikation weiterleiten. Der Angriff ist weiterhin erfolgreich. Wenn das NIDS ebenfalls die TCP-Session reassemblieren kann, so kann es den Angriff auch erkennen.

GET /cgi-bin/phf?Qalias=x%0a/bin/cat%20/etc/passwd

0x0000	4500	0035	3cea	4000	4006	7a26	c0a8	01fd	E..5<.@.@.z&....
0x0010	c0a8	0065	040a	0050	5813	5ea0	b109	f9ee	...e...PX.^.....
0x0020	8018	16d0	f3c1	0000	0101	080a	0017	6279by
0x0030	0020	d8b8	47						G ...

0x0000	4500	0035	3cea	4000	4006	7a26	c0a8	01fd	E..5<.@.@.z&....
0x0010	c0a8	0065	040a	0050	5813	5ea1	b109	f9ee	...e...PX.^.....
0x0020	8018	16d0	f5c0	0000	0101	080a	0017	6279by
0x0030	0020	d8b8	45						E ...

0x0000	4500	0035	3cea	4000	4006	7a26	c0a8	01fd	E..5<.@.@.z&....
0x0010	c0a8	0065	040a	0050	5813	5ea2	b109	f9ee	...e...PX.^.....
0x0020	8018	16d0	e6bf	0000	0101	080a	0017	6279by
0x0030	0020	d8b8	54						T ...

Abbildung 9.5 NIDS müssen Pakete reassemblieren, um einige Angriffe zu erkennen (erzeugt mit `fragrouter -T1`).

9.1.4 Dekodierung des Applikationsprotokolls

Ein sehr großer Anteil der heutigen Angriffe richtet sich nicht gegen das IP-Protokoll, sondern gegen die Applikationen selbst. Hierbei werden Sicherheitslücken im Applikationsprotokoll oder in der Applikation selber ausgenutzt. Einige Applikationsprotokolle bieten nun unterschiedliche Kodierungsmöglichkeiten der Nachricht an. Dieselbe Information kann unterschiedliche binäre Formen annehmen. Das bekannteste Beispiel ist die URL-Kodierung nach RFC 1738. Weitere Kodierungen sind UNICODE und Base64. In Abbildung 9.6 wird ein viertes und letztes Mal der *phf*-Angriff bemüht. In diesem Fall werden einzelne charakteristische Zeichen durch ihre ASCII-Darstellung ersetzt.

```
GET /%63gi-bin/%70hf?Qalias=x%0a/bin/cat%20/etc/passwd
```

```
0x0000 4500 01f9 deb7 4000 3f06 d794 c0a8 01fd E.....@.?......
0x0010 c0a8 0065 040e 0050 d7ef a0d1 31bc 4fe5 ...e...P...1.O.
0x0020 8018 16d0 1b5f 0000 0101 080a 001a 7e65 .....~e
0x0030 0023 f4df 4745 5420 2f25 3633 6769 2d62 .#..GET./%63gi-b
0x0040 696e 2f25 3730 6866 3f51 616c 6961 733d in/%70hf?Qalias=
0x0050 7825 3061 2f62 696e 2f63 6174 2532 302f x%0a/bin/cat%20/
0x0060 6574 632f 7061 7373 7764 2048 5454 502f etc/passwd.HTTP/
```

Abbildung 9.6 Wenn NIDS nicht das Applikationsprotokoll dekodieren können, wird der zweite Angriff nicht erkannt. Hier wurde das *c* durch *%63* und das *p* durch *%70* ausgetauscht. Der Angriff ist weiterhin erfolgreich.

9.1.5 Anomalie-Erkennung

Die neueste Errungenschaft vieler NIDS stellt die Anomalie-Detektion dar. Hierbei untersucht ein statistischer Algorithmus sämtliche Pakete und erkennt ungewöhnliche Pakete. Diese Pakete können zum Beispiel auf einen Portscan hinweisen, welcher von einem normalen Portscan-Detektor nicht erkannt werden könnte. Einfache Portscan-Detektoren erkennen lediglich an der Häufigkeit der Verbindungsaufnahmen und den verwendeten Ports den Portscan. Der Anomalie-Detektor kann auch stark zeitversetzte Portscans erkennen. Derartige Anomalie-Detektoren können auch ungewöhnliche Netzwerkprotokolle erkennen.

9.1.6 Critical Path

Ein NIDS-System muss die aufgenommenen Pakete verarbeiten. Erfolgt dies nicht multithreaded, so kann von einem NIDS-Prozess immer nur ein Paket verarbeitet werden. Es existiert nun ein kritischer längster Pfad, welcher von einem Paket durchlaufen werden kann. Während dieser Zeit kann das NIDS kein anderes Paket verarbeiten. Die Reduktion des kritischen Pfades ist daher sehr wichtig, damit das NIDS auch in Umgebungen mit hohem Netzwerkverkehrsaufkommen sämtliche Pakete untersuchen kann und keine verliert.

9.2 Snort

9.2.1 Einführung

Snort ist laut seinem Programmierer Martin (Spitzname: Marty) Roesch ein »Light Weight Intrusion Detection System«. Obwohl Snort als ein derartiges Leichtgewicht der NIDS gestartet ist, weist es heute fast alle wesentlichen Funktionen der vergleichbaren kommerziellen Systeme auf. Dennoch ist sein Einsatz auf unterschiedlichsten

Plattformen immer noch sehr einfach und genügsam. Snort basiert wie *tcpdump* auf der Berkeley-Paketfilterbibliothek *libpcap*. Snort filtert den Netzwerkverkehr mit Regelsätzen, welche die Fingerabdrücke möglicher Angriffe definieren. Im Falle eines Alarms bietet Snort unterschiedlichste Möglichkeiten einschließlich der Protokollierung in Dateien und verschiedensten Datenbanken, Echtzeitalarmierung mit Pop-up-Fenstern auf Microsoft Windows-Rechnern oder mittels SNMP. Eine flexible Antwort erlaubt das Beenden der gefährlichen Netzwerkverbindung. Dies kann so intelligent erfolgen, dass kein Denial of Service befürchtet werden muss.

Mit der Version 2.0 erfolgte eine komplette Neuentwicklung von Snort, die eine bis zu achtfache Geschwindigkeitssteigerung bewirkte. So kann Snort bei entsprechender Konfiguration und Optimierung ohne Probleme in 100-MBit-Netzwerken und sogar auch in 1-GBit-Netzwerken eingesetzt werden.

Snort ist wahrscheinlich das zurzeit am häufigsten eingesetzte NIDS seiner Art weltweit.

9.2.2 Geschichte

Martin Roesch begann mit der Entwicklung von Snort 1998. Er entwickelte es als *tcpdump*-Ersatz. Die Syntax und das Ausgabeformat von *tcpdump* waren für seinen Einsatz nicht aussagekräftig genug. Daher versuchte er Snort auf derselben Grundlage zu entwickeln. Hierbei handelte es sich um die Berkeley-Paketfilterbibliothek *libpcap*. Diese stellt heute die Grundlage der meisten Netzwerkanalytoren dar. Sie existiert inzwischen auch auf anderen Betriebssystemen. Daher konnte Snort auf eine Vielzahl von Betriebssystemen portiert werden: Solaris 7/8, Tru64, HP-UX, IRIX, OpenBSD, FreeBSD, Win32 und Linux.

2001 gründete Martin Roesch Sourcefire. Sourcefire bietet kommerziellen Support für Snort und führt seine Entwicklung weiter. Des Weiteren entwickelt Sourcefire eine kommerzielle grafische Management-Konsole und eine Fernwartungssuite.

9.2.3 Lizenz

Snort ist Open Source. Martin Roesch veröffentlichte es unter der GNU General Public License in der Version 2 von Juni 1991. Dies sichert den Fortbestand der Software als Open Source. Wesentliche Neuerungen, die seine Firma Sourcefire für Snort entwickelte, wurden ebenfalls unter die GNU GPL gestellt. So wurden Barnyard, das Defragmentierungs-Plug-In *frag2* und der Streamreassembler *stream4* von Sourcefire entwickelt.

9.2.4 Funktionen von Snort

Die Verwendung von Snort ist recht einfach. Jedoch sollte vorher die Funktionsweise verstanden werden. Snort ist in der Lage, als Sniffer, als Netzwerkprotokollant und als Network Intrusion Detection System eingesetzt zu werden. Diese Flexibilität be-

ruht auf der inneren Struktur von Snort. Es ist zunächst lediglich ein Produkt, welches mithilfe der *libpcap*-Bibliothek in der Lage ist, Pakete von einem Ethernet-Interface aufzunehmen. Dies kann in dem normalen Modus erfolgen, in dem die Netzwerkkarte lediglich Pakete entgegen nimmt, welche an ihre MAC-Adresse oder die Broadcast MAC-Adresse gerichtet sind, oder in dem so genannten *promiscuous mode*. Im *promiscuous mode* nimmt die Netzwerkkarte unabhängig von der MAC-Adresse alle Pakete entgegen. Der Benutzer kann nun mithilfe von Kommandozeilenoptionen das Verhalten von Snort definieren. So kann Snort die aufgenommenen IP-Pakete parsen und deren Information in Klartext anzeigen (Sniffer-Modus).

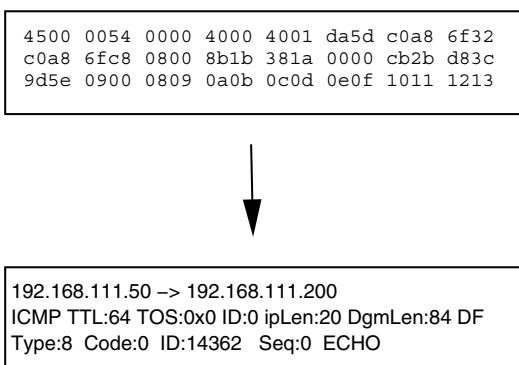


Abbildung 9.7 Snort zeigt Pakete im Klartext an.

Snort ist in der Lage, auch den Datenanteil und die Ethernet-Header anzuzeigen. Anstatt die Daten anzuzeigen, kann Snort auch die Informationen geeignet protokollieren. So steht eine Protokollierung der Pakete in Binärform als auch in Klartext zur Auswahl. In dem Binär-Modus erfolgt die Protokollierung kompatibel zu anderen Produkten wie *tcpdump* und *ethereal*. Im Klartext-Modus wird jedes einzelne Paket in einem Verzeichnis basierend auf der IP-Adresse eines der beiden beteiligten Rechner (s.u.) protokolliert.

Plug-Ins

Schließlich besteht auch die Möglichkeit, alle aufgenommenen Pakete zu inspizieren. Hierzu können Regelsätze definiert werden, die spezifische Aspekte des Paketes untersuchen. Die Detektionsfähigkeiten werden nicht von dem Snortkern selbst zur Verfügung gestellt, sondern mithilfe von Plug-Ins implementiert. Dies erlaubt einen sehr modularen und zukünftig erweiterbaren Aufbau von Snort. So können unterschiedliche Programmierer Plug-Ins für Snort schreiben. Snort unterstützt inzwischen Detektions-Plug-Ins, Präprozessoren und Output-Plug-Ins. Snort kommt mit folgenden Detektions-Plug-Ins:

- *asn1* (ab Snort 2.2)
- *byte_check* (ab Snort 2.0)

- *byte_jump* (ab Snort 2.0)
- *clientserver* (ab Snort 2.0)
- *dsize_check* (ab Snort 2.0)
- *flowbits* (ab Snort 2.1)
- *icmp_code_check*
- *icmp_id_check*
- *icmp_seq_check*
- *icmp_type_check*
- *ip_fragbits*
- *ip_id_check*
- *ipoption_check*
- *ip_proto*
- *ip_same_check*
- *ip_tos_check*
- *isdataat* (ab Snort 2.1)
- *pattern_match*
- *pcrc* (ab Snort 2.1)
- *priority* (bis Snort 1.9)
- *react*
- *respond*
- *reference* (bis Snort 1.9)
- *respond*
- *rpc_check*
- *session*
- *tcp_ack_check*
- *tcp_flag_check*
- *tcp_seq_check*
- *tcp_win_check*
- *ttl_check*

Diese Detektions-Plug-Ins untersuchen unterschiedliche Bestandteile des Paketes. Einige Pakete liegen jedoch fragmentiert vor oder der Inhalt des TCP-Stromes muss zur Analyse zunächst zusammengesetzt werden. Hierzu existieren Präprozessoren. Snort bietet die folgenden Präprozessoren:

- *anomsensor* (SPADE, bis Snort 1.9)
- *arpspoof*
- *bo*

- *conversation* (ab Snort 2.0)
- *flow* (ab Snort 2.1)
 - bietet *portscan*-Plug-In
- *frag2*
- *http_decode* (ersetzt durch *http_inspect*)
- *http_inspect* (ab Snort 2.1)
- *http_flow* (ab Snort 2.0)
- *perfmonitor* (ab Snort 2.0)
- *portscan*
- *portscan2* (ab Snort 2.0)
- *rpc_decode*
- *stream4*
- *tcp_stream2* (veraltet, ersetzt durch *stream4*)
- *telnet_negotiation*
- *unicode* (bis Snort 1.9 ersetzt durch *http_decode*)

Snort unterstützt unterschiedliche Ausgabeformate. So ist Snort in der Lage, Klartextformate, Binärformate und Datenbanken zur Protokollierung zu nutzen.

- *alert_fast*
- *alert_full*
- *alert_sf_socket* (ab Snort 2.1)
- *alert_smb* (bis Snort 2.0.x)
- *alert_syslog*
- *alert_unixsock*
- *csv*
- *database*
- *idmef* (bis Snort 1.9.x enthalten, dann als Patch)
- *log_ascii* (ab Snort 2.0)
- *log_null*
- *log_tcpdump*
- *SnmpTrap* (bis Snort 1.9.x, anschließend nur mit einem Patch)
- *unified*
- *xml* (mit Patch)

Detection-Engine on the fly

Snort implementiert seine Detection-Engine bei jedem Aufruf neu. Dabei liest Snort seine Konfigurationsdatei ein und baut seinen Detektionscode im Speicher auf; hierbei wird die Detektionsmaschine automatisch optimiert. Die Regeln werden zunächst intern sortiert und anschließend in Knoten angeordnet. Die Kenntnis des Optimierungsvorganges erlaubt die Erzeugung angepasster und damit optimaler Regeln. Der Optimierungsvorgang wird weiter unten detailliert besprochen. Mit Snort 2.0 erfolgte ein grundsätzliches Redesign der Detection-Engine. Hierdurch konnte bei Snort 2.0 eine Beschleunigung um den Faktor 3 – 7 erreicht werden. Diese Unterschiede werden auch weiter unten angesprochen.

9.2.5 Installation

Bei der Installation von Snort stehen unterschiedliche Möglichkeiten zur Verfügung. So ist Snort bereits von einigen Linux-Distributionen aufgenommen worden, so dass der Administrator lediglich das entsprechende Distributionspaket installieren muss. Enthält die Distribution keine Unterstützung für Snort oder soll Snort auf einer anderen Plattform installiert werden, kann der Download von vorkompilierten Binärpaketen von der Snort-Homepage erfolgen (<http://www.snort.org/dl/binaries>). Dort befinden sich unter <http://www.snort.org/dl/binaries/RPMS> auch RPM-Pakete für alle RPM-basierten Linux-Distributionen. Bei der Wahl der Installation eines RPM-Paketes ist es wichtig, dass die Abhängigkeiten erfüllt sind. Snort benötigt eine aktuelle Version der Bibliothek *libpcap*. Diese Bibliothek ist als RPM-Paket entweder ebenfalls in der Distribution enthalten oder kann von der Website <http://www.tcpdump.org> heruntergeladen werden. Alternativ kann eine modifizierte *libpcap*-Bibliothek verwendet werden. Diese verfügt über einen so genannten Ringpuffer. Dieser Puffer erlaubt Snort einen wesentlich schnelleren Zugriff auf die Pakete. Die Verwendung dieser Bibliothek wird im Abschnitt »libpcap-Bibliothek mit Ringpuffer« auf S. 246 besprochen.

Installation aus den Quellen

In manchen Fällen stellt die Installation eines RPM-Paketes nicht die ideale Installationsform dar. Zum einen mag das RPM-Paket der Distribution nicht den neuesten Stand der Snort-Entwicklung darstellen. Zum anderen besteht bei der Installation aus dem Quelltext die Möglichkeit, die Software gemäß den eigenen Wünschen zu konfigurieren und entsprechende Eigenschaften zu aktivieren.

Damit die Übersetzung und die Installation von Snort gelingen kann, müssen einige andere Softwarepakete zuvor auf dem System installiert werden. Unabdingbar für die Übersetzung von Snort ist das Vorhandensein der Bibliothek *libpcap*. In Abhängigkeit von den aktivierten Eigenschaften werden einige weitere Produkte benötigt.

Funktionalität	Softwarepaket	URL	configure Option
XML (bis Snort 1.9)	libxml2	xmlsoft.org	<code>--with-libxml2</code>
XML (ab Snort 2.0)	libair	http://aircert.sourceforge.net	<code>--with-libair</code>
XML	OpenSSL	www.openssl.org	<code>--with-openssl</code>
SNMP Traps	libsnmp	net-snmp.sourceforge.net	<code>--with-snmp</code>
WinPopUp (bis Snort 1.9)	Samba	www.samba.org	<code>--enable-smbalerts</code>
FlexResp	Libnet	www.packetfactory.net/Projects/Libnet	<code>--enable-flexresp</code>
ODBC-Log	unixODBC	www.unixODBC.org	<code>--with-odbc</code>
MySQL-Log	MySQL	www.mysql.org	<code>--with-mysql</code>
PostgreSQL-Log	PostgreSQL	www.postgresql.org	<code>--with-postgresql</code>
Oracle-Log	Oracle	www.oracle.com	<code>--with-oracle</code>

Tabelle 9.1: Zur Installation von Snort benötigte Pakete

Snort unterstützt nur bis zur Version 1.9.x SNMP-Meldungen. Ab der Version 2.0 ist ein zusätzlicher Patch erforderlich. Die Anwendung des Patches wird weiter unten besprochen.

Zusätzlich stehen ab der Version 2.0 die folgenden `configure`-Optionen zur Verfügung:

- `--enable-perfmonitor`. Hiermit kann ein Performance-Monitor als Präprozessor angeschaltet werden. Dieser Präprozessor kann statistische Informationen über die Arbeit von Snort ausgeben.
- `--enable-linux-smp-stats`. Hiermit stellt Snort über das Linux-Kernel `/proc`-Interface statistische Informationen zur Verfügung.

Um nun Snort zu übersetzen und zu installieren, laden Sie es zunächst von der Homepage <http://www.snort.org>, entpacken es in einem sinnvollen Verzeichnis, konfigurieren und übersetzen es.

Listing 9.1 Konfiguration, Übersetzung und Installation von Snort

```
$ cd /usr/local/src
$ wget http://www.snort.org/dl/snort-version.tar.gz
$ tar -xvzf snort-version.tar.gz
$ cd snort-version
$ ./configure --prefix=/usr --bindir=/usr/sbin --sysconfdir=/etc/snort \
  --enable-flexresp --with-postgresql --with-mysql
$ make
$ sudo make install
```

Die bei der Konfiguration zusätzlich angegebenen Optionen `--prefix`, `--bindir` und `--sysconfdir` definieren den Ort der Binär- und Konfigurationsdateien. Ein derartig konfiguriertes und übersetztes Snort-Binärprogramm ist in der Lage, eine Protokollierung sowohl in der MySQL- als auch in der PostgreSQL-Datenbank vorzunehmen. Alarmierungen können mithilfe von SNMP und Windows PopUp-Nachrichten durchgeführt werden. Zusätzlich ist Snort in der Lage, flexibel auf Verbindungen zu reagieren und diese sogar zu beenden (siehe Abschnitt »Flexible Antwort« auf S. 294).

CVS

Experimentierfreudige Leser möchten vielleicht die neuesten Errungenschaften und Modifikationen von Snort testen. Diese Gruppe kann an der Entwicklung und dem Test der Software teilhaben. Hierfür besteht die Möglichkeit, den aktuellen Code aus dem Snort CVS zu laden. Hierbei handelt es sich ebenfalls um den kompletten Quelltext von Snort. Dieser ist anschließend genauso zu behandeln, wie im letzten Abschnitt besprochen. Es besteht lediglich die Gefahr, dass er entweder nicht fehlerfrei übersetzt wird oder anschließend nicht fehlerfrei funktioniert. Der Zugang kann auf zwei verschiedene Weisen erfolgen. Zunächst wird jede Nacht ein Schnappschuss des CVS angelegt und auf der Snort-Homepage zum Download angeboten. Dieser Schnappschuss ist unter <http://www.snort.org/dl/snapshots/> verfügbar. Hierbei wird aber jeweils der komplette Quelltext-Baum geladen. Möchte man dauernd auf dem aktuellen Stand bleiben, so bietet sich der direkte anonyme CVS-Zugang an. Dieser erlaubt anschließend dann auch eine Aktualisierung lediglich geänderter Dateien. Es muss nicht jedes Mal der komplette Baum erneut heruntergeladen werden.

```
cvs -d:pserver:anonymous@cvs.snort.sourceforge.net:/cvsroot/snort login
cvs -z3 -d:pserver:anonymous@cvs.snort.sourceforge.net:/cvsroot/snort co snort
```

9.2.6 libpcap-Bibliothek mit Ringpuffer

Die *libpcap*-Bibliothek ist verantwortlich für die Sammlung der Netzwerkpakete. Hierzu werden die Pakete von der klassischen *libpcap*-Bibliothek einzeln aus dem Kernel-Speicherbereich in den Benutzer-Speicherbereich des aufrufenden Programmes (hier Snort) kopiert. Dieser Kopiervorgang ist sehr aufwändig und daher langsam. Der Kernel kann die Netzwerkpakete aber auch in einem gemeinsam genutzten (shared) Ringpuffer sammeln. Dieser Ringpuffer wird in dem Linux-Kernel über den Konfigurationsparameter `CONFIG_PACKET_MMAP` aktiviert. Eine modifizierte *libpcap*-Bibliothek erlaubt dann dem aufrufenden Programm einen direkten Zugriff auf den Ringpuffer. Es ist kein aufwändiger Kopierprozess erforderlich. Durch eine sinnvoll gewählte Größe des Ringpuffers kann möglichen Paketverlusten vorgebeugt werden.

Die modifizierte *mmap-libpcap*-Bibliothek wird von Phil Woods auf <http://public.lanl.gov/cpw/> gepflegt und zum Download angeboten. Dort befindet sich übrigens auch eine sehr wertvolle VIM-Syntax-Datei für Snort-Regeln.

Um die modifizierte Bibliothek zu verwenden, sollten Sie zunächst die Datei herunterladen und an geeigneter Stelle auspacken:

```
$ cd /usr/local/src
$ wget http://public.lanl.gov/cpw/libpcap-1.0.20040706.tar.gz
$ tar xvzf libpcap-<version>.tar.gz
$ cd libpcap-<version>
$ ./configure --prefix /usr --enable-shared
$ make
$ sudo make install
$ sudo ldconfig
```

Bei der oben angegebenen Übersetzung und Installation wird eine vorhandene *libpcap*-Version möglicherweise überschrieben. Wenn Sie dies nicht möchten, können Sie einen anderen Prefix wählen. Sie müssen dann aber darauf achten, dass Snort auch tatsächlich die modifizierte *libpcap*-Bibliothek verwendet.

Eine interessante zusätzliche Configure-Option ist die Option `--enable-ipv6`. Hiermit können Sie die *libpcap*-Bibliothek auch in IPv6-Netzwerken einsetzen.

Zunächst wird sich die Anwendung der modifizierten nicht von der klassischen *libpcap*-Bibliothek unterscheiden. Um den Ringpuffer zu aktivieren, muss vor dem Aufruf von Snort die Umgebungsvariable `PCAP_FRAMES` initialisiert werden. Diese Variable definiert, wie viele Pakete in den Ringpuffer passen und gibt hiermit direkt auch die Größe des reservierten Arbeitsspeichers vor. Wird diese Variable auf den Wert `PCAP_FRAMES=max` gesetzt, so können 32.768 Pakete gespeichert werden. Im Falle den Netzwerkmediums Ethernet (1.530 Bytes pro Frame) benötigt der Ringpuffer dann etwa 52 MByte Arbeitsspeicher.

Weitere Umgebungsvariablen, die das Verhalten der modifizierten *libpcap*-Bibliothek beeinflussen, sind: `PCAP_SNAPLEN`, `PCAP_PROMISC`, `PCAP_TO_MS`, `PCAP_RAW`, `PCAP_PROTO`, `PCAP_MADDR`, `PCAP_FRAMES`, `PCAP_VERBOSE`, `PCAP_STATS`, `PCAP_TIMEOUT` und `PCAP_PERIOD`. Diese Variablen werden jedoch im täglichen Einsatz von Snort nicht benötigt. Der interessierte Leser findet ihre Erläuterung in der Dokumentation der *libpcap*-Bibliothek.

9.2.7 Erste Anwendung

Dieses Kapitel wird Ihnen die erste Verwendung von Snort nahe bringen. Sie werden die verschiedenen Operationsmodi von Snort und die dazugehörigen Kommandozeilenoptionen kennen lernen. Es ist von Vorteil, wenn Sie auf einen Rechner zugreifen können, um die gezeigten Beispiele nachvollziehen zu können.

Snort verfügt über drei verschiedene Operationsmodi. Es kann als Sniffer eingesetzt werden. Hierbei sammelt es lediglich alle Pakete und gibt den Inhalt auf dem Bildschirm aus. Im Paketprotokollier-Modus sammelt Snort ebenfalls sämtliche Pakete, protokolliert sie jedoch auf der Festplatte. Im Network Intrusion Detection Modus

mit den BPF-Regeln von *tcpdump*. Sollen nur ICMP-Pakete angezeigt werden, so kann das erfolgen mit:

```
$ snort -vdei eth0 icmp
...
04/02-20:03:34.422722 0:7:50:B7:16:A6 -> 0:7:E:B4:1C:AB type:0x800 len:0x46
192.168.67.65 -> 192.168.1.230 ICMP TTL:254 TOS:0x0 ID:45264 IpLen:20 DgmLen:56
Type:3 Code:13 DESTINATION UNREACHABLE: ADMINISTRATIVELY PROHIBITED,
PACKET FILTERED
** ORIGINAL DATAGRAM DUMP:
192.168.1.230:3007 -> 10.10.11.49:161 UDP TTL:126 TOS:0x0 ID:7939 IpLen:20
↳ DgmLen:105
Len: 85
** END OF DUMP
00 00 00 00 45 00 00 69 1F 03 00 00 7E 11 45 B8 ....E..i....~.E.
CO A8 01 E6 0A 0A 0B 31 0B BF 00 A1 00 55 19 D0 .....1.....U..
...
```

Paketlogger

Im letzten Abschnitt wurde bereits deutlich, dass Snort mehr Anwenderfreundlichkeit bietet als *tcpdump*. Jedoch bietet *tcpdump* die Möglichkeit, die aufgenommenen Pakete in einer Datei zu protokollieren. Snort bietet ebenfalls einen so genannten Paketlogging-Modus an.

Dieser Modus wird durch die Angabe eines Protokollverzeichnisses aktiviert:

```
snort -vdei eth0 -l /var/log/snort
```

Hierbei wird von Snort die Existenz des Protokollverzeichnisses vorausgesetzt. Ist dies nicht der Fall, so beendet sich Snort mit einer Fehlermeldung. Es wird dann jedes Paket sammeln und protokollieren. Dazu erzeugt es Unterverzeichnisse, in denen die Pakete in einem lesbaren Format protokolliert werden. Dieses Format ähnelt dem Format des Sniffer-Modus. Die Unterverzeichnisse tragen als Namen eine IP-Adresse der beiden Kommunikationspartner. Snort versucht hierbei die IP-Adresse des Clients in der Client/Server-Beziehung zu wählen. Hierzu ermittelt Snort die IP-Adresse, die in der Verbindung den hohen UDP- oder TCP-Port verwendet. Um dieses Verhalten so zu beeinflussen, dass immer die externe IP-Adresse als Verzeichnisname verwendet wird, kann mit der Option *-h* das eigene (HOME) Netzwerk definiert werden:

```
snort -vdei eth0 -l /var/log/snort -h 192.168.111.0/24
```

Nun wird Snort die IP-Adresse des entfernten Kommunikationspartners als Namen des Unterverzeichnisses wählen. Handelt es sich um eine Kommunikation zwischen zwei Rechnern des eigenen Netzwerkes, so erfolgt die Auswahl aufgrund des höheren Ports.

Die Klartextprotokollierung durch Snort ist mit Vorsicht zu genießen. Snort erzeugt pro Protokoll und verwendetem Port eine Datei. Im Falle eines Portscans besteht die Gefahr, das 65.536 TCP-Ports und 65.536 UDP-Ports abgescannt und von Snort protokolliert werden. Dies erzeugt 131.072 Dateien in möglicherweise einem einzelnen Verzeichnis.

In diesen Fällen ist eine binäre Protokollierung (-b) vorzuziehen. Dieser Modus protokolliert die Pakete in dem *libpcap*-Format, welches auch von *tcpdump* oder *ethereal*¹ gelesen und geschrieben werden kann. Snort analysiert hierbei die Pakete nicht, daher wird die Angabe des Heimnetzwerkes ignoriert.

```
snort -i eth0 -b -l /var/log/snort
```

Ein erneutes Einlesen der binär protokollierten Datei kann mit Snort unter Angabe der Option -r erfolgen:

```
snort -vder /var/log/snort/packet.log
```

NIDS

Die interessanteste Anwendung von Snort ist sicherlich der Intrusion Detection Modus. Hierbei werden die Pakete entsprechend einem Regelsatz untersucht. Nur Pakete, welche von den Regeln herausgefiltert werden, werden auch protokolliert. Die Snort-Distribution enthält bereits umfangreiche Regelsätze, welche eingesetzt werden können. Für die meisten Anwendungen sind diese Regelsätze jedoch zu umfangreich und erzeugen auch zu viele falsch-positive Alarmmeldungen.

```
snort -de -l /var/log/snort/ -h 192.168.111.0/24 -c rules.conf
```

Die Datei *rules.conf* wird nun beim Start eingelesen und sämtliche Pakete werden anschließend entsprechend der Regeln untersucht. Wird kein Verzeichnis zur Protokollierung angegeben, so wird */var/log/snort* als Standardwert angenommen. Die Protokollierung erfolgt in Klartext wie im Paketlogging-Modus.

Ausgabeformate im NIDS-Modus

Snort bietet unterschiedliche Ausgabeformate im Intrusion Detection Modus an. Wenn durch den Benutzer keine Angaben zum zu verwendenden Modus gemacht wurden, so protokolliert Snort in Klartext und alarmiert ausführlich (»full alerts«). Insgesamt stehen sechs verschiedene Alarmierungsmethoden zur Verfügung:

- -b. Binär-Protokoll (*libpcap*-Format)
- -N. Keine Protokollierung
- -A fast. Schnellster Klartext-Modus, Zeitstempel, Alarmmeldung, IP-Adressen
- -A full. Standard-Modus, Zeitstempel, Alarmmeldung, Paket-Header
- -A unsock. Sendet Alarmmeldung an einen UNIX-Socket

1 Ethereal ist eine Art grafisches *tcpdump* (<http://www.ethereal.com>)

- -A console. Sendet Alarmierungen an die Konsole
- -A cmg. Alarmierungen im CMG-Stil²
- -A none Schaltet Alarmierung ab
- -s Alarmiert via Syslogd
- -M (bis Snort 1.9.x) Sendet Alarmmeldung via WinPopUp an Windows-Rechner

Eine Beispielausgabe von Snort im »Full Alert“-Modus sieht folgendermaßen aus:

Listing 9.8 Full Alert-Modus

```
[**] [111:10:1] spp_stream4: STEALTH ACTIVITY (nmap XMAS scan) detection [**]
05/06-08:28:05.513152 128.176.61.198:43164 -> 192.168.111.202:7200
TCP TTL:54 TOS:0x0 ID:14357 IpLen:20 DgmLen:40
**U*P**F Seq: 0x0 Ack: 0x0 Win: 0xC00 TcpLen: 20 UrgPtr: 0x0
```

Automatischer Start von Snort

Um als Intrusion-Detection-System eingesetzt zu werden, sollte Snort automatisch bei einem Systemstart gestartet werden. Der manuelle Start ist zu Testzwecken gut zu gebrauchen, jedoch später auf Produktionssystemen unpraktikabel.

Die meisten modernen Linux-Distributionen verwenden den System V Init-Befehl zur Verwaltung der unterschiedlichen Betriebszustände (Runlevel) und zum automatischen Start und Beenden der entsprechenden Dienste. Dieser SysVinit verwendet üblicherweise ein Shellscript mit dem Namen *rc*, um die Dienste für den entsprechenden RunLevel zu beenden und zu starten. Dieses Skript ermittelt über eine Verzeichnisstruktur aus sieben *rcX.d*-Verzeichnissen und den enthaltenen Verknüpfungen, welcher Dienst gestartet und beendet werden soll und ruft entsprechend ein Startscript des Dienstes mit dem Argument *start* oder *stop* auf.

Die Runlevel werden von den modernen Distributionen meist wie folgt verwendet:

- 0 – Halt
- 1 – Single User
- 2 – Multiuser ohne Netzwerkdienste
- 3 – Multiuser im Text-Modus
- 4 – Reserviert
- 5 – Multiuser mit grafischer Anmeldung
- 6 – Reboot

Das Startscript wird üblicherweise von der Distribution mitgeliefert, wenn Snort Teil der Distribution ist. Wurde Snort von Ihnen aus den Quellen übersetzt, so müssen Sie

² CMG ist das Kürzel des Snort-Entwicklers Chris Green. Hier wird die Alarmierung angeschaltet, die Protokollierung deaktiviert. Gleichzeitig wird die Dekodierung der Layer-2-Header (-e) und die Ausgabe des Dateninhalts (-d) aktiviert.

das Skript selbst erstellen. Hierbei sollte darauf geachtet werden, dass auf der Startzeile möglichst wenig Optionen an Snort übergeben werden. Angaben wie das eigene Netzwerk (-h) oder der Alarmierungs-Modus (-A) sollten in der Konfigurationsdatei definiert werden. Im Folgenden ist ein Beispiel angegeben. Hier wird Snort mit der Option -D aufgerufen. Das führt dazu, dass es im Daemon-Modus im Hintergrund läuft.

Listing 9.9 Snort-Startscript

```
#!/bin/sh
#
# Kommentare für Red Hat Distribution (ntsysv, chkconfig)
# snortd          Start/Stop das Snort NIDS
#
# chkconfig: 2345 40 60
# description: snort ist ein Netzwerk-Intrusion-Detection-System
#

# See how we were called.
case "$1" in
  start)
    echo -n "Starting snort: "
    cd /var/log/snort
    /usr/sbin/snort -D -c /etc/snort/snort.conf  && \
    echo " OK " || echo " Failed "
    ;;
  stop)
    echo -n "Stopping snort: "
    killall snort && \
    echo " OK " || echo " Failed "
    echo
    ;;
  restart)
    $0 stop
    $0 start
    ;;
  status)
    ps -ax | grep snor[t]
    ;;
  *)
    echo "Usage: $0 {start|stop|restart|status}"
    exit 1
esac

exit 0
```

Das Einbinden des Startscripts in den SysVinit-Prozess soll nun am Beispiel der beiden am häufigsten eingesetzten Linux-Distributionen Red Hat und SUSE vorgestellt werden:

Red Hat

Bei allen Red Hat-Distributionen bis zum Erscheinen dieses Buches befinden sich die Startscripts im Verzeichnis `/etc/rc.d/init.d`. Diese Skripts werden über Verknüpfungen aus den Verzeichnissen `/etc/rc.d/rc[0-6].d/` aufgerufen. Hierbei zeichnet sich eine Verknüpfung, welche den Dienst startet, durch ein S, und eine Verknüpfung, welche einen Dienst beendet, durch ein K aus. Alle Verknüpfungen im Verzeichnis werden entsprechend ihrer alphabetischen Reihenfolge abgearbeitet. Das bedeutet, dass bei einem Wechsel in einen anderen Runlevel zunächst alle K*-Einträge, anschließend alle S*-Einträge im Verzeichnis des entsprechenden Runlevels abgearbeitet werden.

Zur Integration des Scripts wird es zunächst in das Verzeichnis `init.d` kopiert und S-Verknüpfungen in den Verzeichnissen, in denen Snort gestartet werden soll, erzeugt. Dies sind üblicherweise die Verzeichnisse `rc[2-5].d`. Die Wahl der Startnummer sollte so erfolgen, dass mindestens das Netzwerk und der Syslogd bereits gestartet wurden. Sollen später noch andere Protokolldienste verwendet werden, z.B. Datenbanken, so ist sicherzustellen, dass diese vor Snort gestartet werden. Dadurch wird Snort nun automatisch in den entsprechenden Runleveln gestartet. Damit Snort nun auch ordnungsgemäß beendet wird, sollte eine entsprechende K-Verknüpfung in den Verzeichnissen `rc0.d` und `rc6.d` (entsprechend dem Runlevel Halt und Reboot) angelegt werden. Hier sollte sichergestellt werden, dass Snort beendet wird, bevor eine genutzte Datenbank, der Syslogd oder das Netzwerk beendet wird.

```
# mv snort.startskript /etc/rc.d/init.d/snort
# chmod 755 /etc/rc.d/init.d/snort
# cd /etc/rc.d/rc2.d
# ln -s ../init.d/snort S60snort
# cd /etc/rc.d/rc3.d
# ln -s ../init.d/snort S60snort
# cd /etc/rc.d/rc4.d
# ln -s ../init.d/snort S60snort
# cd /etc/rc.d/rc5.d
# ln -s ../init.d/snort S60snort
# cd /etc/rc.d/rc0.d
# ln -s ../init.d/snort K40snort
# cd /etc/rc.d/rc6.d
# ln -s ../init.d/snort K40snort
```

SUSE

SUSE hat mit dem Erscheinen der Distribution SUSE 8.0 das Startverhalten der eigenen Dienste modifiziert. Im Folgenden wird zunächst das Verhalten der Version 8.x besprochen. Hier werden die Startscripts im Verzeichnis `/etc/init.d` vorgehalten. Diese Startscripts werden wie bei der Red Hat-Distribution unter Verwendung von Start-(S*-) oder Stopp-(K*-)Verknüpfungen aufgerufen. Dabei werden beim Wechsel in einen Runlevel alle S*-Verknüpfungen des entsprechenden Verzeichnisses und bei einem Verlassen alle K*-Verknüpfungen aufgerufen. Es befinden sich also zum Beispiel im Verzeichnis `rc3.d` des Runlevels 3 sowohl ein K*- als auch ein S*-Eintrag für

jeden Dienst. Beim Wechsel in den Runlevel wird die S*-Verknüpfung und beim Verlassen die K*-Verknüpfung abgearbeitet.

Die SUSE-Distributionen 7.* verwenden ebenfalls die gerade vorgestellte Struktur aus Verknüpfungen, jedoch verwenden SUSE-eigene Startscripts zusätzlich die Datei `/etc/rc.config` und die Dateien in `/etc/rc.config.d/`. Jedes SUSE eigene Startscript liest die `rc.config`-Datei ein und überprüft, welcher Wert der Variablen `START_DIENST` zugewiesen wurde. Ist die Variable auf »yes« gesetzt, so wird der Dienst automatisch gestartet. Ist er auf »no« gesetzt, so wird der Dienst automatisch beendet. Das Vorhandensein der S-Verknüpfung in dem entsprechenden Verzeichnis genügt nicht für den automatischen Start des Dienstes. SUSE-Distributionen vor der Version 7.0 verwenden eine andere Zuordnung der Runlevels. Bitte lesen Sie die Kommentare in der Datei `/etc/inittab`, bevor Sie die Verknüpfungen einrichten!

Wenn Sie das selbst erstellte Startscript verwenden, ist es unerheblich, welche Version Sie einsetzen. Erzeugen Sie in dem Verzeichnis, in dessen Runlevel Snort gestartet werden soll, sowohl die S- als auch die K-Verknüpfung.

Snort-Konfiguration

Bisher wurde Snort von Hand auf der Kommandozeile gestartet. Optionen, die die Protokollierung oder die Alarmierung betrafen, wurden beim Aufruf auf der Kommandozeile angegeben. Die meisten dieser Optionen können in die Konfigurationsdatei, welche auch die Snort-Regelsätze enthält, verschoben werden. Darüber hinaus bietet die Konfigurationsdatei weitere mächtige Direktiven, die das Verhalten von Snort modifizieren können.

Listing 9.10 Beispiel Snort-Konfiguration Version 2.1

```
#
# Snort-Konfigurationsdatei
# Ralf Spenneberg
#
# $Id: snort.xml,v 1.15 2004/04/21 10:22:27 spenneb Exp spenneb $
#
# Snort Version 2.1
#
var HOME_NET 192.168.111.0/24
var EXTERNAL_NET any
var SHELLCODE_PORTS !80 !8080
var DNS_SERVERS 192.168.111.53
var INCLUDEPATH ./

# Verwende Interface eth0
config interface: eth0
# Verwende einen anderen Benutzerkontext
config set_gid: snort
config set_uid: snort
```

```

preprocessor frag2
preprocessor stream4
preprocessor flow: stats_interval 0 hash 2
preprocessor http_inspect: global \
    iis_unicode_map unicode.map 1252
preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 } oversize_dir_length 500
preprocessor flow-portscan: output-mode pktkludge
Output log_tcpdump: binary.log
Output alert_syslog: LOG_AUTH LOG_ALERT
Output database: log, mysql, user=snortuser dbname=snortdb host=localhost

Include $INCLUDEPATH/rules.conf

```

Anhand des aufgeführten Beispiels sollen nun die Möglichkeiten der Konfiguration von Snort erklärt werden.

Variablen

Häufig werden bestimmte Angaben wie IP-Adressen, Ports und Pfade in der Konfigurationsdatei angegeben. Müssen diese Angaben später modifiziert werden, so ist es erforderlich, sämtliche Vorkommen anzupassen. Die Definition von Variablen kann diesen Vorgang stark vereinfachen. Dazu wird lediglich zu Beginn des Skriptes die IP-Adresse, der Port oder der Pfad definiert und anschließend nur mit der Variablen gearbeitet. Im Beispiel wird dies recht gut deutlich. Die spätere Administration wird vereinfacht und Fehler aufgrund von Inkonsistenzen können sich seltener einschleichen.

Die Zuweisung eines Wertes an eine Variable erfolgt mit:

```
var VARIABLE wert
```

Die Auswertung der Variablen erfolgt ähnlich wie bei der Bourne Shell-Variablen mit:

```
Include $INCLUDEPATH/rules.conf
```

Bei der Zuweisung von Variablen können auch Standardwerte definiert oder bei fehlender Wertzuweisung Fehlermeldungen ausgegeben werden. Auch diese Syntax wurde aus den Shellsprachen übernommen. Beispiele für Standardwerte oder Fehlermeldungen sind:

```

var HOME_NET $(HOME_NET:-192.168.111.0/24)
var HOME_NET $(HOME_NET:?Variable HOME_NET ist noch nicht definiert!)

```

Konfigurationsdirektiven

Drei Konfigurationsdirektiven wurden in der Konfigurationsdatei angegeben. Diese definieren das Laufverhalten von Snort. Die meisten dieser Optionen können auf der Kommandozeile wie auch in der Konfigurationsdatei angegeben werden. Werden

Sie auf der Kommandozeile spezifiziert, so überschreiben sie immer die in der Datei angegebenen Werte. Hier werden die zu überwachende Netzwerkkarte und der zu verwendende Benutzerkontext spezifiziert. Die Angabe eines Benutzers erhöht die Sicherheit. Snort wird nach dem Start, der mit *root*-Privilegien erfolgen muss, die Rechte von *root* ablegen und nur noch die Rechte des angegebenen Benutzers aufweisen. Es schaltet den Benutzerkontext um auf den angegebenen Benutzer. Weist Snort eine noch nicht bekannte Sicherheitslücke auf, so kann der Angreifer nur auf die Dateien zugreifen, auf die der Snort-Benutzer zugreifen kann. Er erlangt nicht *root*-Rechte auf dem System.

Präprozessor-Plug-Ins

Präprozessoren erlauben Snort den Netzwerkverkehr ausführlicher zu analysieren. Snort wird mit einer Vielzahl von Präprozessoren ausgeliefert, die bereits in der Einleitung besprochen und in weiteren Kapiteln genauer untersucht wurden. Bei der Angabe der Präprozessoren in der Konfigurationsdatei ist jedoch auf deren Reihenfolge zu achten. Die Präprozessoren werden in genau dieser Reihenfolge auch abgearbeitet. Das bedeutet, dass IP-(frag2-)Präprozessoren vor TCP-(stream4-)Präprozessoren aufgerufen werden müssen.

Output-Plug-Ins

Snort stellt sehr flexible Ausgabemöglichkeiten zur Verfügung, welche alternativ oder gleichzeitig genutzt werden können. Hierzu können für die Protokollierung und die Alarmierung jeweils mehrere verschiedene Methoden mit verschiedenen Zielen gewählt werden. Dadurch kann Snort unterschiedliche Instanzen in unterschiedlichen Formaten alarmieren und die Protokollierung dort durchführen. Die wesentlichen Output-Plug-Ins wurden bereits vorgestellt. Bei der Angabe in der Konfigurationsdatei ist jedoch zu beachten, dass Angaben auf der Kommandozeile jegliche Konfiguration in der Datei überschreiben. Daher sollte darauf geachtet werden, dass das Startscript kein Ausgabeformat auf der Startzeile angibt. Die Konfiguration im Einzelnen und die Vor- bzw. Nachteile der einzelnen Plug-Ins werden weiter unter besprochen.

Include

Die Snort-Konfigurationsdatei kann leicht mehrere hundert Zeilen lang werden. Die mit dem Snort-Paket ausgelieferten Regelsätze zählen mehr als 1.000 Regeln. Daher kann es ab einer bestimmten Größe vom administrativen Aufwand und vom Überblick her sinnvoll werden, die Datei auf mehrere Dateien aufzuteilen. So kann in einer Datei die Verwendung der Präprozessoren und der Output-Plug-Ins beschrieben werden. Weitere Dateien enthalten die Regelsätze für Angriffe gegen UNIX-Rechner, Windows-Rechner und allgemeine Applikationsangriffe zum Beispiel auf Webserver. Diese einzelnen Dateien können dann in der Masterdatei *snort.conf* mit der Direktive Include eingebunden werden.

Kommentare

Eine wichtige Eigenschaft der Konfigurationsdatei ist die Möglichkeit, Einträge mit Kommentaren zu versehen. So können alle Einträge mit Erläuterungen versehen werden. Auch eine Dokumentation der Datei selbst mit Änderungsdatum, Autor etc. (auch über RCS) ist möglich. Schließlich soll auch die Möglichkeit hervorgehoben werden, zu Testzwecken Teile der Konfigurationsdatei durch ein Voranstellen des Kommentarzeichens (#) zu deaktivieren



Tipp:

Wenn Sie Modifikationen an der Snort-Konfigurationsdatei durchführen und anschließend Snort neu starten, so starten Sie zunächst Snort von der Kommandozeile ohne die Option `-D`. Diese Option veranlasst Snort, sich von der Konsole zu lösen und als Daemon im Hintergrund zu laufen. Jedoch wird Snort dann keine Fehlermeldungen auf der Konsole ausgeben, welche auf Syntaxfehler in der Konfigurationsdatei hinweisen können.

Konfiguration auf der Kommandozeile und in der Datei

Viele Optionen, die bisher nur auf der Kommandozeile definiert werden konnten, können ab Snort 1.8 auch in der Konfigurationsdatei definiert werden. Dies erlaubt den Aufruf von Snort lediglich mit der Angabe der Konfigurationsdatei. Hiermit können konsistente Aufrufe wesentlich leichter garantiert werden.

Snort verlangt folgendes Format:

```
config DIREKTIVE[: Wert]
```

Snort unterstützt die folgenden `config`-Direktiven:

- `alertfile`. Name der Alert-Datei. Normalerweise protokolliert Snort die Alerts in der Datei `alerts` im Protokollverzeichnis. Der Name kann mit diesem Parameter modifiziert werden.
- `alert_with_interface_name`. Snort hängt den Namen der Netzwerkkarte an die Alert-Meldung an (`snort -I`). Dies erlaubt mehreren Snort-Prozessen gleichzeitig ein Alert-Zielmedium zu verwenden und weiterhin die Snort-Prozesse zu unterscheiden.
- `bpf_file`. Snort sammelt nur Pakete, welche dem BPF-Filter aus der angegebenen Datei genügen (`snort -F datei`). Diese BPF-Filter sind identisch mit den von `tcpdump` verwendeten BPF-Filtern. So kann die Menge der Pakete, die von Snort untersucht werden soll, bereits im Vorfeld reduziert werden. Dies erhöht die Geschwindigkeit.
- `chroot`. Snort wechselt nach dem Start in das entsprechende Verzeichnis und beschränkt sein Wurzelverzeichnis auf dasselbe (`snort -t`). Der Snort-Prozess ist anschließend nicht in der Lage, auf Dateien außerhalb des Verzeichnisses zuzugreifen.

fen. Das bedeutet, dass sämtliche erforderlichen Dateien wie die Bibliotheken, Konfigurationsdateien und Protokolle sich innerhalb dieses Verzeichnisses befinden müssen. Symbolische Links sind nicht erlaubt. Diese Option erhöht die Sicherheit des Systems bei einem Einbruch über Snort. Die Konfiguration wird am Ende dieses Kapitels genauer beleuchtet.

- `checksum_mode`. Hier können die Pakete angegeben werden, deren Prüfsummen berechnet und kontrolliert werden sollen. Mögliche Werte sind `none`, `noip`, `notcp`, `noicmp`, `noudp` und `all`.
- `classification`. Hier werden die Klassifizierungstabellen erzeugt. Die Klassifizierung erlaubt die Zuordnung der Regeln zu bestimmten Angriffsklassen. Dies erleichtert die Bewertung der protokollierten Ereignisse (siehe auch den Abschnitt »Erzeugung der Regeln« weiter unten).
- `daemon`. Snort läuft als Daemon im Hintergrund (`snort -D`).
- `decode_arp`. Aktiviert die Dekodierung und die Anzeige von ARP-Paketen (`snort -a`).
- `decode_data_link`. Snort protokolliert zusätzlich den Data Link Header (`snort -e`). Hierbei handelt es sich üblicherweise um die Ethernet-Rahmendaten, zum Beispiel die MAC-Adressen der beteiligten Rechner.
- `detection`. Diese Option (ab Snort 2.1) kann die Struktur der Muster-Such-Engine modifizieren.

Dazu unterstützt diese Option die folgenden Parameter:

- `debug`. Diese Option schaltet die Debug-Funktion für die Detection Engine an. (Default: Nein).
- `no_stream_inserts`. Mit dieser Option analysiert die Detection Engine keine Pakete, die anschließend von dem `stream4`-Präprozessor zusammengebaut werden. (Default: Nein).
- `max_queue_events`. Diese Option gibt die Größe des Detection Engine-internen Event-Queues an. (Default: 5). Diese Option ist ähnlich der Konfigurationsdirektive `event_queue`.
- `search-method`. Diese Option erlaubt die Wahl des Suchalgorithmus. Im Moment unterstützt diese Option die folgenden Suchalgorithmen:
 - `ac`. Aho-Corasick basierter Algorithmus. Dies ist der alte Snort-Suchalgorithmus, wie er auch in den Snort-Version < 2.0 genutzt wurde.
 - `mwm`. Mu-Wanber basierter Algorithmus. Dieser Algorithmus wurde erstmalig testweise in den Snort 2.0.x Versionen eingeführt und erzeugt eine wesentlich geringere Speicherauslastung. Der `ac`-Algorithmus benötigt etwa zwei- bis dreimal so viel Speicher. Dies ist ab Snort 2.1 der Default-Algorithmus.
 - `lowmem`. Dieser Suchalgorithmus benötigt noch weniger Arbeitsspeicher, ist aber auch weniger effektiv.

- `auto`. Diese Funktion war zur Zeit der Drucklegung noch nicht funktionsfähig.

Beispiel:

```
config detection: search-method mwm
```

- `disable_decode_alerts`. Diese Option schaltet Alarmierungen ab, die bei der Dekodierung des Paketes erzeugt werden.
- `disable_tcptopt_experimental_alerts`. Diese Option schaltet Alarmierungen bei experimentellen TCP-Optionen ab.
- `disable_tcptopt_obsolete_alerts`. Diese Option schaltet Alarmierungen bei obsoleten TCP-Optionen ab.
- `disable_tcptopt_ttcp_alerts`. Diese Option schaltet Alarmierungen durch T/TCP-Optionen ab.
- `disable_tcptopt_alerts`. Diese Option schaltet alle Alarmierungen bei TCP-Optionen ab.
- `disable_ipopt_alerts`. Diese Option schaltet Alarmierungen bei der Verwendung von IP-Optionen ab.
- `dump_chars_only`. Snort gibt lediglich die ASCII-Darstellung der Pakete aus (`snort -C`). HEX-Ausgaben werden unterdrückt.
- `dump_payload`. Snort gibt den Inhalt des Paketes aus (`snort -d`). Normalerweise wird lediglich der Kopf (Header) des Paketes protokolliert.
- `dump_payload_verbose`. Snort gibt den gesamten Inhalt des Paketes inklusive der Data Link Layer (Ethernet-Informationen) aus (`snort -X`).
- `event_queue`. Snort kann für jedes Paket mehrere Alarmmeldungen produzieren. Ihre Anzahl und Gewichtung kann mit diesem Konfigurationsparameter eingestellt werden.
- `interface`. Dies erlaubt die Angabe der Netzwerkkarte, die von Snort verwendet werden soll (`snort -i ethX`). Bisher ist es leider nicht möglich, Snort gleichzeitig auf mehreren Netzwerkkarten zu binden. Hierzu sind im Moment mehrere Snort-Prozesse erforderlich.
- `logdir`. Snort verwendet das angegebene Protokollverzeichnis (`snort -l`).
- `min_ttl`. Alle Pakete mit einer kleineren TTL werden von Snort nicht beachtet. Ein Wert von 0 schaltet dies ab.
- `nolog`. Die Protokollierung wird abgeschaltet (`snort -N`). Es erfolgt jedoch weiterhin eine Alarmierung.
- `no_promisc`. Snort wird nicht die Netzwerkkarte in den so genannten promiscuous Modus schalten (`snort -p`). Das bedeutet, die Netzwerkkarte sieht lediglich die Pakete, die für sie aufgrund der MAC-Zieladresse bestimmt sind.
- `obfuscate`. Diese Option verbirgt die IP-Adressen bei der Klartextausgabe (`snort -O`). Ist das eigene Netz angegeben worden mit `-h` oder `reference_net`, so werden nur die eigenen IP-Adressen verborgen. Dabei werden die IP-Adressen ersetzt

durch xxx.xxx.xxx.xxx. Diese Option bietet sich für die Veröffentlichung der Daten auf Mailinglisten oder Webseiten an. Außerdem ermöglicht diese Variante eine Protokollierung in datenschutzrechtlich bedenklichen Umgebungen, in denen ein Rückschluss auf den Benutzer über die IP-Adresse möglich ist.

- `order`. Ändert die Reihenfolge von Passregeln (`snort -o`). Normalerweise werden die Passregeln von Snort als letztes abgearbeitet. Das bedeutet, dass sie wirkungslos sind. Damit die Passregeln, so wie gewünscht, vor allen anderen Regeln abgearbeitet werden, muss die Reihenfolge geändert werden. Dies wurde als Sicherheitsfeature implementiert. Snort arbeitet die Regeln in der angegebenen Reihenfolge ab. Standardreihenfolge ist:

```
config order: activation dynamic alert pass log
```

- `pkt_count`. Snort sammelt lediglich die angegebene Menge von Paketen und beendet sich anschließend (`snort -n`).
- `quiet`. Snort gibt bei einem Start von der Kommandozeile keine Start/Status-Informationen aus (`snort -q`).
- `reference`. Hiermit kann ein weiteres Referenzsystem (wie CVE) zu Snort hinzugefügt werden.
- `reference_net`. Definiert das eigene Netzwerk (`snort -h`). Alle Protokolleinträge werden nun so durchgeführt, dass die entsprechende Gegenseite als Client angesehen wird. Die Protokollverzeichnisse tragen also als Namen die IP-Adresse der Gegenseite.
- `set_gid`. Snort wechselt in diese Gruppe und verlässt die `root`-Gruppe (`snort -g`). Bei der Verwendung dieser Option und der `set_uid`-Option (s.u.) ist zu beachten, dass der angegebene Snort-Benutzer und die Gruppe Schreib- und Leserechte auf die Regeldateien und die Protokollverzeichnisse benötigen.
- `set_uid`. Snort ändert den ausführenden Benutzer auf die angegebene UID und gibt die Rechte des Benutzers `root` ab (`snort -u`). Es gelten die gleichen Bemerkungen wie bei `set_gid`.
- `show_year`. Snort protokolliert auch das Jahr (`snort -y`). Üblicherweise protokolliert Snort lediglich Monat, Tag und Uhrzeit einer Meldung. Diese Option fügt das Jahr hinzu.
- `stateful`. Diese Option definiert, ob das Stream-Reassemblierungs-Plug-In alle TCP-Pakete untersuchen soll (Wert: `all`) oder nur Pakete, bei denen es den TCP-Handshake zum Verbindungsaufbau gesehen hat (Wert: `est`). Weitere Informationen zu diesem Plug-In folgen weiter unten.
- `umask`. Snort nutzt die angegebene Umask (`snort -m 077`). Das bedeutet, dass alle Dateien mit den entsprechenden Rechten angelegt werden. Beim angegebenen Beispiel dürfte nur der Snort-Benutzer die Dateien lesen und schreiben.
- `utc`. Verwendet für alle Protokolleinträge die Zeitzone `utc` (`snort -U`). Diese Option erleichtert die Korrelation von Snort-Ereignissen auf verschiedenen, sich in verschiedenen Zeitzonen befindlichen Snort-Sensoren.

- `verbose`. Schaltet die Protokollierung auf der Standardausgabe an (`snort -v`). Denken Sie, dass die Standardausgabe lediglich als serielles Gerät simuliert wird. Die Geschwindigkeit beträgt üblicherweise 38.400 Baud. Dies ist wesentlich weniger, als eine 10-Mbit/s-Netzwerkkarte in der Lage ist zu leisten!

9.2.8 Erzeugung der Regeln

Unabhängig davon, ob Sie Snort als RPM-Paket der Distribution installieren oder es selbst übersetzen, wird es mit einem umfangreichen Regelsatz ausgeliefert. Diese Regeln werden ständig modifiziert und erweitert. Sie stellen ein Sammelsurium aller Regeln dar, die als sinnvoll erachtet werden. In 99% aller Fälle ist eine Anwendung aller Regeln jedoch nicht sinnvoll. Die Regeln müssen angepasst und in ihrem Umfang reduziert oder erweitert werden. Damit Sie dies erfolgreich durchführen können, wird Sie dieses Kapitel in die Geheimnisse der Regelsyntax und der Entwicklung eigener Regeln an einigen Beispielen einführen. Die nächsten Kapitel werden sich anschließend mit fortgeschrittenen Regeln und den zur Verfügung stehenden Präprozessoren beschäftigen

Regelsyntax

Die Regeln definieren nun die Eigenschaften der Pakete, die von Snort untersucht werden sollen. Die Syntax ist recht einfach und erlaubt die Inspektion der IP-Adressen, Ports und auch des Inhaltes des Pakets. Jede Regel muss in einer Zeile definiert werden.



Tipp:

Ab der Version 1.8 besteht die Möglichkeit, das Zeilenende mit einem Backslash zu maskieren. So kann eine Regel doch optisch über mehrere Zeilen verteilt werden. Es ist jedoch wichtig, dass der Backslash der letzte Buchstabe vor dem Zeilenende ist.

Jede Regel besteht im Grunde aus zwei Teilen:

- Regelrumpf
- Regeloptionen

Ein Regelrumpf ist bei jeder Regel erforderlich. Die Optionen sind, wie der Name schon sagt, optional. Der Regelrumpf beschreibt die beiden Kommunikationspartner, das »Wer mit wem« der Verbindung. Die Optionen beschreiben den Inhalt, das »Was?«, der Kommunikation. Üblicherweise bestehen Regeln sowohl aus einem Rumpf als auch aus Optionen. In seltenen Fällen kann eine Regel lediglich aus einem Regelrumpf bestehen. Dies kommt zum Beispiel in so genannten Passregeln vor. Die-

se missachten bestimmte Pakete vor der Anwendung der restlichen Regeln. Logregeln können alle Pakete eines bestimmten Rechners protokollieren.

```

alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket");

```

Aktion

Abbildung 9.8 Aufbau der Snortregeln: Aktion

Der Regelrumpf beginnt mit der Aktion. Diese Aktion definiert, wie Snort bei einer zutreffenden Regel reagieren soll. Snort bietet hierzu folgende Schlüsselwörter an:

- **Alert.** Snort alarmiert und protokolliert das Paket. Die Alarmierung und Protokollierung erfolgt entsprechend der gewählten Output-Plug-Ins. Bei der Standardeinstellung werden lediglich die Header des Paketes in der Alert-Datei und in dem entsprechenden Protokollverzeichnis protokolliert.
- **Log.** Snort protokolliert das Paket lediglich. Es wird keine Alarmierung durchführen.
- **Pass.** Snort betrachtet das Paket nicht weiter. So können Pakete ausgefiltert werden, die ansonsten von späteren Regeln betroffen sein würden. Diese Funktion ist ähnlich der Event-Suppression (siehe Abschnitt »Thresholding und Suppression« auf S. 339) ab Snort 2.1.
- **Activate.** Snort alarmiert und aktiviert weitere Regeln. Diese weiteren Regeln sind bereits als »Dynamic« definiert, jedoch noch nicht aktiv. Sie werden dynamisch durch diese Regel aktiviert.
- **Dynamic.** Snort kann diese Regeln dynamisch anschalten. Der Anschaltvorgang wird von »Activate«-Regeln ausgelöst. Anschließend arbeiten diese Regeln analog zu den »Log«-Regeln.
- **Eigene Definition.** Im nächsten Kapitel wird die Definition eigener Aktionen beschrieben. So ist es möglich, in Abhängigkeit von der Regel unterschiedliche Ausgabeformate und -ziele zu wählen.

Anschließend an die Aktion folgt das Protokollfeld. Dieses Feld definiert das Protokoll, welches von dem Paket verwendet werden darf.

```

alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket");

```

Protokoll

Abbildung 9.9 Aufbau der Snortregeln: Protokoll

Snort kennt bisher die folgenden Protokolle:

- IP
- ICMP
- TCP
- UDP

Weitere Protokolle können in der Zukunft hinzukommen.

An das Protokoll schließt sich die Source IP-Adresse an. Diese IP-Adresse definiert die Herkunft des zu untersuchenden böswärtigen Paketes. Die Adresse wird in der Classless Internet Domain Routing (CIDR) Notation angegeben, beispielsweise 192.168.0.0/8. Das bedeutet, dass die Netzmaske als Anzahl der von links durchgängig gesetzten Bits angegeben wird.

```

alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket");

```

Source IP

Abbildung 9.10 Aufbau der Snortregeln: Source IP

Das Format der IP-Adresse kann in folgenden Formen angegeben werden:

- 10.0.0.0/8 – Klasse A
- 172.16.0.0./16 – Klasse B
- 192.168.111.0/24 – Klasse C
- 192.168.111.200/32 – Rechner
- any – Jede IP-Adresse
- ! – Negation
- [10.0.0.0/8,172.16.0.0/16] – IP-Liste
- \$DNS_SERVERS – Angabe mit einer Variablen

Die Verwendung von IP-Listen wird seit der Version 1.7 unterstützt. Die Liste besteht aus IP-Adressen in CIDR-Notation, getrennt durch Kommata und eingeschlossen mit eckigen Klammern. Diese Liste darf keine Leerzeichen enthalten. Bei der Verwendung von Variablen muss die Variable im Vorfeld der Konfigurationsdatei definiert sein. Snort liest die Konfigurationsdatei in einem Durchlauf. Alle später genutzten Variablen müssen vorher definiert werden.

Auf die Source-IP-Adresse folgt die Angabe des Source-Ports.

```

alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket");

```

Source Port

Abbildung 9.11 Aufbau der Snort-Regeln: Source-Port

Der Port definiert den Port, den der Absender verwendet hat, um das Paket zu versenden. Gültige Formate des Quell-Ports (wie auch des Ziel-Ports (Destination Port)) sind:

- 80 – Angabe des einzelnen Ports
- !80 – Negation, alle Ports außer 80
- 33434:33690 - Port-Bereich
- :1023 – Alle Ports kleiner/einschließlich 1023
- 1024: – Alle Ports größer/einschließlich 1024
- any – Jeder Port



Achtung:

Das Protokoll ICMP verwendet keine Ports. Jedoch schreibt die Regelsyntax zwingend einen Quell- und einen Ziel-Port vor. Dies ist auch bei ICMP-Regeln erforderlich, jedoch wird dieser Wert ignoriert. Daher wird üblicherweise hier *any* eingesetzt.

```
alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket");
```

└──┬──┘
Richtung

Abbildung 9.12 Aufbau der Snortregeln: Richtung

Damit ist der Absender des Paketes erschöpfend definiert. Nun erfolgt die Angabe der Verbindungsrichtung. Hier stehen zwei Möglichkeiten zur Auswahl. Entweder handelt es sich um eine unidirektionale (->) Regel, welche Pakete von links nach rechts untersucht, oder um eine bidirektionale (<>) Regel, bei der das Paket von links nach rechts wie auch umgekehrt gesendet werden kann. Hierbei werden dann Quell-Adresse/-Port und Ziel-Adresse/-Port ausgetauscht.

```
alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket");
```

└──────────┘
Destination IP und Port

Abbildung 9.13 Aufbau der Snort-Regeln: Ziel-IP-Adresse und -Port

Es folgt nun die Ziel-IP-Adresse und der -Port. Diese werden analog der Quell-IP-Adresse und dem Quell-Port definiert. Es gelten dieselben Einschränkungen bezüglich des Formates.

Die Regeloptionen erlauben nun eine weitere Spezifikation des Paketes. Sie definieren im Wesentlichen die Funktion des Paketes. Sie werden nur dann von Snort hinzugezogen, wenn der Regelrumpf mit dem Paket übereinstimmt. Die Optionen definieren nun weitere Eigenschaften des verwendeten Protokolls (z.B. TCP-Flaggen) als auch des eigentlichen Paketinhaltes (content). Zusätzlich besteht die Möglichkeit, eine Protokollmitteilung (msg: " ") zu definieren, die bei Zutreffen der Regel zusätzlich protokolliert wird.

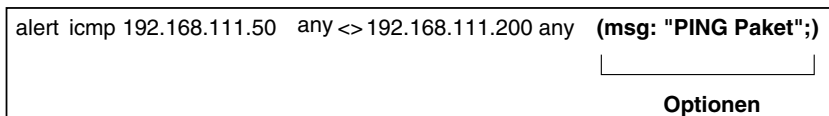


Abbildung 9.14 Aufbau der Snortregeln: Optionen

Das augenscheinlichste Merkmal bei der Syntax der Regeloptionen ist zunächst ihre Angabe in runden Klammern. Diese Klammern sind zwingend erforderlich und nicht optional. Die Regeloptionen können aus Paketattributen, Werten oder Aktionen bestehen. Hier können über die Standardaktionen hinausgehende Aktionen (z.B. SNMP Trap) definiert werden. Innerhalb der Klammern werden die einzelnen Attribute und Aktionen durch Semikola (;) getrennt. Selbst das letzte Attribut wird mit einem Semikolon abgeschlossen.

Die Attribute, welche von Snort in dem Paket untersucht werden, werden üblicherweise als ein Paar aus Schlüsselwort und Wert angegeben. Hierbei folgt auf das Schlüsselwort ein Doppelpunkt und anschließend der zu überprüfende Wert. Dies weist Snort an, die entsprechende Eigenschaft des Paketes auf Übereinstimmung mit dem angegebenen Wert zu testen. Trifft dies zu, so ist diese Option gültig.



Achtung:

Einige dieser Attribute werden als diskrete Attribute bezeichnet. Diese führen zu einem Abbruch der Regelprüfung. Sinnvollerweise werden diese Attribute zu Beginn positioniert, damit es nicht zur rekursiven Prüfung der Regel durch die Multi-Rule-Inspection-Engine kommt (siehe auch Abschnitt »Tuning« auf S. 343).

Snort unterstützt eine ganze Reihe von Attributen. Zum Zeitpunkt der Erstellung dieses Buches handelte es sich um die folgenden Attribute:

■ IP-Protokoll

- *tos. Diskret.* Testet den Wert des Type of Service-Feldes. Dieser Test ist nur bei einer exakten Übereinstimmung erfolgreich.
- *ttl. Diskret.* Testet den Wert des Time to Live-Feldes. Dieser Test ist nur bei einer exakten Übereinstimmung erfolgreich.
- *id. Diskret.* Testet die IP/Fragment-Identifikationsnummer. Häufig weist dieses Feld bei Angriffen charakteristische Einträge auf.
- *ip_proto. Diskret.* Testet das verwendete IP -Protokoll. Sowohl die numerische Angabe als auch der Protokollname entsprechend der Datei */etc/protocols* sind erlaubt.
- *sameip. Diskret.* Testet, ob Quell- und Ziel-IP-Adresse identisch sind.
- *ipopts. Diskret.* Testet auf das Vorhandensein von IP-Optionen. Diese Optionen verändern das Verhalten des Paketes und können zu Angriffen genutzt werden. Die unterstützten Werte sind:
 - *rr. Record Route.* Trägt die Router in einer Liste im IP-Header ein, die diese Pakete weiterleiten.
 - *eol. End Of List*
 - *nop. No Operation.* Wird zum Auffüllen der IP-Optionen verwendet. Der komplette IP-Header muss eine durch 4 teilbare Länge in Bytes besitzen. Er wird im Zweifelsfall mit NOPs aufgefüllt.
 - *ts. Time Stamp.* Diese Option erlaubt die Ermittlung der Laufzeit von Paketen.
 - *sec. IP Security Option.* Proprietär. Wird nur in militärischen Netzwerken genutzt. Hat keine Verbindung zu IPsec.
 - *lsrr. Loose Source Routing.* Definiert bis zu acht Router, über die das Paket unter anderen das Ziel erreichen soll.
 - *ssrr. Strict Source Routing.* Definiert exakt bis zu acht Router, über die das Paket das Ziel erreichen muss. Weitere Router sind nicht zugelassen.
 - *satid. Stream Identifier.*
 - *any. Irgendeine IP-Option ist enthalten.*
- *fragbits. Diskret.* Testet die Bits im IP-Header, die die Fragmente verwalten oder reserviert sind. Dies sind drei verschiedene Bits.
 - *R. Reserved Bit.* Dieses Bit ist reserviert und darf nicht verwendet werden.
 - *M. More Fragments (MF-Bit).* Wurde ein IP-Paket fragmentiert, so trägt jedes Fragment außer dem letzten dieses gesetzte Bit. Das deutet an, dass weitere Fragmente folgen.
 - *D. Do not Fragment (DF-Bit).* Dieses Bit zeigt an, dass der Absender einem Router verbietet, das Paket zu fragmentieren.

Diese verschiedenen Bits können nun in unterschiedlichen Varianten auftreten. Snort erlaubt den Test in folgender Form:

- fragbits: DM. Testet, ob D und M gesetzt und R nicht gesetzt sind.
- fragbits: D+. Testet, ob D und beliebige weitere Bits gesetzt sind.
- fragbits: DM*. Testet, ob D oder M gesetzt sind.
- fragbits: !D. Testet, ob D nicht gesetzt ist.
- fragoffset. Testet das Fragment-Offset-Feld. Beispiel:

```
fragoffset: > 250
```

- dsize. *Diskret*. Testet die Größe des Dateninhaltes des Paketes. Dies kann ein Test gegen einen exakten Wert sein sowie die Vergleichsoperatoren < und >. Beispiel:

```
dsize: > 500
```

■ ICMP-Protokoll

- icmp_id. *Diskret*. Testet die Identifikationsnummer von ICMP Echo-Paket (Ping). Einige Angriffswerkzeuge (Stacheldraht, Tunnelwerkzeuge etc.) verwenden statische, bekannte Identifikationsnummern.
- icmp_seq. *Diskret*. Führt denselben Test wie icmp_id auf dem ICMP-Sequenznummernwert durch.
- itype. *Diskret*. Testet den Typ des ICMP-Paketes. *Destination Unreachable* ist Typ 3. Weitere Typen siehe Anhang.
- icode. *Diskret*. Testet den Code des ICMP-Paketes. *Host Unreachable* ist Typ 3 und Code 1. Weitere Codes siehe Anhang.

■ TCP-Protokoll

- flags. *Diskret*. Testet den Zustand der sechs definierten und zwei reservierten Flags in dem TCP Flags-Byte im TCP-Header.



Achtung:

Die beiden angesprochenen reservierten Bits sind in der Vergangenheit häufig von Angreifern verwendet worden, um Informationen über die angesprochenen Betriebssysteme zu erhalten. Verschiedene Personen haben herausgefunden, dass unterschiedliche Betriebssysteme auf diese gesetzten Bits unterschiedlich antworten. Daraufhin wurden Werkzeuge entwickelt, die mithilfe von Paketen, die diese Bits verwenden, in der Lage sind, das Betriebssystem auf der anderen Seite zu bestimmen. Die bekanntesten Vertreter dieser Gattung

sind *queso* und *nmap*³. In der Vergangenheit wurden daher diese Pakete von vielen Firewalls und Intrusion-Detection-Systemen als gefährlich eingestuft. Dies hat sich geändert. RFC2481 definiert die Explicit Congestion Notification (ECN). Der Linux-Kernel 2.4 ist das erste Betriebssystem, welches ECN nutzen kann. ECN verwendet und setzt die bisher reservierten Bits im TCP-Header. Ob ECN auf Ihrem Linux-Rechner möglich und aktiviert ist, lässt sich an dem Vorhandensein und dem Wert der Datei `/proc/sys/net/ipv4/tcp_ecn` ablesen. Ist die Datei vorhanden und der Wert 1, so ist ECN aktiviert.

Folgende Flags können getestet werden: F (FIN), S (SYN), R (RST), P (PSH), A (ACK), U (URG), 2 (2. reserviertes Bit), 1 (1. reserviertes Bit), 0 (kein Bit gesetzt). Die logische Gruppierung dieser Bits erfolgt wie bei den Fragment-Bits im IP-Header. Eine ausführliche Erläuterung der Bits und ihrer Funktionen erfolgt im Anhang.

- *seq. Diskret.* Erlaubt das Testen der Sequenznummer. Einige Angriffswerkzeuge verwenden statische Sequenznummern. Ansonsten weist diese Option keinen großen Nutzwert auf.
- *ack. Diskret.* Erlaubt das Testen der Acknowledgement-Nummer. Nmap verwendet zum Beispiel eine Ack-Nummer von 0 bei gesetztem Ack TCP Bit bei einem so genannten nmap TCP Ping. Dieser TCP Ping lässt sich wie folgt starten: `nmap -PT host`.
- *window. Diskret.* Hiermit kann die Größe des TCP-Window getestet werden.
- *flow. Diskret.* Wenn die TCP-Reassemblierung (siehe Abschnitt »Stream4« auf S. 300) aktiviert wird, kann Snort mit dieser Option nur den Verkehr in einer bestimmten Richtung beobachten. In älteren Versionen von Snort war es lediglich möglich, mit der Option `flags: A+` zu überprüfen, ob es sich um eine aufgebaute Verbindung handelte. Nun kann die Regel unterscheiden, ob es sich:
 - um eine aufgebaute Verbindung handelt: `established`,
 - um eine Antwort eines Servers an einen Client handelt: `to_client` oder `from_server`,
 - um eine Anfrage eines Clients an einen Server handelt: `from_client` oder `to_server`.

Zusätzlich kann die `flow`-Option die Regel nur auf von Stream4 zusammengesetzte Pakete angewendet werden (`only_stream`) oder auf genau diese Pakete nicht (`no_stream`). Die letztere der Optionen ist sinnvoll, wenn die Regel gleichzeitig die Größe des Paketes prüfen soll (`dsize`).

³ <http://www.nmap.org>

**Achtung:**

Das `flow`-Attribut hat bei Snort 2.1.x Probleme mit der Erkennung bei der Anwendung auf dem lokalen Loopback-Interface. Hier wird der Zustand der Verbindung nicht richtig erkannt.

- `flowbits`. Dieses Attribut (ab Snort 2.1.1) erlaubt es Snort, regelbasierend einer Netzwerkverbindung einen Status zuzuordnen. Dieser Status kann zu einem späteren Zeitpunkt von einer anderen Regel ausgewertet werden. Hierzu kennt `flowbits` sieben Optionen:

- `set`. Diese Option setzt einen Zustand auf den Wert 1.

```
flowbits: set, login;
```

- `unset`. Diese Option löscht einen Zustand.
- `toggle`. Diese Option ändert einen Zustand in sein Gegenteil.
- `isset`. Diese Option prüft, ob ein Zustand gesetzt ist.

```
flowbits: isset, login;
```

- `isnotset`. Diese Option prüft, ob ein Zustand nicht gesetzt ist.
- `reset`. Diese Option setzt alle Zustände zurück. Dieses Attribut ist immer erfolgreich.
- `noalert`. Diese Option erlaubt einen Abbruch der weiteren Abarbeitung der Regel. Hiermit können Regeln definiert werden, die eine normale Verbindung mit einem bestimmten Status versehen, ohne eine Alarmierung zu erzeugen.

Das `flowbits`-Attribut erlaubt es nun, sehr mächtige Regeln zu schreiben, die nur bei bestimmten Zuständen zu Alarmierungen führen.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443 (msg:"WEB-MISC SSLv3
↳ Client_Hello request"; flow:to_server,established;flowbits:isnotset,
↳ sslv3.client_hello.request; content:"|16 03|"; depth:2; content:
↳ "|01|"; depth:1; offset:5; flowbits:set,sslv3.client_hello.request;
↳ flowbits:noalert; reference:cve,2004-0120;
↳ reference:url,www.microsoft.com/technet/security/bulletin/
↳ MS04-011.msp; classtype:protocol-command-decode; sid:2520; rev:5;)
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 443 (msg:"WEB-MISC SSLv3
↳ invalid Client_Hello attempt"; flow:to_server,established;
↳ flowbits:isset,sslv3.server_hello.request; content:"|16 03|";
↳ depth:2; content:"|01|"; depth:1; offset:5; reference:cve,2004-0120;
```

- `reference:url,www.microsoft.com/technet/security/bulletin/`
- `MS04-011.mspx; classtype:attempted-dos; sid:2522; rev:7;)`

Die erste Regel prüft, ob bereits der Zustand `sslv3.client_hello.request` gesetzt wurde. Ist dies nicht der Fall und trifft die Regel zu, so setzt die Regel diesen Zustand und führt keine Alarmierung durch. Die zweite Regel prüft, ob der Zustand gesetzt ist, erkennt eine ungültige Anfrage und führt eine Alarmierung durch. Das `flowbits`-Attribut kann auch sehr gut mit dem Attribut `tag` zusammen eingesetzt werden:

```
alert tcp any any <> any $host (flowbits:isnotset,tagged;
➤ flowbits:set,tagged; tag:host,600,seconds,src;)
```

Diese Regel prüft zunächst, ob die Verbindung bereits den Zustand `tagged` besitzt. Ist dies nicht der Fall, kommt es zu einer Alarmierung und der Zustand wird gesetzt. Alle weiteren Pakete, die von dieser Regel abgearbeitet werden, führen nicht mehr zur Alarmierung, denn die Überprüfung des Zustandes durch `flowbits:isnotset,tagged;` erzeugt einen Fehler und bricht die weitere Abarbeitung der Regel ab. Diese Pakete werden daher auch `tagged`.

Das Attribut `flowbits` kann einen wesentlichen Beitrag zur Reduktion der falsch-positiven Meldungen leisten.

■ Inhalt

- `asn1`. Dieses Attribut steht ab der Version 2.2 zur Verfügung.

Dieses Attribut dekodiert die Abstract-Syntax-Notation (ASN.1) und erkennt verschiedene böartige Kodierungen. Die ASN.1-Kodierung⁴ wird in sehr vielen Anwendungen benutzt. Insbesondere SNMP und viele auf dem X.500-Standard basierende Anwendungen nutzen diese Notation. Hierzu gehören auch X.509-Zertifikate und PKIs. In der Vergangenheit wiesen bereits viele derartige Applikationen, wie zum Beispiel die OpenSSL-Bibliothek⁵, Probleme in ihren ASN.1-Parsern auf, die für einen Angriff genutzt werden konnten. Snort verfügt ab der Version 2.2 erstmalig über einen ASN.1-Dekoder, der gewisse Angriffe erkennen kann.

Eine typische ASN.1-Notation ist zum Beispiel:

```
Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {
    givenName IA5String,
    initial IA5String,
    familyName IA5String}
```

Damit könnte dann ein Name wie folgt angegeben werden:

```
{givenName "Ralf", initial "R", familyName "Spenneberg"}
```

⁴ <http://asn1.elibel.tm.fr/en/>

⁵ <http://www.securityfocus.com/bid/8732/>

Der Präprozessor unterstützt die folgenden Parameter:

- `bitstring_overflow`. Dieser Parameter erkennt ungültige Bitstring-Kodierungen.
- `double_overflow`. Dieser Parameter erkennt doppelte ASCII-Kodierungen, die mehr Platz benötigen, als ein Standardpuffer bietet.
- `oversize_length`. Dieser Parameter erfordert zusätzlich eine Längenangabe und alarmiert bei ASN.1-Typen, die länger sind.
- `absolute_offset`. Dies ist ein Offset für die Dekodierung im Paket. Hier sollte ein Offset von 0 gewählt werden für SNMP.
- `relative_offset`. Dieser Offset zählt ab dem letzten `content`-Attribut.
- `print`. Dieser Parameter gibt das Ergebnis der Dekodierung aus.

Beispiel:

```
alert udp any any -> any 161 (msg: "Oversize SNMP Length"; asn1:
  ↳ oversize_length, 10000, absolute_offset, 0;
```

- `byte_jump`. Dieses Attribut (ab Snort 2.0) erlaubt es, die Suche in einem Paket in Abhängigkeit der Daten in dem Paket durchzuführen. Hierzu kann dieses Attribut eine bestimmte Bytefolge aus dem Paket extrahieren, in eine Zahl umwandeln und die entsprechende Anzahl von Bytes in dem Paket weiterspringen, um dort einen erneuten `content`- oder `byte_test`-Test durchzuführen. Das allgemeine Format des `byte_jump`-Attributes ist:

```
byte_jump: <bytes_to_convert>, <offset> [, [relative], [big], [little],
  ↳ [string], [hex], [dec], [oct], [align]
```

Das Attribut `byte_jump` konvertiert die angegebene Anzahl an Bytes ab dem Offset in dem Paket und springt anschließend um die entsprechende Byteanzahl im Paket weiter. Weitere Tests können dann ab dieser Stelle durchgeführt werden (mit ihrem Parameter `relative`). Das Verhalten des Attributes `byte_jump` kann durch weitere Parameter noch beeinflusst werden. Der Parameter `relative` sorgt dafür, dass der Offset ab dem letzten Treffer und nicht ab dem Paketanfang gezählt wird. Die Parameter `big` (default) und `little` definieren, ob die Bytefolge als Big-Endian- oder Little-Endian-Zahl konvertiert wird. Big-Endian wird häufig auch als Network-Order bezeichnet, da es von den meisten Netzwerkprotokollen verwendet wird. Hier wird im Gegensatz zu Little-Endian das höchstwertige Byte als erstes angegeben.

Der Parameter `string` definiert, dass die zu suchenden Daten als Zeichenkette in dem Paket vorliegen. Die weiteren Parameter `dec`, `oct` und `hex` geben dabei das Zahlensystem an. Wenn der Parameter `string` nicht angegeben wird, wird eine binäre Repräsentation angenommen.

Schließlich gibt der Parameter `align` an, ob die weiterzuspringende Byteanzahl auf die nächste 32-Bit-Grenze aufgerundet werden soll. Viele Netzwerkpro-

tolkole ordnen die Daten in ihren Paketen immer in 4-Byte-Gruppen (32 Bit) an.

```
alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"RPC sadmind query
↳ with root credentials attempt UDP"; content:"|00 01 87 88|"; depth:4;
↳ offset:12; content:"|00 00 00 01 00 00 00 01|"; within:8; distance:4;
↳ byte_jump:4,8,relative,align; content:"|00 00 00 00|"; within:4;
↳ classtype:misc-attack; sid:2256; rev:3;)
```

- `byte_test`. Dieses Attribut (ab Snort 2.0) erlaubt die Prüfung einer Bytefolge in einem Paket gegen einen bestimmten Wert. Dabei kann der Wert in verschiedenen Kodierungen angegeben werden. Zusätzlich erlaubt dieses Attribut unterschiedliche Vergleichsoperationen durch die Angabe der entsprechenden Operatoren: Gleichheit (=), keine Gleichheit (!), Kleiner-Als (<), Größer-Als (>), Logische Und-Verknüpfung (&) und Logische Oder-Verknüpfung (^).

Die allgemeine Syntax des Attributes ist:

```
byte_test: <bytes_to_convert>, <operator>, <value> <offset>
↳ [, [relative], [big], [little], [string], [hex], [dec], [oct]]
```

Hierbei werden bei dem Attribut zunächst die zu konvertierenden Bytes angegeben. Dies ist die Länge der zu untersuchenden Bytefolge. Der anschließende Operator definiert, wie der angegebene Wert (`value`) mit der konvertierten Bytefolge verglichen wird. Bei einer Und- oder Oder-Verknüpfung ist der Test erfolgreich, wenn das Ergebnis größer 1 ist. Der Offset gibt an, ab welchem Byte in dem Paket die zu konvertierende Bytefolge zu finden ist.

Weitere optionale Parameter beeinflussen den Test. So definiert der Parameter `relative`, dass der Offset ab dem letzten `content`-Treffer zu zählen ist. Die Parameter `big`, `little`, `string`, `hex`, `oct` und `dec` haben dieselbe Bedeutung wie bei dem Attribut `byte_jump` (siehe oben).

```
byte_test:1,&,128,6,relative; byte_test:2,>,255,54,relative,little;
```

- `content`. Definiert eine Bytefolge, nach der im Paket gesucht wird. Bei Auftreten dieser Bytefolge ist der Test erfolgreich. Diese Bytefolge wird üblicherweise in ASCII angegeben und in Anführungszeichen eingefasst. Soll die Bytefolge binäre Informationen enthalten, so werden diese erneut in Pipe-Zeichen (|) eingefasst. Eine Negation kann durch Voranstellen des Ausrufezeichens (!) erfolgen. Wird in der Bytefolge das Anführungszeichen, der Doppelpunkt, der Backslash oder das Pipe-Zeichen benötigt, so muss es mit dem Backslash markiert werden. Der `content`-Test ist groß-/kleinschreibungs-sensitiv. Ab der Snort-Version 2.0 gewinnt dieses Feld besondere Bedeutung (siehe Abschnitt »Attribut-Reihenfolge« auf S. 346). Beispiel:

```
(content: "|4053 c0ff 4532|/bin/ksh";offset:45;depth:120;)
```

Das `content`-Attribut wird von den folgenden nachfolgenden Attributen in seiner Wirkung modifiziert: `depth`, `distance`, `isdataat`, `offset`, `nocase`, `rawbytes`, `regex` und `within`.

- `uricontent`. `Uricontent` verhält sich wie `Content`, jedoch durchsucht es lediglich den URI-Anteil eines Paketes nach seiner Normalisierung.



Achtung:

Die Normalisierung wird nach Aktivierung von den Präprozessoren `http_decode` oder `http_inspect` durchgeführt. Ohne diese Präprozessoren ist `uricontent` nicht einsetzbar. Das bedeutet, dass Inhalte wie zum Beispiel `%2a` sich nach der Normalisierung nicht mehr im Paket befinden und von der Regel nicht gesehen werden. Soll der Inhalt ohne Normalisierung geprüft werden, so ist es erforderlich, dass die Regel das Attribut `content` verwendet.

Beispiel:

```
/.%c0%af../winnt/system32/cmd.exe?dir%20c:
```

wird normalisiert zu:

```
/winnt/system32/cmd.exe?dir%20c:
```

- `content-list`. Diese Option ist fehlerhaft implementiert und sollte daher nicht mehr verwendet werden! Eigentlich erlaubt sie die Angabe einer Datei, aus der die zu suchenden Begriffe für das Attribut `react` geladen werden.
- `depth`. Der Content Test durchsucht standardmäßig das ganze Paket. `Depth` definiert die maximale Tiefe, die der Content-Test im Paket durchsuchen soll. Ebenso wie das `offset`-Attribut, bezieht sich das `depth`-Attribut auf das letzte `content`-Attribut.
- `distance`. Während das `depth`-Attribut die Suchtiefe ab dem Beginn des Paketes angibt, erlaubt dieses Attribut die Angabe der Suchtiefe als Abstand zwischen zwei `content`-Treffern. Sucht eine Regel zum Beispiel nach dem Wort `login` und maximal 50 Bytes später nach dem Wort `password`, so könnten die Optionen dieser Regel so aussehen:

```
(content:"login";content:"password";distance:50;)
```

- `isdataat`. Dieses Attribut prüft, ob sich an einer bestimmten Stelle in dem Paket tatsächlich eine bestimmte Menge an Daten befinden. So kann dieses Attribut eine bestimmte Menge Daten ab dem Start des Paketes oder ab dem letzten `content`-Treffer prüfen. Das folgende Beispiel prüft ob ab dem Inhalt `LOGIN` noch weitere 100 Bytes im Paket folgen.

```
(content:"LOGIN";isdaat:100,relative;)
```

Mit dieser Option kann Snort erstmalig eine echte Protokollanalyse durchführen. Sie können nun generische Regeln erzeugen, die »jeden« Bufferoverflow erkennen können (siehe Abschnitt »Erzeugung der Regeln« auf S. 288 und Abschnitt »False Positives/Negatives« auf S. 347).

- `offset`. Der Content-Test durchsucht standardmäßig das gesamte Paket. `offset` definiert den Beginn des zu durchsuchenden Bereiches und bezieht sich immer auf das vorhergehende `content`-Attribut.
- `depth`. Der Content Test durchsucht standardmäßig das ganze Paket. `depth` definiert die maximale Tiefe, die der Content-Test im Paket durchsuchen soll. Ebenso wie das `offset`-Attribut, bezieht sich das `depth`-Attribut auf das letzte `content`-Attribut.
- `nocase`. Der Content-Test beachtet bei Vergleichen die Groß- und Kleinschreibung. Diese Option deaktiviert das Verhalten für das letzte `content`-Attribut.
- `pcre`. Dieses sehr mächtige Attribut ist ab der Version 2.1 in Snort enthalten. Für ältere Versionen existiert unter <http://www.snort.org/dl/contrib/patches/snort-pcre/> ein Patch. Dieses Attribut erlaubt die Verwendung von Perl-Compatible-Regular-Expressions (PCRE) in Snortregeln. Eine ausführliche Behandlung von PCRE würde hier zu weit führen. Daher soll nur auf die Website <http://www.pcre.org> verwiesen werden.

Im Folgenden ist ein Beispiel für eine PCRE-Regel abgedruckt:

```
pcrc: "/name=[^\r\n]*?\. (mim|uue|uu|b64|bhx|hqx|xxe)/smi "
```

Diese Regel überprüft, ob das Paket zum Beispiel die Zeichenkette `name=virus.uue` enthält. Hierbei überprüft die Regel verschiedene Dateierweiterungen und erkennt auch unterschiedliche Groß- und Kleinschreibung (i).

- `regex`. Diese Option erlaubt die Verwendung von Wildcards (* und ?) bei dem Attribut `content`. Es handelt sich nicht um echte reguläre Ausdrücke. In Versionen vor 1.8.3 gab es Probleme im Zusammenhang mit dem Attribut `nocase`. Ab der Version 2.1.0 wurde mit dem Attribut `pcrc` eine mächtigere Version zur Verfügung gestellt. Da das `regex`-Attribut in zukünftigen Versionen wahrscheinlich nicht mehr zur Verfügung stehen wird, sollten die Regeln entsprechend modifiziert werden.
- `rawbytes`. Dieses Attribut beschränkt das `content`-Attribut auf den Paketinhalt vor der Behandlung durch die Präprozessoren. Das bedeutet, dass eine Dekodierung durch die Telnet-, RPC- oder HTTP-Präprozessoren noch nicht stattgefunden hat.

```
alert tcp $EXTERNAL_NET any -> $TELNET_SERVERS 23 (msg:"TELNET
↳ livingston DOS"; flow:to_server,established; content:"|FF F3 FF F3
↳ FF F3 FF F3 FF F3|"; rawbytes; reference:arachnids,370; classtype:
↳ attempted-dos;sid:713; rev:7;)
```

- `rpc`. Betrachtet, dekodiert und testet Remote Procedure Calls. Hierbei können die Anwendung, die Prozedur und die Version einzeln getestet werden. Jeder

der beiden letzten Parameter kann durch * als Wildcard ersetzt werden. Beispiel:

```
(rpc: 100083,*,*; msg:"RPC ToolTalk Server");
```

- **within.** Dieses Attribut definiert den maximalen Abstand zwischen zwei content-Treffern. Am einfachsten erklärt sich dieses Attribut mit einem Beispiel:

```
byte_jump:4,4,relative,align; content:"|00 01 87 03|"; within:4;
```

Hier wird zunächst mit dem Attribut `byte_jump` eine bestimmte Stelle in dem Paket angesprungen. Anschließend prüft das `content`-Attribut, ob innerhalb (`within`) der nächsten vier Bytes die angegebene Bytefolge vorhanden ist. Häufig wird dieses Attribut zusammen mit dem Attribut `distance` verwendet.

■ Ausgabe

- **msg.** Definiert eine Zeichenkette, welche bei Zutreffen der Regel mit protokolliert wird. So kann von jeder Regel eine spezifische Protokollmeldung erzeugt werden. Sonderzeichen werden mit dem Backslash maskiert.
- **logto.** Definiert eine Datei, welche zur Protokollierung dieser Regel genutzt wird. So lassen sich die Ausgaben verschiedener Regeln in einzelnen Dateien zusammenfassen. Dieses Attribut funktioniert nicht im binären Protokoll-Modus.
- **reference.** Erlaubt die Angabe einer Referenz zu einer Regel. Diese Referenz wird definiert mit der Angabe des Systems und einer systemspezifischen ID. Mehrere Referenzen können in einer Regel angegeben werden. Momentan werden unterstützt:
 - **bugtraq.** Die Bugtraq-Mailingliste: <http://www.securityfocus.com/bid/>
 - **cve.** Die Common-Vulnerabilities-and-Exposures-Datenbank (CVE) <http://cve-mitre.org/cgi-bin/cvename.cgi?name=>
 - **arachnids.** Intrusion-Event-Database arachNIDS: <http://www.whitehats.com/info/IDS>
 - **mcafee.** McAfee Virus Information Library: http://vil.nai.com/vil/dispVirus.asp?virus_k=
 - **url.** <http://> Dies erlaubt die Angabe einer wahlfreien URL.
- **sid.** Identifiziert eindeutig eine Snortregel mit der Snort-ID. Die möglichen Nummern sind in drei verschiedene Bereiche eingeteilt worden:
 - <100 – Reserviert für zukünftige Verwendung.
 - 100-1.000.000 – Regeln in der Snort-Distribution.
 - >1.000.000 – Verwendbar für lokal entwickelte Regeln.

Die Datei `sid-msg.map` ordnet die Snort-IDs den Alarmmeldungen zu:

```
128 || BACKDOOR DeepThroat 3.1 Server Status Client Request || arachnids,106
```

Hier wird der SID 128 eine Meldung und eine Referenz zugeordnet. Diese Datei wird von Werkzeugen verwendet, die die Snortmeldungen weiterverarbeiten.

- `rev`. Definiert eindeutig die Revisionsnummer der Regel. In Kombination mit der SID-Nummer erlaubt diese Angabe später die eindeutige Referenzierung der Snortregel, welche die entsprechende Ausgabe erzeugt hat.
 - `classtype`. Classtype erlaubt die Kategorisierung von Angriffen in Klassen. Die Klassifizierung muss grundsätzlich in Snort mit der Konfigurationsoption `config classification` aktiviert worden sein (s.o.). Dann kann eine Regel mit dem Attribut `classtype` einer dieser Gruppen zugewiesen werden. Snort enthält bereits einige Standardklassen, welche in der Datei `classification.config` definiert werden.
 - `priority`. Das `priority`-Attribut bietet die Möglichkeit, der Regel eine Priorität zuzuweisen. Damit kann auch die Default-Priorität einer Regel, die über ihr `classtype`-Attribut definiert wird, undefiniert werden. Die Default-Prioritäten liegen zwischen 1 (hoch) und 4 (niedrig). Diese Priorität kann später bei der Analyse der Protokolle genutzt werden.
- **Reaktion**
- `session`. *Diskret*. Erlaubt die Protokollierung des gesamten Inhaltes der Verbindung. Session kann zwei verschiedene Werte annehmen: `printable` und `all`. `Printable` gibt lediglich lesbare Informationen aus, z.B. die in einer *telnet*-Verbindung eingegebenen Kommandos, und `all` gibt alle auch nicht sichtbaren Buchstaben (in ihrem Hexadezimal-Äquivalent) aus.



Achtung:

Dies Option benötigt viele Ressourcen und sollte nicht im Online-Modus eingesetzt werden. Sinnvoller kann diese Option zur Prozessierung von binären Protokoll-dateien genutzt werden.

- `tag`. Dieses Attribut erlaubt die Protokollierung weiterer Pakete dieses Rechners nach der erfolgreichen Auswertung der Regel. So kann die Antwort eines Rechners auf bösartige Anfragen im Nachhinein untersucht werden.

**Achtung:**

Das Attribut `tag` funktioniert zurzeit noch nicht mit dem Database-Output-Plug-In `database`.

Das Attribut `tag` verlangt die Angabe des Typs, der Anzahl, der Einheit und optional der Richtung:

`tag: Typ, Anzahl, Einheit, Richtung`

- **Typ**
 - `session`. Protokolliert alle Pakete aus derselben Verbindungssitzung.
 - `host`. Protokolliert alle Pakete desselben Rechners. Diese Option erlaubt zusätzlich die Angabe der Richtung.
- **Anzahl**. Anzahl der zu protokollierenden Einheiten.
- **Einheit**
 - `packets`. IP-Pakete.
 - `seconds`. Sekunden.
- **Richtung**
 - `src`. Protokolliert alle weiteren Pakete der Quell-Adresse.
 - `dst`. Protokolliert alle weiteren Pakete der Ziel-Adresse.

**Achtung:**

Jedes weitere Paket, welches einen Alarm auslöst, wird nicht getaggt, sondern erzeugt einen neuen unabhängigen Alarm. Hier kann das `flowbits`-Attribut des `flow`-Präprozessors helfen (s.o).

- `resp`. *Diskret*. Die Flexible Response (flexible Antwort) ermöglicht Snort die aktive Reaktion auf verdächtige Verbindungen. Snort ist in der Lage, diese Verbindungen aktiv zu beenden.

**Tipp:**

Damit Snort diese Funktion anbietet, muss es mit der Option `--flexresp` übersetzt werden.

Weitere Informationen und die Anwendung werden weiter unten im Kapitel besprochen. Dieses Attribut ist in der Lage, mit folgenden Antworten auf Pakete zu reagieren:

- `rst_snd`. Sendet ein TCP-Reset an den Absender des Paketes. Damit wird der Absender informiert, dass die Verbindung beendet wurde und neu aufgebaut werden muss.
- `rst_rcv`. Sendet ein TCP-Reset an den Empfänger des originalen Paketes. Damit wird der Empfänger informiert, dass die Verbindung beendet wurde und im Zweifelsfall neu aufgebaut werden muss.
- `rst_all`. Sendet ein TCP-Reset jeweils an den Absender und den Empfänger. Dies ist die sicherste Variante, eine TCP-Verbindung zu unterbrechen.
- `icmp_net`. Sendet ein ICMP_NET_UNREACHABLE an den Absender.
- `icmp_host`. Sendet ein ICMP_HOST_UNREACHABLE an den Absender.
- `icmp_port`. Sendet ein ICMP_PORT_UNREACHABLE an den Absender.
- `icmp_all`. Sendet alle ICMP-Fehlermeldungen an den Absender.
- `react`. Dieses Attribut erlaubt über das Attribute `resp` hinausgehende Antworten.⁶ So können Anfragen von Benutzern an bestimmte Webserver blockiert und diesen Warnmeldungen gesendet werden. Momentan werden bei `react` folgende Aktionen und Argumente unterstützt, die aber noch nicht alle funktionieren:
 - `block`. Beenden der Verbindung und Senden der Mitteilung
 - `warn`. Senden der Warnung, Kein Beenden der Verbindung. *Noch nicht implementiert!*
 - `msg`. Mitteilung
 - `proxy`. Zu verwendender Proxy-Port. *Noch nicht implementiert!*
- **Modifikation der Präprozessoren**
 - `stateless`. Betrachtung des Paketes unabhängig vom Verbindungszustand des Paketes entsprechend dem stream4-Präprozessor. Ab Version 1.9 wurde das Attribut unter `flow` eingeordnet (s.o.)

⁶ Für ihren Einsatz ist ebenfalls die Aktivierung der Compile-Option `--enable-flexresp` erforderlich.

Beispiele

Im Folgenden sollen nun einige Beispiele für einfache Regeln gegeben werden. Hierbei werden zunächst die Pakete vorgestellt, die erkannt werden sollen und anschließend die Regeln entwickelt.

Ping

In diesem Beispiel wird eine Regel entwickelt, um Ping-Pakete zu erkennen. Dabei soll diese Regel nur Ping-Anfragen erkennen. Eine typische lokale Ping-Anfrage und ihre Antwort bei einem Linux-Kernel 2.4 sieht, ausgegeben durch Snort, folgendermaßen aus:

```
05/07-21:32:27.614104 192.168.111.50 -> 192.168.111.200
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14362 Seq:0 ECHO
CB 2B D8 3C 9D 5E 09 00 08 09 0A 0B 0C 0D 0E 0F .+.<.^.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#$$%&'()*+,-./
30 31 32 33 34 35 012345
```

```
05/07-21:32:27.616047 192.168.111.200 -> 192.168.111.50
ICMP TTL:255 TOS:0x0 ID:23999 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14362 Seq:0 ECHO REPLY
CB 2B D8 3C 9D 5E 09 00 08 09 0A 0B 0C 0D 0E 0F .+.<.^.....
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F .....
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F !"#$$%&'()*+,-./
30 31 32 33 34 35 012345
```

Bei dem ersten der beiden angegebenen Pakete handelt es sich um eine Ping-Anfrage (Echo Request) von 192.168.111.50 an 192.168.111.200. Diese Ping-Anfrage wird in dem zweiten Paket von 192.168.111.200 an 192.168.111.50 beantwortet. Als IP-Protokoll wird ICMP verwendet. Diese Tatsache könnte in einer Regel ausgedrückt werden als:

```
alert icmp 192.168.111.50 any <> 192.168.111.200 any
```

Bedenken Sie, dass das Protokoll ICMP keine Ports verwendet, jedoch die Regelsyntax zwingend die Angabe der Ports fordert. Es wird hier als Port `any` eingesetzt, da Snort den Port sowieso ignorieren wird. Die oben angegebene Regel würde folgende Ausgabe im Falle der angegebenen Pakete erzeugen:

```
[**] Snort Alert! [**]
05/07-21:32:27.614104 192.168.111.50 -> 192.168.111.200
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14362 Seq:0 ECHO
```

```
[**] Snort Alert! [**]
05/07-21:32:27.616047 192.168.111.200 -> 192.168.111.50
```

```
ICMP TTL:255 TOS:0x0 ID:23999 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14362 Seq:0 ECHO REPLY
```

Zum einen ist diese Ausgabe noch ein wenig nichts sagend, zum anderen dürfte die bisher verwendete Regel noch Probleme in ihrer Eindeutigkeit aufweisen. So würden von der Regel auch Fehlermeldungen vom Typ ICMP DESTINATION UNREACHABLE protokolliert werden.

Um die Ausgabe auf dem Bildschirm aussagekräftiger zu gestalten, soll eine Mitteilung ausgegeben werden. Diese Mitteilung soll Informationen über das protokollierte Paket enthalten. Als Meldung für das Attribut `msg` wird gewählt: »PING-Paket, möglicher Erkundungsversuch«. Um diese Mitteilung nun zu implementieren, wird die Regel modifiziert:

```
alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket, \
möglicher Erkundungsversuch";)
```

Werden dieselben Ping-Pakete nun von dieser Regel bearbeitet, so ändert sich die Ausgabe in der Datei `/var/log/snort/alert` folgendermaßen:

```
[**] [1:0:0] PING Paket, möglicher Erkundungsversuch [**]
05/07-21:32:27.614104 192.168.111.50 -> 192.168.111.200
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14362 Seq:0 ECHO
```

```
[**] [1:0:0] PING Paket, möglicher Erkundungsversuch [**]
05/07-21:32:27.616047 192.168.111.200 -> 192.168.111.50
ICMP TTL:255 TOS:0x0 ID:23999 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14362 Seq:0 ECHO REPLY
```

Um später eine genaue Zuordnung der Meldung zu der erzeugten Regel durchführen zu können, soll die Regel eine Identifikationsnummer und eine Revisionsnummer erhalten.

```
alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Paket, \
möglicher Erkundungsversuch"; sid:1000001; rev:1)
```

```
[**] [1:1000001:1] PING Paket, möglicher Erkundungsversuch [**]
05/07-21:32:27.614104 192.168.111.50 -> 192.168.111.200
ICMP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:14362 Seq:0 ECHO
```

```
[**] [1:1000001:1] PING Paket, möglicher Erkundungsversuch [**]
05/07-21:32:27.616047 192.168.111.200 -> 192.168.111.50
ICMP TTL:255 TOS:0x0 ID:23999 IpLen:20 DgmLen:84
Type:0 Code:0 ID:14362 Seq:0 ECHO REPLY
```

Um nun die Treffsicherheit der Regel zu erhöhen, können weitere Eigenschaften des Ping-Paketes ausgenutzt werden. Die Ping-Anfrage verwendet den ICMP Typ 8 und

den ICMP Code 0. Die Ping-Antwort verwendet den ICMP-Typ 0 und den ICMP-Code 0. Soll die Regel nun lediglich die Ping-Anfragen melden, so muss dies mit dem Attribut `itype` geprüft werden. Wenn sowohl Ping-Anfragen als auch Ping-Antworten getrennt voneinander gemeldet werden, so sind zwei Regeln erforderlich:

```
alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Anfrage, \
  möglicher Erkundungsversuch"; itype:8; sid:1000001; rev:1)
alert icmp 192.168.111.50 any <> 192.168.111.200 any (msg: "PING Antwort, \
  mögliche Informationsleckage"; itype:0; sid:1000002; rev:1)
```

Versuchen Sie die Entwicklung der Regel nachzuvollziehen und die einzelnen Schritte zu kontrollieren. Hierzu sollten Sie sich selbst die entsprechenden Pakete erzeugen und anschließend mit Snort bearbeiten:

1. Erzeugen Sie sich Ihre eigenen Ping-Pakete. Am einfachsten erfolgt dies unter Linux, indem Sie zwei verschiedene virtuelle Textkonsolen oder zwei *xterm* verwenden. Starten Sie *tcpdump* in einem der beiden Terminals:

```
tcpdump -n icmp -w pingpakete
```

2. In dem anderen Fenster starten Sie *ping* exakt einmal. Wählen Sie bitte als Rechner einen Computer, der Ihr *ping* beantwortet.

```
ping -c 1 RECHNER
```

3. Anschließend beenden Sie *tcpdump* mit `[Strg]+[C]`. Nun sollte die Datei *pingpakete* zwei ICMP-Pakete enthalten. Dies können Sie kontrollieren mit:

```
snort -vdr pingpakete
```

Webzugriff

Dieses Beispiel entwickelt eine Regel, die den Zugriff auf einen Webserver meldet. Diese Regel wird im Weiteren so entwickelt, dass Sie einen typischen Webangriff meldet. Hierbei handelt es sich um einen Zugriff auf einen so genannten Data Stream. Dieser Zugriff erlaubt auf einem verwundbaren Microsoft Internet Information Server den Zugriff auf den Quelltext einer Active-Server-Pages-(ASP-)Datei. Die Kenntnis des Quelltextes kann einen Angreifer den internen Aufbau der Webapplikation erkennen lassen oder sogar Kennwörter für einen Datenbankzugriff enthalten. Microsoft hat dieses Problem unter <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q188806> veröffentlicht und diskutiert.

Ein Zugriff auf einen Webserver wird mit dem Applikationsprotokoll HTTP durchgeführt. HTTP ist ein Protokoll, welches eine TCP-Verbindung für die Übertragung der Daten nutzt. Daher benötigen wir eine Regel, welche TCP-Pakete zwischen zwei beliebigen Rechnern untersucht. Das TCP-Protokoll ordnet den Verbindungen die Applikationen auf der Basis von Ports zu. Das bedeutet, dass eine Applikation sich an einen Port binden kann und ab diesem Moment alle Verbindungen an diesen Port automatisch an die Applikation weitergeleitet werden. HTTP verwendet üblicher-

weise den Port 80 (siehe auch die Datei */etc/services*). Die Regel kann also schon so weit eingeschränkt werden, dass sie lediglich TCP-Pakete an Port 80 untersucht.

```
alert tcp any any <> any 80 (msg: "Webzugriff Port 80");
```

Nun werden während eines Webzugriffes viele Pakete ausgetauscht. Interessieren uns nur die Aufbauten, so können wir ein weiteres Attribut definieren, welches die TCP-Flags testet. TCP-Verbindungsaufbauten sind dadurch gekennzeichnet, dass nur das SYN-Bit der TCP-Flags gesetzt ist. Snort bietet diesen Test mit dem Attribut *flags*. Die Regel sieht dann folgendermaßen aus:

```
alert tcp any any -> any 80 (msg: "Webzugriff Port 80"; flags:S;)
```

Bei der gerade entwickelten Regel kann dann auch die Richtung eindeutig angegeben werden. Das SYN-Paket wird immer an Port 80 gesendet.

Diese Regel erlaubt nun die Verfolgung sämtlicher Webzugriffe im Netzwerk. Dies erscheint vielleicht zunächst nur akademischen Nutzen zu haben, da ja bereits die Webserver alle Zugriffe protokollieren. Jedoch kann die Regel leicht abgeändert werden, so dass ein tatsächlicher Nutzen zu erkennen ist. Möchten Sie zum Beispiel nicht autorisierte Webserver in Ihrem Netzwerk aufspüren, so modifizieren Sie die Regel so, dass Zugriffe auf bekannte Webserver nicht protokolliert werden. Dazu definieren Sie der Einfachheit halber eine Variable `WEBSERVER`, in der alle in Ihrem Netz bekannten Webserver aufgeführt werden. Des Weiteren benötigen Sie eine Variable `HOME_NET`, die Ihr Netzwerk definiert. Anschließend können Sie folgendermaßen die Regel anpassen:

```
config order: pass alert
WEBSERVER=[192.168.111.50/32,192.168.111.80/32]
HOME_NET=192.168.111.0/24
pass tcp any any -> $WEBSERVER 80
↳ alert tcp any any -> $HOME_NET 80 (msg: "Webzugriff Port 80";
↳ flags:S;)
```

Bedenken Sie, dass die *Passregel* alle zutreffenden Pakete ignoriert. Sie werden von weiteren Regeln nicht untersucht. Dazu muss jedoch Snort mit der Option `-o` gestartet werden oder in der Konfigurationsdatei die Option `order` gesetzt sein.

Diese Option wird nun zunächst die Reihenfolge der Regeln verkehren, so dass die Passregeln als erste Regeln abgearbeitet werden. Anschließend werden alle Pakete an bekannte Webserver ignoriert und lediglich verbleibende HTTP-Anfragen auf unbekannte oder unautorisierte Webserver protokolliert.

Erzeugen Sie einen Regelsatz, welcher nicht autorisierte Telnet-Server aufzeigt.

1. Stellen Sie sicher, dass der Telnet-Server auf Ihrem Rechner aktiviert ist und zur Verfügung steht.
2. Erzeugen Sie zunächst den Regelsatz. Geben Sie als autorisierten Telnet-Server Ihre eigene IP-Adresse an.

3. Überprüfen Sie Ihre IP-Adresse mit dem Befehl `ifconfig eth0`.
4. Geben Sie sich vorübergehend eine weitere IP-Adresse. Wählen Sie hierzu eine nicht belegte IP-Adresse IP2.

```
ifconfig eth0:0 IP2
```

5. Starten Sie anschließend Snort mit Ihrer Konfigurationsdatei und testen Sie die Telnet-Verbindung unter Angabe der ersten und der zweiten IP-Adresse. Betrachten Sie die Ausgaben von Snort.

Data Stream-Zugriff

In diesem Beispiel wird die im letzten Abschnitt entwickelte Regel erweitert, so dass Sie den Zugriff auf Data Streams (<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q188806>) erkennt. Hierzu ist es nun wichtig, nicht die Pakete des Verbindungsaufbaus zu untersuchen, sondern spätere Pakete nach Aufbau der Verbindung. Diese TCP-Pakete zeichnen sich dadurch aus, dass sie mindestens das Acknowledge Bit (ACK) gesetzt haben. Weitere Bits wie Push (PSH) können gesetzt sein. Hierzu wird die Regel unter Verwendung des Attributes `flags` folgendermaßen abgewandelt:

```
alert tcp any any -> any 80 (msg: "Webzugriff Port 80"; flags:A+;)
```

Diese Regel wird nun jedes Paket anzeigen, welches zu einer aufgebauten HTTP-Verbindung gehört und von dem Client kommt.

Wenn Snort eine TCP-Streamreassemblierung mit dem `stream4`-Präprozessor durchführt, kann auch das Attribut `flow` genutzt werden. Dann sieht die Regel folgendermaßen aus:

```
alert tcp any any -> any 80 (msg: "Webzugriff Port 80"; flags:A+;
↳ flow:to_server,established;)
```

Diese Regel soll jedoch nur Anfragen protokollieren, welche in der URI die Zeichenkette `.asp::$data` enthalten. Mit dieser Anfrage gelang es, auf den Microsoft Internet Information Server (IIS) in der Version 1.0 bis 4.0 den Inhalt der ASP-Datei herunterzuladen, anstatt sie auf dem IIS auszuführen. Um dies zu erreichen, kann nun gut das Attribut `content` eingesetzt werden.

```
alert tcp any any -> any 80 (msg: "IIS Data Stream Zugriff"; content:\
".asp\:\:$data"; flags:A+; flow:to_server,established;)
```

Denken Sie daran, dass `\` | in Content-Regeln maskiert werden muss. Diese Regel wird bereits ihre Aufgabe zufriedenstellend erfüllen. Sie kann jedoch noch optimiert werden. Es existiert eine besondere Version des Attributes `Content`: `uricontent`. Dieses Attribut betrachtet nur den URI-Anteil des Paketes. Dadurch wird nicht das komplette Paket durchsucht.

```
alert tcp any any -> any 80 (msg: "IIS Data Stream Zugriff"; uricontent:\
".asp\:\:$data"; nocase; flags:A+; flow:to_server,established;)
```

Um nun vor unterschiedlichen Groß- und Kleinschreibungen gefeit zu sein, wird zusätzlich das Attribut `nocase` definiert.

Im Abschnitt »Dynamische Regeln« auf S. 293 wird diese Regel weiterentwickelt, um als Grundlage einer dynamischen Regel zu dienen.



Tipp:

Snort verfügt bereits über eine Regel, die diesen Angriff (SID: 975) erkennen kann.

```

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
  (msg:"WEB-IIS Alternate Data streams ASP file access
  attempt"; flow:to_server,established; uricontent:".asp|3A 3A
  24|DATA"; nocase; reference:bugtraq,149; reference:cve,
  CVE-1999-0278; reference:nessus,10362;
  reference:url,support.microsoft.com/default.aspx?
  scid=kb\;EN-US\;q188806;
  classtype:web-application-attack; sid:975; rev:11;)
  
```

Auch alle weiteren vorgestellten Angriffe werden von Snort bereits erkannt. Jedoch soll an realen Beispielen die Entwicklung derartiger Regeln vorgeführt werden.

SYN/FIN Scan

Der SYN/FIN Scan wurde in der Vergangenheit häufig eingesetzt, um Firewalls und IDS zu umgehen und zu verwirren. Dies soll im Folgenden kurz erläutert werden.

Das SYN-Bit darf lediglich in den ersten beiden Paketen einer TCP-Kommunikation gesetzt sein und zeigt den Verbindungsaufbau und den Wunsch der Synchronisation der TCP-Sequenznummern an. Das FIN-Bit wird von den Kommunikationspartnern verwendet, um den Wunsch des Abbaus der Verbindung auszudrücken.

Frühe auf Paketfilter basierende Firewall-Implementierungen testen bei eingehenden TCP-Paketen lediglich das Vorhandensein des SYN-Bits. Ist das SYN-Bit als einziges Bit gesetzt, handelt es sich um das erste Paket eines Verbindungsaufbaus. Ist gleichzeitig das ACK-Bit gesetzt, so handelt es sich bereits um die Antwort auf den Verbindungsaufbau. Einige frühe Paketfilterimplementierungen prüfen lediglich ob neben dem SYN-Bit ein zweites Bit gesetzt ist, aber nicht, ob es sich tatsächlich um das ACK-Bit handelt. Sind weitere Bits gesetzt (unter der Annahme, es müsse das ACK-Bit sein), akzeptiert der Paketfilter das Paket. Wie reagiert nun der entsprechende Rechner, der schließlich das Paket erhält? Handelt es sich um ein SYN/ACK-Paket, welches zu einer aufgebauten Verbindung gehört, wird es normal verarbeitet. Handelt es sich um ein SYN/ACK-Paket einer nicht bekannten Verbindung, wird ein TCP-Reset-Paket (RST) an den Absender zurückgeschickt. Handelt es sich um ein SYN/FIN-Paket, so wird das FIN-Bit ignoriert und das Paket als SYN-Paket akzep-

tiert. Die Firewall ordnete das Paket einer aufgebauten Verbindung zu, während der Rechner das Paket als Verbindungsaufbau akzeptierte!

Auch IDS-Systeme waren zu Beginn nicht auf SYN/FIN-Pakete eingestellt. Heutzutage weisen keine Firewallprodukte und kein IDS diese Schwächen mehr auf. Jedoch kann durch eine falsche Konfiguration eine derartige Schwäche unter Umständen durch den Administrator eingeführt werden.

Die Entwicklung dieser Regel ist nun recht einfach. Sie soll TCP-Pakete untersuchen. Hierbei interessieren uns nur Pakete, welche an unser eigenes Netz gerichtet sind. Diese sollen auf die Bits SYN und FIN untersucht werden. In einem »echten« SYN/FIN-Scan handelt es sich hierbei um die beiden einzigen Bits. Jedoch soll unsere Regel auch Pakete erkennen, die zusätzlich weitere Bits gesetzt haben.

```
HOME_NET=192.168.111.0/24
┌ alert tcp any any -> $HOME_NET any (msg: "SYN-FIN Scan
┌ (klassisch)"; flags:SF;)
┌ alert tcp any any -> $HOME_NET any (msg: "SYN-FIN Scan
┌ (modifiziert)"; flags:SF+;)
```

Als Übung erzeugen Sie eine Regel für den Nmap XMAS-Scan. Nmap ist ein sehr mächtiges Open-Source-Werkzeug zur Kartierung von Netzwerken. Es bietet die Möglichkeit, verschiedene Arten von Portscans durchzuführen. Einer dieser Portscans ist der so genannte XMAS Tree-Scan. Dieser Scan erfolgt mit TCP-Paketen, in welchen die Bits FIN, URG und PSH gesetzt sind (siehe `man nmap`).

1. Installieren Sie `nmap`. `nmap` ist möglicherweise Teil Ihrer Linux-Distribution. Ansonsten können Sie `nmap` von <http://www.nmap.org> herunterladen.
2. Erzeugen Sie die Regel.
3. Starten Sie Snort und testen Sie Ihre Regel mit `nmap -sX host`.

Bufferoverflow

In diesem Beispiel soll die Entwicklung einer Regel für einen Bufferoverflow vorgestellt werden. Hierzu wurde eine im April 1998 entdeckte Sicherheitslücke des Washington University IMAP Servers ausgewählt. Es wurde bewusst ein derartig alter Bufferoverflow gewählt, jedoch entspricht die Vorgehensweise exakt der Vorgehensweise bei aktuellen Bufferoverflows.



Exkurs: Was ist ein Bufferoverflow?

Die Möglichkeit eines Bufferoverflows in einer Anwendung resultiert aus einem Programmierfehler. Im Wesentlichen werden zu viele Daten in einen zu kleinen Kasten gezwungen.

In vielen Programmen kommen einzelne Aufgaben wiederkehrend vor. Diese Aufgaben werden dann gerne in einem Unterprogramm realisiert, welches von verschiedenen Stellen des Hauptprogramms aufgerufen werden kann. Damit das Unterprogramm später weiß, wo-

hin es im Hauptprogramm zurückspringen muss, sichert der Prozessor vor dem Aufruf des Unterprogramms den aktuellen Stand des Befehlszeigers (Instruction Pointer, IP) auf dem Stapel (Stack). Der Stapel ist eine dynamische Struktur, auf der ein Programm vorübergehend Daten ablegen kann. Jedes Programm besitzt einen eigenen Stapel. Ein Stapel erlaubt lediglich das Lesen und Schreiben der Daten auf dem höchsten Punkt. Dieser wird mit dem Stapelzeiger (Stackpointer) referenziert (siehe Abbildung 9.15). Benötigt nun das Unterprogramm vorübergehend Speicherplatz für eine Eingabe des Benutzers, so fordert es diesen Puffer (Buffer) auf dem Stapel an. Der Stapelzeiger wird um die entsprechende Anzahl Bytes verschoben und als Referenz an das Unterprogramm zurückgegeben (siehe Abbildung 9.16). Das bedeutet, das Unterprogramm kann nun Daten auf dem Stapel beginnend bei dem Stapelzeiger in rückwärtiger Richtung ablegen.

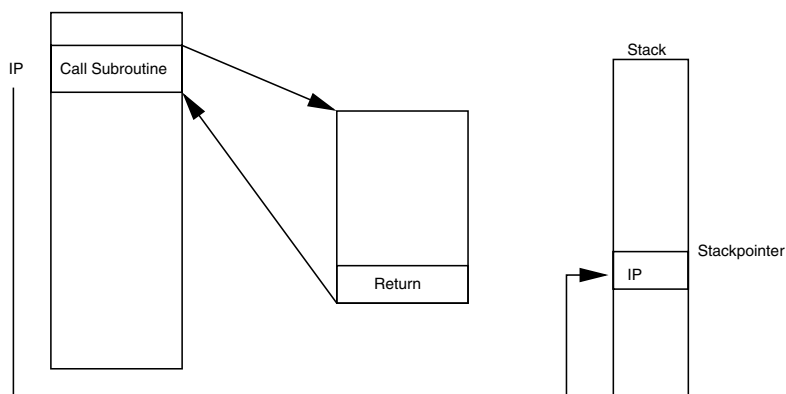


Abbildung 9.15 Funktionsweise eines Bufferoverflow

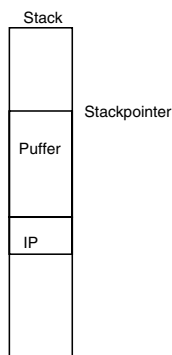


Abbildung 9.16 Pufferreservierung auf dem Stapel

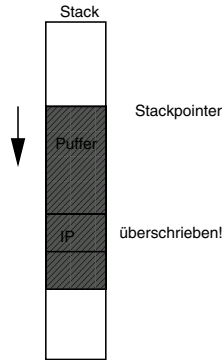


Abbildung 9.17 Überlaufen des Puffers

Stellen wir uns vor, das Programm erwartet die Eingabe des Geburtsdatums des Benutzers. So genügen sicherlich 25 Bytes, um diese Eingabe entgegenzunehmen. Diese 25 Bytes werden nun auf dem Stapel reserviert. Aus irgendeinem Grund (Unwissenheit, Börsartigkeit) gibt der Benutzer jedoch 100 Zeichen ein. Überprüft der Programmierer vor der Kopie der Zeichenkette in den dynamisch allozierten Datenbereich auf dem Stapel ihre Länge nicht, so werden alle 100 Bytes auf den Stapel kopiert. Dadurch werden auch Bereiche überschrieben, die andere gültige Daten enthielten. Es kommt zu einem Überlaufen (Overflow) des originalen Puffers (siehe Abbildung 9.17). Hierbei können auch zum Beispiel Rücksprungadressen der Unterprogramme überschrieben werden.

Es gibt einige Programmiersprachen, die dem Programmierer diese Arbeit (das so genannte Boundary Checking) abnehmen, jedoch gehört die Programmiersprache C nicht dazu. In C werden die meisten Anwendungen für UNIX- und auch Windows-Betriebssysteme programmiert.

Handelt es sich um willkürliche Daten, so stürzt das Programm ab und es kommt zum Segmentation Fault (Linux) oder zu einer Allgemeinen Schutzverletzung (General Protection Fault, Windows). Der Prozessor erkennt, dass es sich um eine unerlaubte Rücksprungadresse handelt. Ein Zugriff auf den Speicher an der Zieladresse ist dem Programm nicht erlaubt.

Unter Umständen besteht jedoch auch die Möglichkeit, den Ort einer Rücksprungadresse auf dem Stack vorherzusagen und so zu überschreiben, dass ein gezielter Rücksprung auf Code, der sich im Vorfeld in den eingegebenen Daten befand, erfolgt. Dann wird in dem Benutzerkontext des missbrauchten Programms der gerade eingegebene Code ausgeführt. Üblicherweise handelt es sich hierbei um den Aufruf einer UNIX-Shell. Daher bezeichnet man diesen von dem Einbrecher verwendeten Code auch häufig als Shellcode. Dies ist unter UNIX besonders brisant, da viele Netzdienste über *root*-Pri-

privilegien verfügen und Eingaben aus ungewisser Quelle entgegennehmen. Weisen diese Dienste derartige Mängel auf, so können sie ausgenutzt werden, um *root*-Privilegien auf dem entsprechenden Rechner zu erlangen. In diesem Fall spricht man von Root Exploits.

Diese Art des Bufferoverflows ist recht leicht an dem so genannten NOP Sled zu erkennen. Ein gezielter Rücksprung, wie gerade beschrieben, ist meist nicht möglich. Der Angreifer kann nicht genau den Zustand des Stacks vorhersagen. Daher kann er auch nicht die Rücksprungadresse berechnen. Nun versieht er seinen Code zu Beginn mit bis zu mehreren hundert NOP-Befehlen. Ein NOP ist ein No-Operation-Befehl. Dieser Befehl hat, wenn er von dem Prozessor ausgeführt wird, keine Funktion außer den Instructionpointer weiterzubewegen. Ist der Code für den Bufferoverflow derartig angepasst, so muss die Rücksprungadresse nur noch ungefähr in den Bereich der NOP-Befehle zeigen. Dies bezeichnet man als NOP-Schlitten (Sled), da der Befehlszeiger wie auf einem Schlitten über die NOPs zum Bufferoverflow rutscht und schließlich diesen Code ausführt.

Bufferoverflows sind nicht die einzigen Sicherheitslücken, welche auf Programmierfehler zurückzuführen sind. So genannte Format-String-Fehler sind ähnlich tückisch und erlauben Angriffe ähnlicher Wirksamkeit. Im Anhang finden Sie weitere Informationen zu Bufferoverflows, Format String Errors und dem Schutz vor der Ausnutzung dieser Sicherheitsmängel. └─

Erzeugung der Regeln

Um nun eine Regel erzeugen zu können, die diesen Bufferoverflow erkennen kann, benötigen wir zunächst ein wenig mehr Informationen über den Bufferoverflow selbst. Eine recht umfangreiche Datenbank, in der viele relevante Sicherheitslücken archiviert werden, ist die Bugtraq-Datenbank auf <http://www.securityfocus.com>. Der angesprochene IMAP-Bufferoverflow wird dort unter <http://online.securityfocus.com/bid/130/> besprochen.

Der Washington University IMAP-Server ist ein Netzwerkdienst, welcher einen Zugriff mit dem IMAP-Protokoll (Internet Message Access Protocol) auf E-Mail-Konten erlaubt. Dieser Dienst besitzt üblicherweise *root*-Privilegien, wenn er unter UNIX gestartet wird.

Um nun eine Regel zu entwickeln, welche diesen Angriff erkennen kann, benötigen wir zunächst mehr Informationen über die beteiligten Pakete. Momentan ist bekannt:

1. Das IMAP-Applikationsprotokoll nutzt das TCP-Protokoll zur Übertragung der Daten.
2. Der IMAP-Server bindet sich auf den Port 143 (*/etc/services*).

3. Da es sich um einen Bufferoverflow der Anwendung handelt, muss bereits eine TCP-Verbindung aufgebaut sein. Es kann sich nicht um eine Eigenschaft der SYN-Pakete handeln.

Diese Informationen genügen jedoch noch nicht. Würden alle Pakete, die diesen Anforderungen genügen, protokolliert werden, so würde jede korrekte Nutzung des IMAP-Servers zu einer Alarmierung führen. Weitere Informationen über den Inhalt des kritischen Paketes werden benötigt. Hierzu eignet sich am besten ein Blick in den Code, welcher zur Ausnutzung dieser Sicherheitslücke geschrieben wurde. Ein weiteres sehr gutes Archiv derartiger Programme befindet sich in dem Packetstorm-Archiv unter <http://www.packetstormsecurity.org>. Hier wird unter <http://packetstormsecurity.nl/new-exploits/ADM-imap4r1-linux.c> das originale Programm vorgehalten. Dieses Programm enthält Shellcode, der während des Angriffes übertragen wird und den Bufferoverflow auslöst.

Das folgende Listing enthält einen Auszug:

```

/*

THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF THE ADM CREW

Linux WU-IMAPD 4.1 remote root exploit (4/98)
by ndubee||plaguez
   dube0866@eurobretagne.fr

Usage:  ./imapx <offset>
where offset = -500..500 (brute force if 0 doesnt work)

Credits:
- Cheez Whiz (original x86 BSD exploit)
- #!w00w00 and #!ADM

Note:
if you plan to port this to other OS., make sure the
shellcode doesn't contain lower case chars since imapd
will toupper() the shellcode, thus fucking it up.

*/

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

#define BUFLen 2048
#define NOP 0x90

```

```

char shell[] =
/*
    jmp 56
    popl %esi
    movl  %esi,%ebx
    movl  %ebx,%eax

    addb  $0x20,0x1(%esi)
    addb  $0x20,0x2(%esi)
    addb  $0x20,0x3(%esi)
    addb  $0x20,0x5(%esi)
    addb  $0x20,0x6(%esi)

... gekürzt ...

"\x31\xc0\xaa\x89\xf9\x89\xf0\xab"
"\x89\xfa\x31\xc0\xab\xb0\x08\x04"
"\x03\xcd\x80\x31\xdb\x89\xd8\x40"
"\xcd\x80\xe8\xc3\xff\xff\xff\x2f"
"\x42\x49\x4e\x2f\x53\x48\x00";

void
main (int argc, char *argv[])
{
    char buf[BUFLEN];
    int offset=0,nop,i;
    unsigned long esp;

    fprintf(stderr,"usage: %s <offset>\n", argv[0]);

    nop = 403;
    esp = 0xbffff520;
    if(argc>1)
        offset = atoi(argv[1]);

    memset(buf, NOP, BUFLen);
    memcpy(buf+(long)nop, shell, strlen(shell));
    for (i = 512; i < BUFLen - 4; i += 4)
        *((int *) &buf[i]) = esp + (long) offset;

    printf("* AUTHENTICATE {%d}\r\n", BUFLen);
    for (i = 0; i < BUFLen; i++)
        putchar(buf[i]);

```

```
printf("\r\n");

return;
```

Aus diesem Shellcode können nun eine gewisse Anzahl von Bytes gewählt werden, die im Inhalt des Paketes überprüft werden sollen. Hierbei sollte darauf geachtet werden, dass es sich nicht um zu wenige Bytes handelt, da es sonst zu falsch-positiven Meldungen kommen könnte. In diesem Beispiel sollen die Bytes `\x03\xcd\x80\x31\xdb\x89\xd8\x40` verwendet werden. Die Regel soll also jedes in Frage kommende Paket nach dieser Byte-Sequenz durchsuchen.

```
alert tcp any any -> $HOME_NET 143 (msg:"IMAP Bufferoverflow";flags: A+; \
  content:"|03cd 8031 db89 d840|";reference:bugtraq,130; \
  flow:established,to_server; reference:cve,CVE-1999-0005; \
  classtype:attempted-admin; sid:1000003; rev:1;)
```



Achtung:

Dies ist keine besonders gute Regel. Sie erkennt zwar den analysierten Angriff, aber wenn der Angreifer in der Lage ist, den verwendeten Code leicht zu modifizieren, wird diese Regel den Angriff nicht mehr melden. Sinnvoller ist es, die Regel an die Sicherheitslücke anzupassen. In diesem Fall handelt es sich um einen Bufferoverflow des IMAP-Kommandos AUTHENTICATE. Jeder derartige Bufferoverflow wird also den Befehl AUTHENTICATE übertragen und im Anschluss mehr Informationen senden, als der Puffer des IMAP-Servers aufnehmen kann. Dies kann mit dem Attribut `isdataat` geprüft werden:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 143 (msg:"IMAP
  ↳ authenticate overflow attempt"; flow:established,to_server;
  ↳ content:"AUTHENTICATE"; nocase; isdataat:100,relative;)
```

Hier prüft das Attribut `isdataat`, ob sich 100 Bytes hinter der AUTHENTICATE-Zeichenkette im Paket noch Daten befinden.

Da das auch normal sein kann, müssen wir zusätzlich prüfen, ob diese 100 Byte Daten tatsächlich zum AUTHENTICATE-Kommando gehören. IMAP erkennt das Ende eines Kommandos am Newline (`\n`). Daher ist es sinnvoll, anschließend zu prüfen, ob sich in den weiteren 100 Bytes ein Newline befindet, und den Alarm nur auszulösen, wenn dies nicht der Fall ist. Dann strebt der Angreifer unter Umständen einen Bufferoverflow an.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 143 (msg:"IMAP
  ↳ authenticate overflow attempt"; flow:established,to_server;
  ↳ content:"AUTHENTICATE"; nocase; isdataat:100,relative;
  ↳ pcre:"/\sAUTHENTICATE\s [^\n]{100}/smi";reference:cve,
  ↳ 1999-0042; reference:nessus,10292; classtype:misc-attack;
```

```
↳ sid:1844; rev:9;)
```

Das `pcpre`-Attribut sucht hier nach der Zeichenkette `AUTHENTICATE`, gefolgt von 100 Zeichen, die kein Newline enthalten.

9.2.9 Fortgeschrittene Regeln

In diesem Kapitel sollen fortgeschrittene Regeln vorgestellt werden. Hierbei werden zunächst eigene Aktionen für Regeln definiert. Anschließend werden dynamische Regeln vorgestellt. Hieran wird die flexible Antwort aufgezeigt und an Beispielen implementiert. Des Weiteren wird die Konfiguration der Präprozessoren besprochen.

Regel-Aktionen

Das erste Schlüsselwort im Regelrumpf definiert die Aktion, die ausgeführt wird, wenn die Regel zutrifft. Snort bietet hier eine Auswahl von fünf vordefinierten Aktionen. Hierbei handelt es sich um die bereits in Abschnitt »Erzeugung der Regeln« auf S. 261 erwähnten Aktionen: `alert`, `log`, `pass`, `activate` und `dynamic`. Zur Protokollierung werden lediglich die Aktionen `alert` und `log` zur Verfügung gestellt. Die Protokollierung erfolgt dann entsprechend der Konfiguration der Output-Plug-Ins (siehe Abschnitt »Fortgeschrittene Protokollierung« auf S. 326).

Diese zentrale Definition der zu verwendenden Output-Plug-Ins ist in einigen Fällen zu starr und unflexibel. Wünschenswert wäre bei unterschiedlichen Ereignissen, die Alarmierung in unterschiedlichen Dateien vorzunehmen. Individuelle Alarmmeldungen können so an bestimmte Benutzer versandt werden. Damit besteht die Möglichkeit, dass Webadministratoren spezifisch über Angriffe auf dem Webserver informiert werden.

Snort bietet hierzu die Möglichkeit, eigene Regeltypen (Ruletype) zu definieren. Die allgemeine Syntax dieser Regeltypen ist:

```
ruletype NAME
{
    type TYP
    output PLUGIN
}
```

Ein mögliches Beispiel für eine derartige Regel ist im Folgenden angegeben. Dabei wird der Regeltyp `AlarmStufeRot` definiert. Dieser Regeltyp führt eine `alert`-ähnliche Aktion aus. Dazu verwendet er drei verschiedene Output-Plug-Ins mit den entsprechenden Optionen. Für eine genauere Betrachtung der Output-Plug-Ins und ihrer Optionen sehen Sie bitte unter Abschnitt »Fortgeschrittene Protokollierung« auf S. 326 nach. Spätere Regeln können dann diesen Typ verwenden, wie im Beispiel angegeben.

```

ruletype AlarmStufeRot
{
    type alert
    output alert_syslog: LOG_AUTH LOG_CRIT LOG_CONS
    output alert_unified: /var/log/alert/unified.log
    output csv: /var/log/alert/csv.log timestamp,msg
}

```

```

AlarmStufeRot tcp any any -> any 80 (msg: "IIS Data Stream Zugriff"; \
    uricontent:".asp\:\:$data"; nocase; flags:A+;\
    flow:established,to_server;)

```

Mithilfe des Schlüsselwortes `ruletype` kann für diesen Typ ein Name vergeben werden. Mit diesem Namen kann später auf den Regeltyp zugegriffen werden. Die Option `type` übt in der Definition eine wichtige Aufgabe aus. Sie definiert, was passiert, wenn dieser Regeltyp verwendet wird. So kann eine eigene dynamische (`type dynamic`), aktivierende (`type activate`) oder alarmierende (`type activate`) Regel definiert werden.

Dynamische Regeln

Snort bietet die Möglichkeit dynamischer Regeln. Dazu hat es zwei verschiedene Arten von Regeln: aktivierende und dynamische Regeln.

Die `activate`-Regel enthält ein zwingend erforderliches Attribut `activates`: Nummer. Die angegebene Nummer verknüpft die `activate`-Regel mit der entsprechenden `dynamic`-Regel.

Die `dynamic`-Regel enthält ein analoges Attribut `activated_by`: Nummer. Die Regel wird durch alle `activate`-Regeln mit identischer Nummer aktiviert. Die Dauer der Aktivierung wird mit dem Attribut `count`: Anzahl festgelegt.

Die Verwendung der Optionen wird am einfachsten an einem Beispiel deutlich:

```

activate tcp any any -> any 80 (msg: "IIS Data Stream Zugriff"; \
    uricontent:".asp\:\:$data"; nocase; flags:A+;\
    flow:established,to_server; activates: 1;)
dynamic tcp any 80 -> any any (msg: "IIS Data Stream Pakete"; \
    flow:established,to_server; flags: A+; logto: "datastream.log" \
    activated_by: 1; count: 100; )

```

Die erste Regel ist bereits bekannt. Sie prüft, ob ein Zugriff auf einen ASP Data Stream erfolgen soll. Dies wird dann alle Regeln aktivieren, welche ebenfalls die Nummer 1 verwenden. Dadurch wird die zweite Regel angeschaltet. Diese zweite Regel wird anschließend alle weiteren 100 Pakete von einem Port 80 in der Datei `datastream.log` protokollieren. Wahrscheinlich wird sich die Antwort des Webserver innerhalb dieser 100 Pakete befinden. So kann durch eine spätere Analyse dieser Datei ermittelt werden, wie der Webserver auf diesen Angriff reagiert.

**Achtung:**

Die dynamischen Regeln werden vom Snort-Team nicht mehr weiterentwickelt. Die Funktionen können jetzt bereits komplett mit den Attributen `tag` und `flowbits` nachgestellt werden. Zukünftige Snort-Versionen werden insbesondere im Tagging-Bereich Verbesserungen erfahren.

Flexible Antwort

Die flexible Antwort erlaubt Snort die aktive Reaktion auf bösartige Pakete und unerwünschte HTTP-Anfragen (siehe unten). Hierbei werden nicht sämtliche Pakete von einem bestimmten Rechner unterbunden, sondern nur spezifisch die Verbindungen beendet, die die Verletzung der Richtlinien verantworten. Es kann also kaum zu einem Denial of Service durch gefälschte Absenderadressen kommen. Andere Werkzeuge führen häufig eine Reaktion durch, die von dem Angreifer ausgenutzt werden kann. Hierbei blockiert das Werkzeug alle weiteren Pakete des Angreifers. Fälscht der Angreifer im Folgenden die Absenderadresse seiner Pakete, so dass diese scheinbar von DNS-Servern oder vom E-Mail-Server des wichtigsten Kunden kommen, so werden diese Rechner danach komplett gesperrt. So kann ein Angreifer einen Denial of Service erzeugen. Snort beendet jedoch spezifisch nur die Verbindung, die die Richtlinienverletzung erzeugt. Weitere Verbindungen des gleichen Rechners bleiben unbenommen.

**Achtung:**

Dies ist keine echte Intrusion Prevention, da es sich lediglich um eine Reaktion auf die Regelverletzung handelt. Der Transport des Paketes wird nicht proaktiv verhindert. Das den Alarm auslösende Paket kommt immer noch beim Zielrechner an. In schnellen Netzwerken erreichen möglicherweise auch noch weitere Pakete des Angreifers den Zielrechner, bevor Snort reagieren und mit Fehlerpaketen die Verbindung abbrechen kann!

Dies ermöglicht, dass Snort nicht nur passiv die Pakete protokollieren und einen Alarm auslösen kann, sondern in der Lage ist, die Verbindung aktiv zu beenden. Dazu kann Snort Fehlermeldungen an einen oder beide Kommunikationspartner senden, welche diesen suggerieren, dass die Verbindung von der jeweils anderen Seite unterbrochen wurde. So besteht die Möglichkeit, Angriffe direkt während ihrer Durchführung zu vereiteln.

Führt ein Angreifer zum Beispiel einen Angriff mit dem oben besprochenen IMAP-Bufferoverflow durch, so verfolgt er das Ziel, im Kontext des IMAP-E-Mail-Servers

eine Bourne Shell zu starten, welche dann von ihm über die bestehende Netzwerkverbindung mit weiteren Befehlen gefüttert werden kann. Das bedeutet, dass der eigentliche Angriff erst anschließend erfolgt, da entsprechend von dem Angreifer die Befehle über die bestehende Netzwerkverbindung eingegeben werden.



Achtung:

Das ist nur bei Bufferoverflows so üblich. Andere Angriffe übertragen die auszuführenden Befehle bereits im ersten Paket. Ein berühmtes Beispiel ist der SQL-Slammer-Wurm, bei dem ein einziges UDP-Paket für die erfolgreiche Ausführung des Angriffs ausreichte.

Bricht Snort nun direkt nach der Erkennung des Bufferoverflow-Paketes diese Verbindung ab, so ist der Angreifer nicht mehr in der Lage, die eigentlichen Befehle mithilfe der gerade erfolgreich gestarteten Shell auszuführen.

Vorraussetzung für diesen Abbruch ist jedoch eine Fehlermeldung, die Snort an einen oder beide Kommunikationspartner sendet. Dies erfolgt mit dem Attribut `resp`. Hierbei stehen in Snort folgende Möglichkeiten zur Verfügung:

- TCP Reset-Pakete an den Absender (`rst_snd`), an den Empfänger (`rst_rcv`) oder beide Kommunikationspartner (`rst_all`)
- ICMP-Fehlermeldungen an den Absender
 - `icmp_net`. ICMP Network Unreachable.
 - `icmp_host`. ICMP Host Unreachable.
 - `icmp_port`. ICMP Port Unreachable.
 - `icmp_all`. Alle drei Fehlermeldungen werden an den Absender gesendet.

Diese Optionen können auch miteinander gemischt werden.

Nun besteht die Möglichkeit, beim Auffinden gefährlicher Pakete in aufgebauten Verbindungen die entsprechenden Verbindungen zu beenden. Hierbei eignen sich die TCP Reset-Pakete für TCP-Verbindungen und die ICMP-Pakete für alle Arten von Verbindungen. Jedoch ist bei einer TCP-Verbindung ein TCP Reset erfolgreicher. Eine ICMP-Fehlermeldung zeigt eine (vorübergehende) Netzwerkstörung an. Ein TCP Reset setzt die entsprechende Verbindung immer zurück.

Wenn diese Funktionen nun zum Schutz vor dem besprochenen Bufferoverflow eingesetzt werden sollen, so kann die entsprechende Regel wie folgt abgeändert werden:

```
alert tcp any any -> $HOME_NET 143 (msg:"IMAP Bufferoverflow";flags: A+; \
  flow: to_server,established; content:"|03cd 8031 db89 d840|"; \
```

```
resp: rst_all; reference:bugtraq,130; reference:cve,CVE-1999-0005;\nclasstype:attempted-admin; sid:1000003; rev:1;)
```

Snort wird bei einem Einbruchversuch mit diesem IMAP-Bufferoverflow an beide Kommunikationspartner ein TCP Reset senden. Es ist sinnvoll, das TCP Reset an beide Partner zu senden, da unter Umständen der Angreifer seinen Rechner so modifiziert hat, dass er nicht auf TCP Reset-Pakete reagiert. Werden die Pakete in beide Richtungen gesendet, so sollte mindestens der angegriffene Rechner die Verbindung schließen und keine weiteren Pakete für diese Verbindung annehmen.

Die flexible Antwort bietet auch die Möglichkeit, gezielt HTTP-Anfragen und die Antworten der Webserver nach Schlüsselwörtern zu durchsuchen und dem Benutzer Warnmeldungen zu senden bzw. den Zugriff auf die Webseite zu blockieren. Dies erfolgt mit dem Attribut `react`. Damit kann Snort bei HTTP-Verbindungen eine erklärende Webseite an den Benutzer senden. Dieses Attribut besitzt die folgenden Optionen:

- `block`. Schließe die Verbindung und sende eine HTML/JavaScript-Seite an den Benutzer.
- `warn`. Sende eine Warnung, aber schließe die Verbindung nicht!

Diese beiden Optionen können mit zwei weiteren Optionen in ihrem Verhalten modifiziert werden:

- `msg`. Zeige den Inhalt des Attributs `msg`: in der Webseite mit an.
- `proxy`. Verwende für die Antwort den angegebenen Proxy-Port. Diese Option ist erforderlich, wenn der Zugriff auf Webseiten über einen Proxy erfolgt.



Achtung:

Zum Zeitpunkt der Verfassung dieser Zeilen waren die Optionen `warn` und `proxy` noch nicht implementiert, aber bereits vorgesehen.

Das Attribut `react` wird häufig mit dem Attribut `content-list` eingesetzt, um den Zugriff auf Webseiten, die spezifische Schlüsselwörter enthalten, zu unterbinden. Im Folgenden soll ein Beispiel dargestellt werden.

**Achtung:**

Das Attribut `content-list` ist zurzeit fehlerhaft und sollte in den aktuellen Versionen Snort 2.1.x nicht eingesetzt werden. Inwieweit dieses Attribut in zukünftigen Versionen wieder unterstützt wird, lesen Sie bitte in der dazugehörigen Dokumentation nach.

Der Zugriff auf Webseiten, die in Zusammenhang mit Raubkopien stehen, soll unterbunden werden. Dazu wird zunächst eine Datei mit Schlüsselwörtern erzeugt:

```
# Inhalt der Datei warez.keywords
"warez"
"Raubkopie"
"key generator"
```

Anschließend wird eine Regel definiert, die diese Datei zur Untersuchung der Pakete nutzt.

```
alert tcp any any <> any 80 (msg: "Mögliches Raubkopie Archiv"; flags: A+; \
  content-list: "warez.keywords"; react: block, msg;)
```

Greift nun ein Benutzer auf eine Webseite zu, welche eines der angegebenen Schlüsselwörter enthält, wird Snort den Zugriff abbrechen und dem Benutzer eine Webseite senden, auf der dieses Verhalten erklärt wird.

Leider ist der Inhalt dieser Webseite in den aktuellen Versionen noch im Quelltext des Plug-Ins `react` kodiert. Das bedeutet, dass eine Modifikation dieser Seite eine anschließende Übersetzung des Quelltextes erfordert.

**Tipp:**

Wenn Sie diese Funktion wünschen, so sollten Sie sich die entsprechende Stelle in der Quelltextdatei `sp_react.c` ansehen. Dort finden Sie eine Funktion `ParseReact`, welche auf den ersten Zeilen die Definition der Webseite in Inline-HTML enthält. Modifizieren Sie diese, soweit es für Sie nötig ist. Wichtig ist sicherlich die Anpassung der dort eingetragenen E-Mail-Adresse und möglicherweise eine Übersetzung in die deutsche Sprache. Anschließend übersetzen Sie dann Snort neu und installieren das entsprechende Plug-In `react`.

Präprozessoren

Snort selbst ist ein sehr einfacher Paketanalysator und kann lediglich ein einziges Paket gleichzeitig untersuchen. Häufig erfordert der Einsatz als NIDS jedoch die gleichzeitige Untersuchung mehrerer Pakete, zum Beispiel beim Vorliegen fragmentierter Pakete.

Snort verwendet Präprozessoren, welche die ankommenden Pakete untersuchen und defragmentieren, reassemblieren und dekodieren können. Des Weiteren können Sie Portscans durch mehrere Pakete erkennen und Netzwerkanomalien untersuchen.

frag2

frag2 ist der aktuelle Präprozessor (seit Snort 1.8) zur Defragmentierung von Paketen. Dieser Präprozessor ist in der Lage, Fragmente zu erkennen, zu sammeln und das komplette defragmentierte Paket an die Detektionsmaschine von Snort weiterzuleiten. *frag2* stellt dabei eine komplett überarbeitete Defragmentierungsroutine zur Verfügung. Das ältere Plug-In *defrag* wies einige Nachteile auf, die in *frag2* entfernt wurden.

Exkurs: Fragmentierung



Fragmentierung ist ein ganz natürlicher Prozess in Computer-Netzwerken. Computer-Netzwerke besitzen in Abhängigkeit des eingesetzten Mediums (Ethernet, Token Ring etc.) eine maximale Größe der Übertragungseinheit (Maximum-Transmission-Unit, MTU). Diese beträgt bei Ethernet typischerweise 1.500 Bytes. Verbindet ein Router zwei Netze mit unterschiedlichen MTUs, zum Beispiel 4.464 Bytes und 1.500 Bytes, so kann ein Paket, welches mit einer Größe von 4.000 Bytes aus dem ersten Netzwerk ankommt, nicht in das zweite Netzwerk weitergesendet werden, da die MTU eine maximale Paketgröße von 1.500 Bytes vorschreibt. Es existieren nun zwei verschiedene Möglichkeiten:

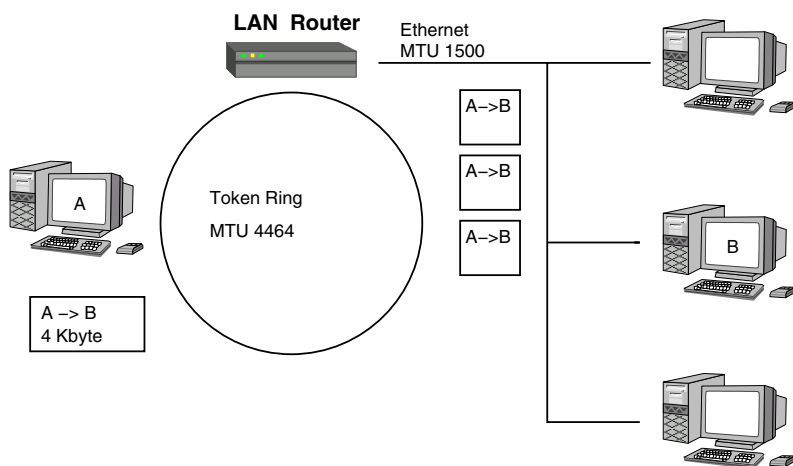


Abbildung 9.18 Normale Fragmentierung

1. Der Absender hat zusätzlich im Paket vermerkt, dass dieses nicht fragmentiert werden darf (DF-Bit, Do not Fragment). Dann verwirft der Router das Paket und sendet eine Fehlermeldung an

den Absender (ICMP-Destination Unreachable, Fragmentation needed). Der Absender muss anschließend das Paket erneut in kleinerer Größe senden. Dieses Verfahren wird auch bei der Path MTU Discovery eingesetzt. Hierbei ermittelt der sendende Rechner zunächst die MTU der gesamten Verbindung und sendet anschließend nur noch mit dieser MTU.

2. Der Router darf das Paket fragmentieren. Er schneidet das Paket bei 1.500 Bytes ab und erzeugt weitere Paket mit identischem IP-Header. An diese Pakete hängt er die restlichen Daten an. Als Hinweis der Fragmentierung setzt er im ersten Paket das MF-Bit (More Fragments) und initialisiert den Fragment Offset mit 0. Im letzten Paket wird das MF-Bit gelöscht. Der Offset des zweiten Paketes wird entsprechend auf 1.480 (1.500-20 (IP-Header)) gesetzt. Genauso wird der Offset des dritten Paketes berechnet. Der empfangende Rechner wird alle Fragmente sammeln, zusammensetzen und das fertige Paket auswerten.

Diese Fragmente können nun zur Umgehung von NIDS eingesetzt werden. Stellen Sie sich vor, der Angreifer ist in der Lage, das Paket für den IMAP-Bufferoverflow so zu produzieren, dass der erste Teil des Shellcodes sich in dem ersten Fragment befindet und der zweite Teil in dem zweiten Fragment. Wenn Snort nun nicht in der Lage ist, die Fragmente zusammenzusetzen, bevor die Pakete inspiziert werden, wird es diesen Angriff nicht erkennen, da die Signatur über zwei Pakete verteilt ist. Der Angriff wird jedoch weiterhin erfolgreich sein, da der Rechner, auf dem der IMAP-Server läuft, zunächst alle Fragmente sammelt und anschließend defragmentiert. Das fertige Paket kann dann den Bufferoverflow im IMAP-Server verursachen.

Bei der Anwendung dieses Präprozessors sind einige Überlegungen erforderlich, denn die Defragmentierung ist nicht ganz unproblematisch. Für diesen Vorgang benötigt der Präprozessor Arbeitsspeicher. Ist einem Angreifer diese Funktion bekannt, so kann er Hunderte von Fragmenten senden, welche scheinbar alle zu unterschiedlichen IP-Paketen gehören. Der Präprozessor wird hierfür dann den Arbeitsspeicher reservieren. So kann von einem Angreifer der gesamte Arbeitsspeicher des analysierenden Rechners aufgebraucht werden. Dies ist noch wirksamer, wenn der Angreifer nie sämtliche Fragmente der entsprechenden Pakete sendet, da Snort den Arbeitsspeicher dann nicht wieder freigeben kann. Dies war ein großes Problem beim originalen Defragmentierungs-Plug-In *defrag_frag2* verwendet aus diesem Grund Grenzwerte. Diese Werte können jedoch beim Aufruf des Präprozessors modifiziert werden:

```
preprocessor frag2: memcap 8000000, timeout 120
```

Als Arbeitsspeicher müssen mindestens 16 Kbyte zur Verfügung gestellt werden, ansonsten verwendet *frag2* seinen Standardwert von 4 Mbyte.

Die Parameter von *frag2* sind im Einzelnen:

- `timeout`. Wartezeit für alle Fragmente eines Paketes in Sekunden (Default: 60).
- `memcap`. Der *frag2* zur Verfügung stehende Arbeitsspeicher in Bytes (Default: 4 Mbyte).
- `detect_state_problems`. Alarmiert bei überlappenden Fragmenten.
- `min_ttl`. Fragmente mit einer kleineren TTL werden von *frag2* nicht berücksichtigt (Default: 0).
- `tll_limit`. Definiert die maximale Abweichung des TTL-Wertes der Fragmente in einem Paket (Default: 5).

Stream4

Der *stream4*-Präprozessor bietet TCP-Streamassemblierung und zustandsabhängige (stateful) Analysen. Diese Fähigkeit beschränkt sich jedoch auf TCP-Pakete. Damit besteht die Möglichkeit, die Anzahl der falsch-positiven Meldungen stark zu reduzieren. *Stream4* besteht aus zwei Präprozessoren: *stream4* und *stream4_reassemble*.

Um zu verstehen, welche Aufgabe der *stream4*-Präprozessor hat, sollen kurz die Werkzeuge *stick*⁷ und *snot*⁸ erwähnt werden. Diese Werkzeuge sind in der Lage, auf der Basis der Snortregeln Pakete zu erzeugen, die eine Alarmierung auslösen. Dies führt üblicherweise zu einer Überlastung des NIDS. Der *Stream4*-Präprozessor versucht nun, bei TCP-Paketen zu überprüfen, ob diese Pakete gültig sind. Hierfür pflegt der Präprozessor eine Zustandstabelle, in der jede gültige TCP-Verbindung eingetragen wird. Für diese Gültigkeit verlangt Snort einen erfolgreichen TCP-Handshake. Anschließend kann Snort überprüfen, ob Pakete zu derartigen aufgebauten Verbindungen gehören und alle weiteren Pakete verwerfen (Option `config stateful:est`). Pakete, die nicht Teil einer aufgebauten Verbindung sind, werden auch vom Empfänger nicht beachtet und verworfen. Die Detektionsmaschine braucht daher ebenfalls diese Pakete nicht zu testen.

Die Bedeutung des *stream4_reassemble*-Präprozessor soll kurz am Beispiel einer Telnet-Session vorgestellt werden. Hierbei wird jeder Buchstabe, der vom Benutzer in den Telnet-Client eingegeben wird, in einem einzelnen Paket vom Client zum Telnet-Server übertragen. Der Telnet-Server nimmt diesen Buchstaben entgegen und bestätigt den Empfang. Zusätzlich sendet er den Buchstaben im Falle der Telnet-Verbindung auch wieder an den Client, damit der Buchstabe auf dem Bildschirm des Telnet-Clients dargestellt wird.

Snort sieht also jeden Buchstaben einzeln und nicht im Kontext. Es kann daher nicht die Eingaben des Benutzers auf bestimmte unerlaubte Befehle (z.B. `su - root`) untersuchen. Wenn Snort den gesamten TCP-Datenstrom betrachtet, kann es doch diese Zeichenkette entdecken. Dies ermöglicht dieser Präprozessor. Im Falle von Telnet ist jedoch noch ein weiterer Präprozessor (*telnet_decode*) erforderlich, da in jedem Paket

⁷ <http://www.eurocompton.net/stick/projects8.html>

⁸ <http://www.stolenshoes.net/sniph/index.html>

nicht nur der einzelne Buchstabe, sondern auch noch Befehle des Telnet-Protokolls übertragen werden.

Ein weiterer häufiger Versuch, NIDS-Systeme zu umgehen, wird ebenfalls vom *stream4*-Präprozessor erkannt. Um beim Telnet-Beispiel zu bleiben, sendet der Angreifer zunächst die Pakete, welche die Buchstaben *su* enthalten. Anschließend sendet er ein TCP Reset-Paket. Dieses Reset-Paket weist jedoch eine falsche Sequenznummer oder eine falsche Prüfsumme bei korrekten Portnummern auf. Einfache zustandsorientierte NIDS ordnen dieses Paket der überwachten Verbindung zu und brechen die Beobachtung der Verbindung aufgrund des TCP Resets ab. Der empfangende Rechner verwirft jedoch das Paket als ungültig und nimmt weitere Pakete für diese Verbindung an. Anschließend sendet der Angreifer die restlichen Buchstaben, um das Kommando zu vervollständigen. Der *stream4*-Präprozessor erkennt derartige Zustandsprobleme und -verletzungen und protokolliert diese. Auch SYN-Pakete, welche bereits Nutzdaten enthalten, werden erkannt.

Weitere Funktionen sind die mögliche Protokollierung aller Verbindungen in der Datei *session.log* und die Erkennung von Portscans. Sämtliche Optionen werden im Folgenden angegeben:

- *stream4*:
 - *noinspect*. Keine Zustandsanalyse (Default: Nein)
 - *keepstats*. Speichere Verbindungen in der Datei *<logdir>session.log* (Default: Nein).
 - *timeout*. Dauer (Sekunden), die eine inaktive Verbindung in der Zustandstabelle verbleibt (Default: 30)
 - *memcap*. Arbeitsspeicher. Wenn Snort diese Grenze überschreitet, werden inaktive Verbindungen entfernt. (Default: 8 Mbyte)
 - *detect_scans*. Alarmiere bei einem Portscan (Default: Nein).
 - *detect_state_problems*. Alarmiere bei Zustandsverletzungen (Default: Nein).
 - *disable_evasion_alerts*. Schaltet die Alarmierung für bestimmte Pakete ab:
 - RST-Pakete mit falscher Sequenznummer oder ohne Acknowledgment-Nummer.
 - Pakete mit fehlerhafter Prüfsumme.
 - Pakete mit Sequenznummern, die bereits gesehen wurden.
 - Schnelle Paket-Retransmissionen.
 - SYN-Pakete, die bereits Daten enthalten.
 - *tll_limit*. Diese Option definiert die maximal erlaubte Variation des TTL-Wertes in einer Verbindung (analog *frag2*).
- *stream4_reassemble*
 - *clientonly*. Reassemblierung nur für Client-Pakete (Default: Ja)
 - *serveronly*. Reassemblierung nur für Server-Pakete (Default: Nein)

- both. Reassemblierung für beide Seiten (Default: Nein)
- favor_old. Diese Option bevorzugt alte Verbindungen und vergisst neue bei Auslastung des Speicherbereiches (Default: Ja).
- favor_new. Diese (noch nicht implementierte) Option bevorzugt neue Verbindungen bei Auslastung des Speicherbereiches (Default: Nein).
- noalerts. Keine Alarmierung bei Problemen bei der Reassemblierung (Default: Nein)
- ports. Liste der zu überwachten Ports. all betrachtet alle Ports. default aktiviert die Überwachung folgender Ports: 21, 23, 25, 53, 80, 110, 111, 143 und 513.

Der *stream4*-Präprozessor kann zusätzlich noch über zwei Kommandozeilenooptionen bzw. Konfigurationsdirektiven in seinem Verhalten modifiziert werden.

Die Konfigurationsdirektive `checksum_mode` (`snort -k`) erlaubt das Abschalten der Prüfsummenberechnung für einzelne (`noip`, `notcp`, `noudp`, `noicmp`) oder alle Protokolle (`none`). Diese Prüfsummenberechnung ist sehr zeitaufwändig. Häufig führen jedoch bereits Router oder Switches diese Berechnung durch und werfen ungültige Pakete.

Die Konfigurationsdirektive `stateful` (`snort -z`) ermöglicht die Beschränkung der Analyse von TCP-Paketen auf tatsächlich aufgebaute Verbindungen (`config stateful: est`). Künstlich erzeugte TCP-Pakete, welche nicht Teil einer vorher aufgebauten Verbindung sind, werden nicht von Snort untersucht. In der Standardeinstellung (`config stateful: all`) untersucht die Detektionsmaschine trotz *stream4*-Präprozessor alle Pakete.

Beispielkonfiguration:

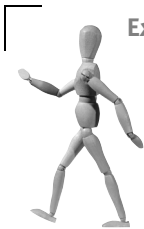
```
# Unser Switch führt bereits Prüfsummenvalidierung durch
config checksum_mode: none
# Lediglich TCP Pakete in aufgebauten Verbindungen betrachten
config stateful: est

preprocessor stream4: keepstats, detect_state_problems
preprocessor stream4_reassemble
```

HTTP-Normalisierung

Es existieren in Abhängigkeit der Snort-Version drei Präprozessoren zur Dekodierung von HTTP URI-Zeichenketten: `http_decode` (Snort 1.x und 2.0), `unidecode` (Snort 1.x) und `http_inspect` (Snort 2.1 und 2.2). Alle drei führen eine Normalisierung der HTTP-Anfragen durch und können zusätzliche Angriffe auf der Ebene des HTTP-Protokolls erkennen.

Diese Plug-Ins wurden in erster Linie geschrieben, um Angriffe, welche in ASCII oder Unicode kodiert sind, auf Webservern zu erkennen.



Exkurs: Was ist Unicode?

Üblicherweise kodieren Computer Buchstaben im ASCII-Alphabet. Da das ASCII-Alphabet die Kodierung in einem Byte durchführt, kann es nicht sämtliche landestypischen Zeichen aufnehmen. Daher wurden zu Beginn so genannte (landestypische) Codepages eingeführt. Auch diese genügen jedoch nicht, um sämtliche chinesischen Schriftzeichen zu kodieren. Hierzu wurde Unicode geschaffen. Dabei handelt es sich um ein neues Alphabet, welches die Kodierung der Zeichen in einem Wort (2 Bytes) vornimmt. Damit können 65.536 verschiedene Zeichen in Unicode kodiert werden. So gibt es zum Beispiel für das Zeichen / (der Schrägstrich) den ASCII-Code `0x2f` und den Unicode `0xc11c`.

Die meisten verfügbaren Webserver sind fähig, ASCII-Code und Unicode genauso wie normale Zeichen zu verarbeiten. Das bedeutet, dass ein / in der URI genau die gleiche Auswirkung hat wie ein `%2f` und ein `%c1%1c`. Wenn nun ein NIDS die URI untersucht, muss es nicht nur mit den üblichen Buchstaben, sondern auch mit ASCII-Kodierungen und auch mit einer Unicode-Kodierung umgehen können. Um nun sämtliche Regeln Hexadezimal- und Unicode-fähig zu machen, müssen alle Regeln in weiteren Formen vorliegen. Das Ganze wird zusätzlich erschwert, da zum Beispiel der / (Schrägstrich) in Unicode mehrere verschiedene Kodierungen kennt (`0xc11c`, `0xc0af`, `0xc19c`).

Einfacher wird die Analyse für Snort, wenn die Detektionsmaschine von Snort lediglich die aus der Kodierung resultierende URI sieht. Hierzu sind die beiden Präprozessoren in der Lage. Sie dekodieren jegliche kodierte Angabe in der URI und Snort untersucht die dekodierte Anfrage.

http_decode und uni_decode

Diese beiden Präprozessoren weisen folgende Optionen auf:

- **Portliste.** Hier können die Ports angegeben werden, bei denen die Dekodierung durchgeführt werden soll.
- **-unicode.** Dies führt dazu, dass lediglich hexadezimale Bytes dekodiert werden, jedoch kein Unicode.
- **-cginull.** Hier werden bei CGI NULL-Code Angriffe nicht erkannt. Dabei handelt es sich um CGI-Aufrufe, welche in der URI ein Byte mit dem Wert 0 enthalten, hinter dem weitere Befehle gesendet werden. Die Programmiersprache C sieht das Nullbyte als Ende der Zeichenkette.

Ein Beispiel für die Anwendung des Unidecode-Präprozessors:

```
preprocessor unidecode: 80 8080 -cginull
```

**Achtung:**

Diese Präprozessoren werden nicht mehr weiterentwickelt und wurden in Snort 2.1 durch den neuen Präprozessor `http_inspect` ersetzt.

http_inspect

Der wesentlich mächtigere Präprozessor `http_inspect` löst die alten Präprozessoren `http_decode` und `uni_decode` ab.

Dieser Präprozessor verfügt über eine Vielzahl von Optionen, die sein Verhalten beeinflussen können. Die meisten dieser Optionen sind jedoch mit sinnvollen Default-Werten vorbelegt, so dass eine Anpassung nur in seltenen Fällen vorgenommen werden muss.

Die Konfiguration des Präprozessors erfolgt in zwei Schritten. Zunächst werden globale Parameter konfiguriert:

```
preprocessor http_inspect: global \
    iis_unicode_map unicode.map codemap 1250
```

Hierfür stehen die folgenden Parameter zur Verfügung:

- `iis_unicode_map`. Dieser Parameter gibt den Dateinamen der zu verwendenden Unicode-Map an. Dabei handelt es sich um eine Datei, die die Unicode-Zeichen in normale ASCII-Zeichen übersetzt. Diese Datei muss sich in demselben Verzeichnis befinden, in dem sich auch die Datei `snort.conf` befindet. Zusätzlich ist die Angabe der Codemap erforderlich. Diese Zahl kann aus der Unicode-Map gelesen werden und ist für Zentraleuropa 1250 und für die USA 1252. Anpassungen der Unicode-Map können mit dem mitgelieferten Werkzeug `ms.unicode_generator.c` erfolgen. Dieses befindet sich in dem `contrib/`-Verzeichnis von Snort.
- `detect_anomalous_servers`. Diese Option aktiviert die Analyse des Verkehrs auf nicht konfigurierten Ports. Sobald hier HTTP-Verkehr erkannt wird, wird eine Alarmierung ausgegeben. Damit können unbekannte HTTP-Server erkannt werden.

**Achtung:**

Diese Option benötigt auch eine Default-Server-Konfiguration (s.u.).

- `proxy_alert`. Diese Option aktiviert die Alarmierung bei der Verwendung nicht definierter Proxies. Hierzu müssen die Proxies als HTTP-Server mit der Option `allow_proxy_use` definiert werden.

Nachdem die globalen Parameter definiert wurden, müssen die einzelnen Webserver, die sich in dem eigenen Netzwerk befinden und überwacht werden sollen, spezifiziert werden. Hierzu gehören dann auch mögliche HTTP-Proxy-Server.

```
preprocessor http_inspect_server: server default \
  profile all ports { 80 8080 8180 } oversize_dir_length 500
preprocessor http_inspect_server: server 192.168.0.5 \
  profile apache ports {80}
```

Bei der Konfiguration der Server sollte möglichst immer eine Default-Konfiguration angegeben werden, die neue noch nicht bekannte Server erkennen kann. Alle existierenden und bekannten Server werden dann einzeln angegeben. Hierzu stehen die folgenden Parameter zur Verfügung. Bei einigen dieser Parameter kann zusätzlich noch ein `yes` oder `no` angegeben werden. Diese Angabe definiert, ob ein Alarm ausgegeben werden soll. Sie deaktiviert nicht die Anwendung des Parameters in dem Normalisierungsprozess. So wird eine ASCII-Kodierung im HTTP-Verkehr bei der Angabe von `ascii no` dekodiert, aber eine Alarmierung ausgeführt.

- `profile`. Hiermit können die anzuwendenden Regeln entsprechend dem eingesetzten Server eingeschränkt werden. Dies reduziert sowohl die Verarbeitungszeit als auch falsch-positive Meldungen. Mögliche Werte sind im Moment `apache`, `iis` und `all`.

Das Profil `all` aktiviert die folgenden Parameter:

Parameter	Wert
<code>flow_depth</code>	300
<code>chunk_length</code>	500000
<code>iis_unicode_map</code>	wie in der globalen Konfiguration
<code>ascii</code>	<code>no</code>
<code>multiple_slash</code>	<code>no</code>
<code>directory</code>	<code>no</code>

Parameter	Wert
double_decode	yes
u_encode	yes
bare_byte	yes
iis_unicode	yes
iis_backslash	no
iis_delimiter	yes
apache_whitespace	yes
no_strict	no

Das Profil apache aktiviert die folgenden Parameter:

Parameter	Wert
flow_depth	300
chunk_length	500000
ascii	no
multiple_slash	no
directory	no
apache_whitespace	yes
utf_8	no
non_strict	no

Das Profil iis aktiviert die folgenden Parameter:

Parameter	Wert
flow_depth	300
iis_unicode_map	wie in der globalen Konfiguration
ascii	no
multiple_slash	no
directory	no
double_decode	yes
u_encode	yes
bare_byte	yes
iis_unicode	yes
iis_backslash	no
iis_delimiter	yes
apache_whitespace	yes
no_strict	no

Das Profil muss als erster Server-Parameter angegeben werden, wenn es benutzt werden soll. Wird ein Profil verwendet, so sind nur noch die folgenden weiteren Parameter erlaubt: `ports`, `iis_unicode_map`, `allow_proxy_use`, `flow_depth`, `no_alerts`, `inspect_uri_only` und `oversize_dir_length`.

- `allow_proxy_use`. Wenn der globale Parameter `proxy_alert` angegeben wurde, erlaubt diese Option die Verwendung des Servers als Proxy und verhindert die Erzeugung einer Alarmierung für diesen Proxy. Wenn der globale Parameter nicht gesetzt wurde, hat diese Option keine Auswirkung.
- `apache_whitespace`. Diese Option erlaubt auch die Verwendung eines `Tab` als Whitespace in einer URI. Dies ist eine Eigenschaft des Apache. Die Alarmierung wird mit `yes` aktiviert. Dies kann jedoch zu falsch-positiven Meldungen führen.
- `ascii`. Dieser Parameter dekodiert ASCII-kodierte Zeichen wie zum Beispiel das sehr häufige `%20` für ein Leerzeichen. Es empfiehlt sich, die Alarmierung mit `no` abzuschalten.
- `bare_byte`. Der Microsoft Internet Information Server erlaubt UTF-8 für die Kodierung von nicht-ASCII-Zeichen. Laut HTTP-Standard sollten diese aber mithilfe des `%`-Zeichen kodiert werden. Obwohl der IIS dies unterstützt, gibt es noch keinen Webclient, der diese Funktion nutzt. Daher ist es sinnvoll, hier auch die Alarmierung mit `yes` zu aktivieren, wenn ein IIS genutzt wird.
- `base36`. Die Base36-Kodierung wird in einigen asiatischen Softwareversionen genutzt. Dieser Parameter aktiviert die Dekodierung. Diese Option schaltet auch automatisch die ASCII-Dekodierung an.
- `chunk_length`. Diese Option benötigt die maximal erlaubte Chunk-Größe und alarmiert, wenn größere Chunks gesehen werden. Damit kann `http_inspect` mögliche Chunk-Encoding-Angriffe gegen den Apache erkennen. Außerdem nutzen einige HTTP-Tunnel Chunk-Encoding. Diese können damit auch möglicherweise erkannt werden.
- `directory`. Dieser Parameter normalisiert Directory-Traversals (`http://www.ms.com/scripts/../../system32/cmd.exe`) und selbst referenzierende URLs (`http://www.ms.com/data/./index.html`) und sollte immer aktiviert werden. Ob eine Alarmierung (`yes`) sinnvoll ist, kann nicht eindeutig gesagt werden. Jedoch gibt es einige Angriffe, die mithilfe des Directory-Traversals auf Dateien außerhalb der Document-Root des Webservers versuchen zuzugreifen.
- `double_decode`. Dieser Parameter dekodiert jede URL in zwei Durchgängen. Dieses Verhalten ist auch typisch für den IIS. Daher sollte dieser Parameter immer angegeben werden, wenn ein IIS überwacht werden soll. Die Alarmierung ist jedoch mit Vorsicht zu aktivieren, da hierbei eine große Anzahl an falsch-positiven Meldungen erzeugt werden können.
- `flow_depth`. Dieser Parameter gibt an, wie viele Bytes der Webserver-Antwort untersucht werden sollen. Da Snort im Moment nur Regeln für den HTTP-Header enthält, genügen hier üblicherweise 300 Bytes. Viele Antworten sind jedoch wesentlich größer, so dass mit dieser Option die Analyse der Antworten eher abgebrochen werden kann.

- `iis_backslash`. Dieser Parameter normalisiert den Backslash in einer URI zum normalen Slash, wie es auch der IIS tut. Ein `yes` führt zusätzlich eine Alarmierung durch.
- `iis_delimiter`. Normalerweise werden HTTP-Anfragen mit einem Carriage-Return/Linefeed (`\r\n`) abgeschlossen. Der IIS und auch der Apache akzeptieren jedoch auch nur einen Linefeed (`\n`). Dieser Parameter führt eine entsprechende Normalisierung durch. Eine Alarmierung mit `yes` ist nicht sinnvoll.
- `iis_unicode`. Dieser Parameter dekodiert Unicode-Anfragen. Hierzu wird die `iis_unicode_map` eingesetzt. Hier sollte die Alarmierung (`yes`) aktiviert werden, da Unicode insbesondere von vielen Angriffen zu Vereitelung der Erkennung durch ein IDS verwendet wird. Diese Option erzwingt auch die Aktivierung von `ascii` und `utf_8`.
- `iis_unicode_map`. Dieser Parameter erlaubt die Angabe einer serverspezifischen Map für die Unicode-Dekodierung. Wenn keine angegeben wird, verwendet der Unicode-Dekoder die globale Map.
- `inspect_uri_only`. Wenn dieser Parameter angegeben wird, betrachtet der `http_inspec`-Präprozessor nur noch die URI in der HTTP-Anfrage. Das bedeutet gleichzeitig, dass nur Regeln mit dem Attribut `uricontent` ausgewertet werden. Regeln mit dem Attribut `content` werden wirkungslos bei Anwendung auf HTTP-Verkehr.
Diese Option kann eine starke Performanzsteigerung erwirken. Jedoch muss sichergestellt sein, dass alle Regeln auf das Attribut `uricontent` umgestellt werden.
- `multi_slash`. Dieser Parameter normalisiert mehrere aufeinanderfolgende Slashes zu einem: Aus `dir/////index.html` wird `dir/index.html`. Ein `yes` aktiviert eine Alarmmeldung beim Auftreten mehrfacher Slashes.
- `no_alerts`. Hiermit werden alle Alarmmeldungen des `http_inspect`-Präprozessors deaktiviert. Die Normalisierungen werden weiterhin durchgeführt.
- `non_rfc_char`. Dieser Parameter erlaubt die Angabe von bestimmten Zeichen, die bei Verwendung in der URI eine Alarmmeldung erzeugen. So kann durch die Angabe von 0 zum Beispiel nach einem Null-Byte in der URI gesucht werden.
- `non_strict`. Der Apache-Webserver akzeptiert URIs mit Leerzeichen (`get /index.html` hier geht es weiter). Dabei wertet er nur die Zeichenkette zwischen dem ersten und dem zweiten Leerzeichen als URI (`/index.html`). Dieser Parameter dekodiert die URI identisch. Eine Alarmierung ist nicht möglich.
- `no_pipeline_req`. HTTP erlaubt es, mehrere Anfragen nacheinander in einer Verbindung zu stellen. Normalerweise analysiert der `http_inspect`-Präprozessor alle diese Anfragen und normalisiert sie entsprechend der aktivierten Parameter. Dieser Parameter deaktiviert diese Eigenschaft.
- `oversize_dir_length`. Dieser Parameter definiert, wie lang maximal das Verzeichnis in der URI sein darf. Längere Verzeichnisse erzeugen eine Alarmmeldung. Ein sinnvoller Wert ist 300.

- `ports`. Dieser Parameter definiert die Ports, die der `http_inspect`-Präprozessor überwachen soll (Typisch: 80). Da Snort SSL-Verkehr nicht entschlüsseln kann, ist es nicht sinnvoll auch den Port 443 zu überwachen!
- `u_encode`. Dieser Parameter normalisiert Kodierungen vom Typ `%uXXXX`. So entspricht zum Beispiel `%u002f` einem Slash. Für diesen Parameter sollte auch die Alarmierung mit `yes` aktiviert werden, da bisher keine Clients diese Kodierung verwenden.
- `utf_8`. Dieser Parameter dekodiert UTF-8 Unicode, wie es auch der Apache-Webserver macht. Eine Alarmierung bei Verwendung von UTF-8 ist kritisch, da es Clients gibt, die diese Kodierung verwenden.

telnet_decode

Das Telnet-Protokoll ist ein wenig komplizierter als weiter oben angesprochen. Leider werden noch sehr viele zusätzliche Zeichen übertragen, welche zur Verwaltung der Telnet-Verbindung genutzt werden. Dieser Präprozessor entfernt diese Zeichen und normalisiert die Daten so, dass Snort später nur die tatsächlich übertragenen Nutzdaten, zum Beispiel eingegebene Befehle, sieht. Dieser Präprozessor akzeptiert als Argument die zu überwachenden Ports. Werden keine Ports angegeben, so überwacht der Präprozessor die Ports 21, 23, 25 und 119:

```
preprocessor telnet_decode: 23
```

Die Normalisierung erfolgt in einem eigenen Speicherbereich, so dass Regeln, die das Attribut `rawbytes` verwenden, immer noch auf die nicht normalisierten Rohdaten zugreifen können.

rpc_decode

Das Remote Procedure Call-Protokoll von Sun bietet ähnlich dem HTTP-Protokoll unterschiedliche Kodierungsmöglichkeiten. Damit das `rpc`-Attribut in der Lage ist, alle verschiedenen Kodierungen und Fragmente zu verarbeiten, normalisiert dieser Präprozessor diese Anfragen. Hierzu unterstützt er die folgenden Parameter:

- Als erster Parameter können die zu überwachenden Ports angegeben werden. Die Default-Ports sind 111 und 32771.
- `alert_fragments`. Alarmiert bei einer fragmentierten Anfrage. Die Aktivierung ist nicht sinnvoll, da dies durchaus legitime Gründe haben kann.
- `no_alert_multiple_requests`. Alarmiert nicht bei mehreren Anfragen in einem Paket.
- `no_alert_large_fragments`. Alarmiert nicht, wenn die Summe der Fragmente die Paketgröße überschreitet.
- `no_alert_incomplete`. Alarmiert nicht, wenn eine einzelne Anfrage bereits die Paketgröße überschreitet. RPC erlaubt durchaus sehr große Anfragen von mehreren MByte Größe. Diese Anfragen überschreiten dann immer die MTU!

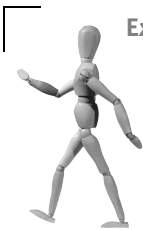
```
preprocessor rpc_decode: 111 32768
```

**Achtung:**

Der `npc_decode`-Präprozessor kann nicht Microsoft-RPC-Anfragen normalisieren. Hierbei handelt es sich um ein anderes Protokoll!

ARP-Spoof-Detection

Hierbei handelt es sich um einen experimentellen Präprozessor, der ARP-Spoofing-Angriffe erkennen kann.

**Exkurs: Was ist ARP-Spoofing?**

ARP-Spoofing ist eine Technik, welche von Netzwerksniffern eingesetzt wird.

Um in einem Netzwerk sämtliche Pakete mitzulesen, ähnlich der Operation, die Snort durchführt, muss sich die entsprechende Netzwerkkarte im so genannten promiscuous-Modus befinden. In diesem Modus nimmt die Netzwerkkarte alle Pakete entgegen und nicht nur die Pakete, welche an ihre oder die Broadcast MAC-Adresse gerichtet sind. Dies funktioniert auch in Netzwerken, welche über Hubs kommunizieren. Wird im Netzwerk jedoch ein Switch eingesetzt, so verteilt der Switch bereits die Pakete in Abhängigkeit der MAC-Adresse. Das bedeutet, dass ein Rechner, welcher an einem Switch angeschlossen wurde, nur noch maximal die Pakete sehen kann, welche an seine MAC- oder die Broadcast MAC-Adresse gesendet wurden. Andere Pakete werden von dem Switch nicht mehr an den Rechner weitergeleitet. Snort leidet in einer geschwichten Umgebung unter demselben Problem. Für den Sniffer heißt die Lösung: ARP-Spoofing. Bevor ein Rechner ein Paket an einen anderen Rechner senden kann, muss er zunächst dessen MAC-Adresse ermitteln. Dies erfolgt in einem ARP (Address Resolution Protocol)-Request. Diese Anfrage ist an alle (Broadcast) gerichtet und bittet um die zu einer IP-Adresse gehörende MAC-Adresse. Kann der Angreifer diese Antwort fälschen und seine eigene MAC-Adresse dem fragenden Rechner unterschieben, so wird das resultierende IP-Paket an die richtige IP-Adresse, aber die falsche MAC-Adresse gesendet. Der Switch kann nicht die IP-Adressen, sondern nur die MAC-Adressen zuordnen. Er wird das Paket zunächst an den Angreifer zustellen. Dieser kann das Paket analysieren und an den echten Empfänger weiterleiten. ARP-Spoofing erlaubt also einem Angreifer in einem

geswitchten Netzwerk dennoch den gesamten oder gezielten Netzwerkverkehr mitzulesen und zum Beispiel auf Kennwörter hin zu analysieren.

Der Präprozessor *arpspoof* ist in der Lage, ARP-Spoofing-Versuche zu erkennen. Dazu benötigt er eine Datenbank mit den IP-Adressen und den dazugehörigen MAC-Adressen. Snort wird dann die Anfragen und die Antworten mit diesen Informationen vergleichen und bei fehlender Übereinstimmung alarmieren. Zusätzlich kann Snort Unicast-ARP-Requests erkennen. Dies wird aktiviert mit der Option `-unicast`. Legitime Unicast-ARP-Requests werden aber durchaus auch von dem Betriebssystem Linux verwendet. Daher ist bei der Aktivierung mit falsch-positiven Meldungen zu rechnen.

```
preprocessor arpspoof: -unicast
preprocessor arpspoof_detect_host: 192.168.111.1 00:50:8B:D0:61:C3
```

**Tipp:**

Ein besseres Werkzeug für diese Aufgabe stellt ARPwatch dar. ARPwatch wird ebenfalls am Ende dieses Kapitels betrachtet.

BackOrifice-Detektor (bo)

Der BackOrifice-Detektor *bo* wurde von Martin Roesch geschrieben, da vor einigen Jahren in einem Vergleich verschiedener NIDS durch eine Fachzeitschrift das Fehlen eines derartigen Detektors als großer Nachteil dargestellt wurde. BackOrifice ist ein Backdoor-Trojaner, der die Kontrolle eines Windows-Rechners von außen ermöglicht. BackOrifice wurde am 3. August 1998 von der Hackergruppe Cult-of-the-dead-Cow (<http://www.cultdeadcow.com>) freigegeben. Dieser BackOrifice-Detektor ist in der Lage, BO-(nicht BO2k-)Verkehr zu erkennen und unter Umständen auch zu entschlüsseln. Bei der Entschlüsselung kann ein Standardwert angegeben werden oder der Detektor versucht eine Entschlüsselung mit brutaler Gewalt (*brute force*). Hierbei werden alle Möglichkeiten für einen Schlüssel durchgespielt.

Ein Beispiel:

```
preprocessor bo
```

Portscan-Detektor

Dieser Detektor versucht Portscans zu erkennen. Hierzu hält er eine Liste sämtlicher Verbindungen vor und prüft, ob ein bestimmter Rechner in einer bestimmten Zeit-

periode einen Grenzwert von zugelassenen Verbindungen überschreitet. Wenn dies der Fall ist, wird ein Alarm ausgelöst. Das bedeutet, dass in der angegebenen Datei der Rechner protokolliert wird.

Der *portscan*-Detektor erkennt auch Stealth-Portscans (NULL, FIN, SYN/FIN und XMAS). Hierbei handelt es sich um Portscans, welche üblicherweise auf dem gescannten Rechner nicht zu einem Protokolleintrag führen. Es ist möglich, durch eine Aktivierung des Portscan-Detektors eine Reduktion der Protokolle zu erlangen, wenn gleichzeitig sämtliche Regeln, die diese Pakete auch melden, deaktiviert werden. Die Pakete würden ansonsten doppelt protokolliert werden.

Das Portscan-Präprozessor-Plug-In erwartet einige Angaben:

```
preprocessor portscan: "Netzwerk" "Anzahl Ports" "Beobachtungsdauer" "Logdatei"
```

Hierbei bedeuten:

- *Netzwerk*. Das zu beobachtende Netzwerk
- *Anzahl Ports*. Der Grenzwert, wann von einem Portscan auszugehen ist
- *Beobachtungsdauer*. Die Zeit (Sekunden), in der der Grenzwert überschritten werden muss
- *Logdatei*. Die Datei, in der diese Informationen protokolliert werden

Ein Beispiel für den Einsatz des Portscan-Detektors folgt:

```
preprocessor portscan: $HOME_NET 10 5 /var/log/snort/portscan.log
```

Häufig stellt man nach kurzer Zeit fest, dass es im Netzwerk Rechner gibt, die derartig häufig Verbindungen zu anderen Rechnern aufbauen, dass sie vom Detektor gemeldet werden. Dies kann jedoch vollkommen normal sein wie zum Beispiel bei DNS-Servern. Daher gibt es die Möglichkeit, gewisse Rechner auszusparsen:

```
preprocessor portscan-ignorehosts: $DNS_SERVERS
```



Achtung:

Dieser Portscan-Detektor wurde in der Version 2.1 mit der Einführung des *flow*-Präprozessors durch den *flow-portscan*-Präprozessor abgelöst. Dieser Portscan-Detektor ist wesentlich mächtiger und wird weiter unten beschrieben.

Conversation

Der *conversation*-Präprozessor ist mit Snort 2.0 eingeführt worden. Mit diesem Präprozessor ist Snort in der Lage, den Verbindungsstatus sämtlicher Protokolle zu

überwachen. Bisher konnte Snort dies nur für das TCP-Protokoll mit dem *stream4*-Präprozessor leisten. Nun kann Snort auch feststellen, dass eine UDP-Verbindung aufgebaut wurde. Diese Verbindungsüberwachung kann dann von weiteren Präprozessoren wie zum Beispiel *portscan2* genutzt werden.

Zusätzlich unterstützt dieser Präprozessor die Erkennung ungewöhnlicher IP-Protokolle. Hierzu können Sie dem Präprozessor eine Liste erlaubter IP-Protokolle angeben.

Im Einzelnen bietet der Präprozessor folgende Parameter:

- `allowed_ip_protocols`. Dieser Parameter gibt die erlaubten IP-Protokolle an. Hier nicht aufgeführte Parameter führen zur Alarmierung (Default: all).
- `timeout`. Dieser Parameter gibt an, wie lange sich Snort an eine Verbindung erinnert. Sieht Snort innerhalb der angegebenen Zeitspanne (Sekunden) kein weiteres Paket, so wird die Verbindung vergessen (Default: 120).
- `alert_odd_protocols`. Wenn dieser Parameter angegeben wird, werden nicht erlaubte IP-Protokolle eine Alarmierung auslösen (Default: Aus).
- `max_conversations`. Dieser Parameter spezifiziert die maximale Anzahl der überwachten Verbindungen (Default: 65.335).

Ein Beispiel für die Anwendung:

```
preprocessor conversation: timeout 30, max_conversations 30000
```



Achtung:

Obwohl aktuelle Snort-Versionen weiterhin diesen Präprozessor enthalten, scheint keine weitere Entwicklung hier mehr zu erfolgen. Dieser Präprozessor ist weitgehend durch den *flow*-Präprozessor abgelöst worden, der einen wesentlich generischeren Ansatz wählt. Die weitere Entwicklung bleibt abzuwarten.

Portscan2

Der *portscan2*-Präprozessor wurde gleichzeitig mit dem *conversation*-Präprozessor in Snort 2.0 eingeführt. Für seine Funktion nutzt der *portscan2*-Präprozessor die Daten des *conversation*-Präprozessors. Daher muss der letztere ebenfalls aktiviert werden, um diesen Präprozessor zu nutzen.

Dieser Präprozessor ist in der Lage, schnelle *nmap*-Scans zu erkennen. Hierzu unterstützt er die folgenden Parameter:

- `scanners_max`. Dieser Parameter definiert, wie viele Scanner maximal gleichzeitig überwacht werden können (Default: 1000).

- `targets_max`. Dieser Parameter gibt an, wie viele Rechner sich in dem Netzwerk befinden, die gescannt werden können (Default: 1000).
- `target_limit`. Hiermit kann ein Schwellwert definiert werden, wie viele Rechner ein Scanner scannen muss, damit eine Alarmierung ausgelöst wird (Default: 5).
- `port_limit`. Dieser Parameter definiert den analogen Schwellwert für die Ports (Default: 20).
- `timeout`. Dieser Wert definiert in Sekunden, wann der Präprozessor seine Daten vergisst (Default: 60).

Beispiel:

```
preprocessor portscan2: 500, 150, 5, 10, 120
```



Achtung:

Gemeinsam mit dem neuen *flow*-Präprozessor ist auch der *flow-portscan*-Präprozessor eingeführt worden, der gegenüber diesem Präprozessor über mehr Möglichkeiten verfügt. Bei Migrationen sollte die neuere Variante eingesetzt werden.

Httpflow

Der *httpflow*-Präprozessor wurde mit Snort 2.0 eingeführt und mit Snort 2.1 bereits ersetzt durch den mächtigeren *http_inspect*-Präprozessor. Dennoch soll er hier beschrieben werden, damit Sie für Migrations- oder Upgrade-Projekte seine Funktion kennen und in den modernen Snort-Versionen nachbilden können.



Achtung:

Der Präprozessor *httpflow* wurde in Snort 2.1 durch *http_inspect* ersetzt und sollte nicht mehr eingesetzt werden!

Moderne Netzwerke transportieren meist zu mehr als 50% HTTP-Verkehr. Dabei machen die Antworten der Webserver etwa 90-95% des HTTP-Verkehrs aus. Die Angriffe verstecken sich aber meist in den Anfragen der Webclients. Der *http_flow*-Präprozessor ist in der Lage, die Antworten von den Anfragen zu trennen. Die Analyse be-

schränkt sich dann lediglich auf die Anfragen und die HTTP-Header der Antworten. Damit kann eine erhebliche Geschwindigkeitssteigerung erreicht werden.

Der Präprozessor unterstützt zwei verschiedene Modi: `quick` und `full`. Im `quick`-Modus analysiert der Präprozessor jedes einzelne Paket und bietet die folgenden Parameter:

- `depth`. Mit diesem Parameter kann die Analysetiefe der Serverantworten definiert werden (analog zu dem Parameter `flow_depth` des `http_inspect`-Präprozessors; Default: 200).
- `ports`. Dieser Parameter definiert die zu überwachenden Ports (analog zu dem gleichnamigen Parameter bei `http_inspect`; Default: 80).

Beispiel:

```
preprocessor httpflow: quick depth 200 ports 80 8080
```

Im `full`-Modus nutzt der Präprozessor den `stream4`-Präprozessor für die Reassemblierung. Weitere Parameter können nicht angegeben werden. Der Präprozessor erkennt selbstständig die HTTP-Header und Ports.

Beispiel:

```
preprocessor httpflow: full
```

Flow

Der mit Snort 2.1 eingeführte `flow`-Präprozessor ersetzt den `conversation`-Präprozessor. In der Zukunft soll dieser Präprozessor zentral für Snort die Zustände aller Verbindungen überwachen. Bisher wird diese Funktion jedoch nur von dem `flow-port-scan`-Präprozessor genutzt.

Der `flow`-Präprozessor hält die Informationen über alle Verbindungen in großen Hash-Tabellen vor. Er unterscheidet die verschiedenen IP-Verbindungen anhand ihres IP-Protokolls, der Absender-IP-Adresse, der Ziel-IP-Adresse und des Absender- und Ziel-Ports im Falle von TCP und UDP. Wenn es sich nicht um TCP- oder UDP-Verbindungen handelt, verwendet der Präprozessor intern die Portnummer 0.

Dieser Präprozessor kann mit den folgenden Parametern konfiguriert werden:

- `memcap`. Größe des Arbeitsspeichers für `flow` (Default: 10.485.760 = 10 MByte).
- `rows`. Anzahl Zeilen in der Hash-Tabelle (Default: 4096).
- `stats_interval`. Zeitintervall in Sekunden, in dem der Präprozessor statistische Informationen ausgibt (Default: 0 = abgeschaltet).
- `hash`. Hash-Methode: 1 oder 2 (schneller, aber möglicherweise kollisionsanfälliger; Default: 2).

Flow-Portscan

Dieser Präprozessor erkennt Portscans, in denen ein Rechner mehrere andere Rechner oder mehrere Ports auf einem Rechner scannt. Der *flow-portscan*-Präprozessor ersetzt den *portscan2*-Präprozessor von Jason Larsen und Jed Haile.

Die neue Architektur benötigt wesentlich weniger Speicher als der alte Präprozessor und sollte darüberhinaus Portscans wesentlich sicherer erkennen und falsch-positive Meldungen reduzieren.

Im Wesentlichen besteht der Präprozessor aus drei Komponenten:

- *Scoreboards*. Es existieren zwei Scoreboards: Talker und Scanner. Talker sind IP-Adressen, die regelmäßig aktiv Netzwerkverkehr erzeugen. Scanner sind IP-Adressen, die einen vorher nicht genutzten Port angesprochen haben. Die Scoreboards enthalten Zeitskalen für diese IP-Adressen.
- *Uniqueness-Tracker*. Diese Komponente prüft, ob eine Verbindung neuartig ist. Dabei werden Quell-IP-Adresse, Ziel-IP-Adresse und Ziel-Port verglichen. Wenn diese Verbindung noch nicht gesehen wurde, wird sie weiter analysiert und in die Bewertung aufgenommen. Der Quell-Port geht nicht in die Bewertung ein.
- *Server Statistics*. Diese Komponente beobachtet das *server-watchnet*. Sie zählt in einem *Hitcount*, wie häufig ein Zugriff auf die Dienste erfolgt. Dabei unterscheidet sie die Ziel-IP-Adresse, den Ziel-Port und das Protokoll.

Dabei geht der Präprozessor folgendermaßen vor:

1. Der *flow*-Präprozessor meldet eine neue Verbindung an den *flow-portscan*-Präprozessor.
2. Der Uniqueness-Tracker prüft, ob es sich um eine neuartige Verbindung handelt. Wenn dies nicht der Fall ist und die Verbindung normale TCP-Flags verwendet, beendet der *flow-portscan*-Präprozessor die weitere Analyse.
3. Wenn die Verbindung nicht an das *server-watchnet* gerichtet ist, wird die Verbindung als Talker bezeichnet.

Ist die Verbindung an das *server-watchnet* gerichtet, so prüft der Präprozessor, ob er sich noch in der *server-learning-time* befindet. In dieser Zeit wird lediglich der *Hitcount* in der Server-Statistik inkrementiert. Ist die Zeit abgelaufen, so vergleicht er den *Hitcount* mit dem *server-ignore-limit*. Ist der *Hitcount* niedriger, so zählt die Verbindung als Scanner.

4. Sowohl Scanner als auch Talker besitzen jeweils eine fixe und eine variable Zeitskala. Die fixe Zeitskala detektiert Ereignisse in einem festen Zeitfenster (*scanner-fixed-window*). Bei Überschreiten eines Schwellwertes (*scanner-fixed-threshold*) wird ein Alarm ausgelöst. Die variable Zeitskala passt ihr Ende immer an eingehende Ereignisse an. Hierzu wird nach jedem Ereignis das Ende neu berechnet:

$$\text{end} = \text{end} + ((\text{end} - \text{start}) * \text{sliding-scale-factor})$$

Dadurch vergrößert sich effektiv das Zeitfenster mit jedem Ereignis. Auch hier kommt es zu einem Alarm, wenn innerhalb des Zeitfensters (`scanner-sliding-window`) der Schwellenwert (`scanner-sliding-threshold`) überschritten wird.

Im Folgenden werden die Parameter des `flow-portscan`-Präprozessors einzeln vorgestellt:

- `scoreboard-memcap-scanner`. Speichergröße für das Scanner-Scoreboard. Default: 6291456.
- `scoreboard-memcap-talker`. Speichergröße für das Talker-Scoreboard. Default: 25165824.
- `scoreboard-rows-scanner`. Anzahl der Zeilen im Scanner-Scoreboard. Default: 250000



Achtung:

Je mehr Zeilen Sie verwenden, desto mehr Speicher wird benötigt. Dennoch kann ein größerer Hash-Table aufgebaut werden, der die Geschwindigkeit erhöht. Sie sehen diesen Speicherbereich als Overhead-Bytes in der Ausgabe des `flow`-Präprozessors beim Beenden von Snort.

- `scoreboard-rows-talker`. Anzahl der Zeilen in dem Talker-Scoreboard (Default: 1000000).
- `scanner-fixed-window`. Dauer des Zeitfensters in Sekunden (Default: 15).
- `scanner-fixed-threshold`. Dieser Schwellenwert definiert, wann ein Alarm im festen Zeitfenster ausgegeben wird (Default: 15).
- `scanner-sliding-window`. Dauer des variablen Zeitfensters in Sekunden (Default: 20).
- `scanner-sliding-threshold`. Dieser Schwellenwert definiert, wann ein Alarm im variablen Zeitfenster ausgegeben wird (Default: 40).
- `scanner-sliding-scale-factor`. Vergrößerungsfaktor für das variable Zeitfenster (Default: 0.5).
- `talker-fixed-window`. Dauer des Zeitfensters in Sekunden (Default: 30).
- `talker-fixed-threshold`. Dieser Schwellenwert definiert, wann ein Alarm im festen Zeitfenster ausgegeben wird (Default: 15).
- `talker-sliding-window`. Dauer des variablen Zeitfensters in Sekunden (Default: 30).

- `talker-sliding-threshold`. Dieser Schwellenwert definiert, wann ein Alarm im variablen Zeitfenster ausgegeben wird (Default: 40).
- `talker-sliding-scale-factor`. Vergrößerungsfaktor für das variable Zeitfenster (Default: 0.5).
- `server-learning-time`. In dieser Zeit lernt der Präprozessor die Dienste, die sich auf den Server in dem `server-watchnet` befinden (Default: 28800).
- `server-ignore-limit`. Ignoriere Ports, die mehr Anfragen als dieses Limit in der `server-learning-time` erhalten haben (Default: 500).
- `server-scanner-limit`. Wie viele Verbindungen zu einem Port aufgebaut werden müssen, bevor er als Talker angesehen wird (Default: 500).
- `server-memcap`. Speicher, in dem die gelernten Server abgelegt werden (Default: 2097152).
- `server-rows`. Anzahl der Zeilen für die Server (Default: 65536).
- `server-watchnet`. Diese IP-Adressen werden als Server angesehen. Der Präprozessor lernt die zur Verfügung stehenden Dienste.
- `unique-memcap`. Speichergröße für den *Uniqueness-Tracker* (Default: 25165824). Der *flow-portscan*-Präprozessor ist in der Lage, um so genauer falsch-positive Portscans zu erkennen, je mehr Speicher ihm zur Verfügung steht.
- `unique-rows`. Anzahl der Zeilen für den *Uniqueness-Tracker* (Default: 1000000).
- `dst-ignore-net`. Diese Ziel-IP-Adressen werden ignoriert.
- `src-ignore-net`. Diese Quell-IP-Adressen werden ignoriert.
- `tcp-penalties`. Diese Option kann an- (*on*, default) oder abgeschaltet (*off*) werden. Ist sie angeschaltet, so wirken sich die TCP-Flags folgendermaßen aus:
 - *SYN, SYN+ECN*: 1 Punkt (*base-score*)
 - *SYN+FIN+ACK*: 5 Punkte
 - *SYN+FIN, kein ACK*: 3 Punkte
 - *Alles weitere*: 2 Punkte
- `alert-mode`. Hier gibt es zwei Einstellungen. Bei *all* meldet der Präprozessor jeden Portscan. Bei *once* (Default) wird nur der erste Scan gemeldet.
- `output-mode`. Hier gibt es ebenfalls zwei Einstellungen: *msg* und *pktkludge*. Im ersten Fall generiert der Präprozessor eine Textmeldung und im letzteren Fall erzeugt er ein gespooftes Paket und verwendet das normale Output-Plug-In.
- `base-score`. Der *base-score* definiert die Punktzahl für eine neue Verbindung (Default: 1).
- `dumpall`. Diese Option gibt den Inhalt aller Tabellen bei der Beendigung von Snort aus, wenn dieser Parameter auf 1 gesetzt wird.

Perfmonitor

Dieser Präprozessor misst die echte und die theoretische Leistung von Snort und gibt diese entweder auf dem Bildschirm oder in einer Datei aus. Dabei gibt Snort die folgenden Informationen aus:

- Empfangene Pakete
- Verworfen Pakete
- % verworfene Pakete
- kpackets/Sekunde
- Durchschnittliche Paketgröße
- Mbit/s Bandbreite, die physikalisch auf dem Draht vorgefunden wird
- Mbit/s Bandbreite, die von der Reassemblierungsmaschine injiziert wird
- Mbit/s Bandbreite (komplett)
- % Anteil des Netzwerkverkehrs, der von der Mustererkennung analysiert wird
- CPU-Nutzung
- Alarmmeldungen pro Sekunde
- TCP-SYN-Pakete pro Sekunde
- TCP-SYN/ACK-Pakete pro Sekunde
- Sessions pro Sekunde
- Beendete Sessions pro Sekunde
- Gesamtzahl aller Sessions
- Maximale gleichzeitige Sessions
- Stream-Fehler pro Sekunde
- Stream-Timeouts
- Durchgeführte Defragmentierungen pro Sekunde
- Neue Fragmente pro Sekunde
- Fragment-Timeouts
- Fragment-Fehler

Die Ausgabe der statistischen Informationen kann mit den folgenden Parametern angepasst werden:

- `flow`. Dieser Parameter gibt zusätzliche Informationen über die Protokollverteilung aus.
- `events`. Dieser Parameter gibt zusätzliche Informationen über die Signaturen aus, die eine Alarmmeldung erzeugten. Dabei wird unterschieden zwischen der reinen Mustererkennung (unqualifiziert) und der Validierung durch die zusätzlichen Attribute in der Regel (qualifiziert).
- `max`. Dieser Parameter schaltet die Berechnung der maximalen theoretischen Leistung von Snort an. Diese Berechnung ist fehlerhaft auf Mehrprozessorsystemen.

- `console`. Dieser Parameter gibt die Daten auf dem Bildschirm aus (Default).
- `file`. Dieser Parameter schreibt die Daten in die angegebene Datei.
- `time`. Dieser Parameter gibt das Zeitintervall in Sekunden zwischen den Statistikausgaben an.

Hier ein Beispiel:

```
preprocessor perfmonitor: time 30 flow events max file snort.statistics
```

SPADE

Der im letzten Abschnitt vorgestellte Portscan-Detektor ist in der Lage, Portscans an der Häufigkeit der initiierten Verbindungen zu erkennen. Werden diese Portscans jedoch sehr langsam oder ausgehend von mehreren unterschiedlichen Rechnern ausgeführt, ist der Portscan-Detektor nicht in der Lage, diese zu erkennen. Dies ist die Aufgabe von SPICE. SPICE ist die Stealthy Probing and Intrusion Correlation Engine. SPICE besteht aus SPADE und einem Korrelator. SPADE ist die Statistical Packet Anomaly Detection Engine. Der Korrelator steht noch nicht zur Verfügung, sondern befindet sich noch in der Entwicklung durch Silicondefense (<http://www.silicondefense.com/>), den Entwickler von SPADE.



Achtung:

SPADE, die Statistical Packet Anomaly Detection Engine, wird nicht mehr weiterentwickelt. Aktuelle Snort-Versionen enthalten allerdings im *contrib*-Verzeichnis noch die SPADE-Version 092200 vom 22. September 2000. Aus diesem Grund wurde der folgende Abschnitt noch nicht entfernt. Dennoch sollte SPADE nicht mehr eingesetzt werden. Die Website der Firma Silicondefense ist seit einigen Monaten nicht mehr erreichbar.

Für die Erkennung von Portscans sollte ab Snort 2.1 der Präprozessor `flow-portscan` eingesetzt werden.

Wie funktioniert nun SPADE? SPADE beobachtet das Netzwerk und vergibt Anomaliewerte. Je seltener ein bestimmtes Paket im Netzwerk vorkommt, desto höhere Anomaliewerte bekommt es zugewiesen. Dazu pflegt SPADE intern eine Wahrscheinlichkeitstabelle, in der die Wahrscheinlichkeit einer bestimmten IP-Adressen-/Port-Paarung vorgehalten und aktualisiert wird. Die Anomalie ergibt sich logarithmisch aus der Wahrscheinlichkeit.

Zitat aus der Datei `README.spade`:

We would know, for example, that $P(dip=10.10.10.10, dport=8080)$ is 10% but that $P(dip=10.10.10.10, dport=8079)$ is 0.1%. The anomaly score is calculated directly from the probability. For a packet X, $A(X) = -\log_2(P(X))$. So the anomaly score for a 10.10.10.10, 8080 packet is 3.32 (not very anomalous) and the score for a 10.10.10.10, 8079 packet is 9.97 (fairly anomalous (?)).

Werte oberhalb eines bestimmten Grenzwertes werden dann von SPADE protokolliert.

Zusätzlich kann SPADE statistische Auswertungen der in einem Netzwerk existierenden Pakete erstellen. Dies erlaubt eine statistische Analyse der in einem Netzwerk verwendeten Ports und Dienste.

SPADE ist nicht in der Lage, gut von böse zu unterscheiden. Es ist lediglich eine Methode, seltene und damit ungewöhnliche Pakete zu erkennen. SPADE kann keine Korrelation von verwandten Paketen durchführen. Dies ist die Aufgabe des sich in der Entwicklung befindlichen Korrelators.

SPADE erzeugt seine Meldungen dort, wo auch sämtliche anderen Snortregeln Alert-Meldungen erzeugen. Dabei können zwei verschiedene Ausgaben erzeugt werden. Zum einen werden ungewöhnliche Pakete protokolliert, zum anderen meldet SPADE automatische Anpassungen des Grenzwertes. SPADE ist in der Lage, seinen Grenzwert selbst automatisch anzupassen (s.u.).

Die Verwendung von SPADE erfordert folgende Konfiguration:

```
preprocessor spade: "Grenzwert" "Zustandsdatei" "Logdatei"  
↳ "Wahrscheinlichkeitsmodus" "Sicherungsfrequenz" "-corrscore"
```

Hierbei bedeuten:

- *Grenzwert*. Der Grenzwert, ab dem ein Paket als unnormal angesehen und protokolliert wird. Eine negative Zahl deaktiviert die Protokollierung. (Default: -1)
- *Zustandsdatei*. Hier werden die Wahrscheinlichkeitstabellen abgespeichert, so dass bei einem Neustart von SPADE die Tabellen nicht neu aufgebaut werden müssen, sondern aus dieser Datei eingelesen werden (Default: spade.rcv).
- *Logdatei*. SPADE-Protokolldatei. Hier werden die Anzahlen der untersuchten Pakete und der generierten Alarmmeldungen protokolliert. Die eigentlichen Alarmmeldungen werden von Snort zentral protokolliert.
- *Sicherungsfrequenz*. Diese Zahl gibt an, nach wie vielen Paketen regelmäßig die Zustandsdatei neu geschrieben werden soll. Eine 0 deaktiviert diesen Vorgang (Default: 50000).
- *Wahrscheinlichkeits-Modus*. Es existieren vier verschiedene Modi, um die Wahrscheinlichkeit des Paketes zu berechnen. Weitere Informationen enthält die Datei *README.Spade.Usage*. Eine Änderung des Wahrscheinlichkeits-Modus erfordert eine neue Zustandsdatei. Die alte Zustandsdatei muss gelöscht werden, da ansonsten falsche Wahrscheinlichkeiten ermittelt werden. (Default: 3)
- *-corrscore*. Ein Fehler in alten Implementierungen führte zu falschen absoluten Berechnungen. (Die Datei *README.Spade.Usage* hat genauere Angaben.) Aus Kompatibilitätsgründen ist diese »fehlerhafte« Berechnung noch das Standardverhalten. Diese Option korrigiert dies. Des Weiteren verwendet die Alarmierung statt »spp_anomsensor:« den Namen »Spade:«.

Ein Beispiel:

```
preprocessor spade: 9 spade.rcv spade.log 3 50000
```

Spade wird alle Pakete mit einem Anomaliewert größer 9 protokollieren. Dabei wird die Wahrscheinlichkeit nach dem Modell 3 (Default) ermittelt. Die Zustandsdatei wird alle 50.000 Pakete (Default) in der Datei *spade.rcv* (Default) gespeichert. Allgemeine Protokolle werden in der Datei *spade.log* vorgehalten.

Spade erlaubt weiteres Tuning seiner Funktion. Zunächst kann das *spade-homenet* definiert werden. Diese Direktive weist Spade an, nur noch an dieses Netzwerk gerichtete Pakete zu analysieren.

```
preprocessor spade-homenet: $HOME_NET
```

Das größte Problem beim Spade-Tuning ist jedoch die Ermittlung eines geeigneten Grenzwertes. Wird dieser Grenzwert zu niedrig gewählt, wird Spade viele vollkommen normale Pakete als unnormal melden. Ist er zu hoch gewählt, so wird Spade viele Pakete übersehen. Spade bietet verschiedene Hilfen hierfür an.

- **Automatisches Erlernen des Grenzwertes.** Spade bietet die Möglichkeit, den Netzwerkverkehr für eine gewisse Zeit (Dauer in Stunden, Default 24) zu beobachten und anschließend den Grenzwert zu errechnen und zu melden, der für eine gewisse Anzahl an Meldungen (*Events*, Default 200) benötigt worden wäre.

```
preprocessor spade-threshlearn: Events Dauer
```

Zwischenberichte können mit einem `killall -HUP snort` erzeugt werden.

- **Automatische Grenzwertanpassung I.** Hierbei beobachtet Snort den Trend und passt in regelmäßigen Abständen den Grenzwert an. Dies wird in der Spade-Protokolldatei protokolliert. Hierbei wird die Zeitdauer (*Dauer*, Default 2) definiert, in der eine gewünschte Anzahl von Meldungen (*Events*, Default 20) erzeugt werden soll. Anschließend wird die Wichtung (*Wichtung*, Default 0.5) angegeben. Eine Wichtung größer 0.5 berücksichtigt die neuen Werte stärker. Abschließend kann optional der Wert 1 angegeben werden, der dazu führt, dass die Dauer nicht in Stunden, sondern in Paketen gemessen wird.

```
preprocessor spade-adapt: Events Dauer Wichtung 0 (oder 1)
```

- **Automatische Grenzwertanpassung II.** Hierbei beobachtet Snort ebenfalls den Trend, führt jedoch eine Langzeitwichtung durch. Snort benötigt die Angabe der Beobachtungsdauer (*Dauer* in Minuten, Default: 15) und die gewünschten Events. Sind die angegebenen größer oder gleich 1, handelt es sich um die Anzahl gemeldeter Pakete pro Zeiteinheit (*Dauer*). Ist diese Zahl kleiner 1, so handelt es sich um den prozentualen Anteil am Netzwerkverkehr (Default: 0.01). Anschließend werden drei Wichtungszeiträume definiert: Kurzzeit (Einheit sind die definierten Beobachtungzeiträume *Dauer*, Default: 4), *Mittelzeit* (Einheit ist die *Kurzzeit*, Default: 24) und *Langzeit* (Einheit ist die *Mittelzeit*, Default: 7).

```
preprocessor spade-adapt2: Events Dauer Kurzzeit Mittelzeit Langzeit
```

- **Automatische Grenzwertanpassung III.** Hierbei führt Snort ebenfalls eine Langzeittrendanalyse durch. Diese ist jedoch wesentlich einfacher als die gerade vorgestellte Variante. Es werden die angestrebten Meldungen (*Events*, Default: 0.01) pro Zeiteinheit (*Dauer* in Minuten, Default: 60) definiert. Anschließend ermittelt Snort das Mittel des optimalen Grenzwertes der letzten *N* (Default: 168) Beobachtungszeiträume.

```
preprocessor spade-adapt3: Events Dauer N
```

Für die zukünftige Unterstützung von dem sich in Entwicklung befindenen Korrelator gibt es die Möglichkeit, mit dem Output-Plug-In `idmef` (s.u.) Pakete, die den Anomaliegrenzwert überschreiten, an einen anderen Rechner zu senden.

```
preprocessor spade-correlate: IP-Adresse Port
```

Schließlich kann Spade auch statistische Berichte seiner Arbeit verfassen. Diese Berichte betrachten immer einen bestimmten Beobachtungszeitraum (*Dauer* in Minuten, Default: 60) und können in einer Datei abgespeichert werden (*Logdatei*, Default: Standardausgabe).

```
preprocessor spade-survey: Logdatei Dauer
```

Zusätzlich existiert jedoch noch ein Statistik-Modus, in dem statistische Informationen über den Netzwerkverkehr gesammelt und protokolliert werden. Hierbei stehen Entropie- (`entropy`), einfache Wahrscheinlichkeiten (`uncondprob`) und bedingte Wahrscheinlichkeiten (`condprob`) zur Verfügung. Diese werden als Optionen angegeben.

```
preprocessor spade-stats: option option option
```

Spade ist sicherlich ein sehr interessantes Plug-In. Leider fehlt ihm bisher die Eigenschaft der Korrelation. Da Spade momentan ein experimentelles Plug-In ist und darüber hinaus hohe Prozessorleistung verlangt, sollte das Spade-Plug-In nicht unüberlegt aktiviert werden. Im Zweifelsfall ist es sicherlich sinnvoller, zwei Snort-Prozesse zu starten. Ein Snort-Prozess übernimmt die »übliche« Arbeit, während der zweite Snort-Prozess lediglich das Spade-Plug-In startet und nutzt. Siehe auch den Abschnitt »Tuning« auf S. 343 weiter unten.

Netzwerkdatenuntersuchung zur Regelentwicklung

Im Abschnitt über Bufferoverflows wurde beschrieben, wie die Entwicklung von Regeln erfolgen kann, wenn der Quelltext des Angriffswerkzeuges existiert. Häufig steht jedoch der Quelltext nicht zur Verfügung. Lediglich die Binärprogramme oder mitgeschnittener Netzwerkverkehr sind möglicherweise vorhanden.

Im Folgenden soll am Beispiel des bereits vorgestellten Bufferoverflows die Vorgehensweise besprochen werden.

Sie benötigen idealerweise drei Rechner, können das Ganze aber auch auf einem lokalen Rechner durchführen. Die benötigte Software besteht aus dem Bufferoverflow in Binärform, Snort und idealerweise Netcat (http://www.atstake.com/research/tools/index.html#network_utilities). Netcat ist ein Werkzeug, welches in der Lage ist, Informationen über das Netzwerk mit dem TCP- oder UDP-Protokoll zu versenden und entgegenzunehmen. In diesem Fall dient es als IMAP-Server-Ersatz, da kein echter Bufferoverflow erzeugt werden soll.

Zunächst konfigurieren Sie Netcat auf dem Server. Netcat soll den IMAP-Server auf Port 143 ersetzen. Dazu starten Sie Netcat mit den Optionen:

```
nc -l -p 143
```

Anschließend konfigurieren Sie Snort auf einem zweiten Rechner. Snort soll sämtlichen Verkehr auf dem Port 143 protokollieren:

```
snort -l ./imaplog -b port 143
```

Schließlich können Sie den Client konfigurieren. Dazu verwenden Sie die Binärform des Bufferoverflows. Diese rufen Sie auf und übergeben die Ausgabe an Netcat, welche sie an den Server auf Port 143 weiterleitet.

```
./a.out 0 | nc server 143
```



Tipp:

Wenn Sie dieses Beispiel nachempfinden möchten, so können Sie sich selbst den IMAP-Bufferoverflow übersetzen. Sie erhalten den Quelltext an der oben angegebenen Adresse und können ihn einfach mit dem GNU C Compiler übersetzen. Unter Umständen gibt es eine Fehlermeldung, da die letzte geschweifte Klammer im Quelltext fehlt.

```
gcc ADM-imap4r1-linux.c
```

Sie können den kompletten Aufbau auch auf einem einzelnen Rechner durchführen. Hierzu definieren Sie beim Aufruf von Snort das Loopback-Interface mit der Option `-i lo` und geben beim Aufruf des Clients als Servernamen `localhost` an.

```
snort -l ./imaplog -b -i lo port 143  
./a.out 0 | nc localhost 143
```

Anschließend können Sie mit dem Kommando

```
snort -vdr imaplog/snort-0514\@1129.log
```

die protokollierten Daten analysieren:


```

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 .....
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 .....
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 .....
90 90 90 90 90 90 90 90 90 90 EB 38 5E 89 F3 89 .....8^...
D8 80 46 01 20 80 46 02 20 80 46 03 20 80 46 05 ..F. .F. .F. .F.
20 80 46 06 20 89 F7 83 C7 07 31 C0 AA 89 F9 89 .F. ....1.....
F0 AB 89 FA 31 C0 AB B0 08 04 03 CD 80 31 DB 89 ....1.....1..
D8 40 CD 80 E8 C3 FF FF FF 2F 42 49 4E 2F 53 48 .@...../BIN/SH
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 .....

```

.... weitere Daten folgen

Bei der Analyse der Pakete ist zunächst der TCP-Handshake zu sehen. Dies ist ein ganz normaler TCP-Handshake. Er wurde auch nicht vom dem IMAP-Bufferoverflow, sondern von Netcat generiert. Das interessante Paket ist das vierte sehr große Paket. Dieses Paket weist folgende Merkmale auf:

- Es ist ein TCP-Paket mit den gesetzten Flags ACK und PSH und ist an Port 143 gerichtet.
- Es beginnt mit einem Text AUTHENTICATE.
- Es weist am Ende den Text /BIN/SH auf.
- Im mittleren Bereich besitzt es einen so genannten NOP-Schlitten (NOP sled). Dieser wird häufig in Bufferoverflows verwendet (siehe den Exkurs »Bufferoverflows« auf S. 285). NOP ist ein Assemblerbefehl, bei dem der Prozessor keine Operation durchführt (No Operation). Er wird hexadezimal 0x90 kodiert.

Mit diesem Wissen lässt sich nun folgende Regel implementieren

```

alert tcp any any -> $HOME_NET 143 (flags: AP+; content:"AUTHENTICATE"; \
  content "/BIN/SH"; content: "|9090 9090 9090 9090 9090 9090|"; \
  msg:"IMAP Bufferoverflow");)

```

Diese Regel wird ebenso den Bufferoverflow erkennen wie die oben entwickelte Regel.

9.2.10 Fortgeschrittene Protokollierung

In diesem Kapitel werden die Output-Plug-Ins von Snort vorgestellt. Ihr Einsatz in einem Unternehmensnetzwerk mit mehreren Snort-Sensoren wird später im Buch vorgestellt. In diesem Kapitel werden jedoch bereits die zur Verfügung stehenden Plug-Ins und ihr Einsatz beschrieben. Es gibt zwei unterschiedliche Formen der Plug-Ins: *alert* und *log*. Alert-Plug-Ins protokollieren die Snort-Meldung und Informationen über das Paket. Log-Plug-Ins protokollieren das Paket selbst.

**Achtung:**

Die in der Konfigurationsdatei definierten Output-Plug-Ins sind nur aktiv, wenn beim Aufruf von Snort auf der Kommandozeile keine Output-Optionen (wie `-l`, `-s`, `-A` etc.) definiert wurden. Funktionieren die Output-Plug-Ins nicht, prüfen Sie den Aufruf von Snort!

Der Snort Event-Queue

Ab Version 2.1 unterstützt Snort die Modifikation seines Event-Queue. Hiermit ist es möglich, mehrere Alarmmeldungen pro Paket zu erzeugen. Dieser Event-Queue war auch schon in Snort 2.0 enthalten, jedoch ist es hier nicht möglich, diesen anzupassen und mehrere Alarmmeldungen pro Paket auszugeben.

Um den Event-Queue anzupassen, unterstützt Snort die folgenden Parameter:

```
config event_queue: [max_queue <size>] [log <size>] [order_events <TYPE>]
```

Die Parameter haben hierbei die folgende Bedeutung:

- `max_queue`. Dieser Parameter definiert, wie lang der Event-Queue maximal ist. Bei der Default-Einstellung werden maximal acht Ereignisse pro Paket oder Stream im Queue gespeichert.
- `log`. Dieser Parameter definiert, wie viele Meldungen maximal pro Paket oder Stream protokolliert werden. Default-Wert ist 3.
- `order_events`. Hiermit können die Ereignisse im Event-Queue unterschiedlich sortiert werden. Während die Angabe `content_length` (default) die Ereignisse in Abhängigkeit von der Länge des `content`-Attributes in der zutreffenden Regel sortiert, sortiert die Angabe `priority` die Ereignisse in der Reihenfolge ihrer Priorität.

Klartextprotokollierung in eine Datei

Die Default-Einstellung von Snort ist die Klartextprotokollierung. Diese Protokollierung wird durch zwei verschiedene Output-Plug-Ins zur Verfügung gestellt: `alert_fast` und `alert_full`.

alert_fast

Dieses Output-Plug-In protokolliert jedes Paket in der Alert-Protokolldatei (Default: `/var/log/snort/alert`) auf einer einzelnen Zeile. Hierbei handelt es sich sicher um eines der schnellsten Output-Plug-Ins. Jedoch werden die Header und auch der Inhalt des Paketes nicht protokolliert. Diese Option kann jedoch gut zusammen mit der Binärprotokollierung eingesetzt werden.

```
output alert_fast: /var/log/snort/alert
```

Die Ausgabe sieht dann wie folgt aus:

```
05/14-12:31:11.631460  [**] [1:528:2] BAD TRAFFIC loopback traffic [**]
↳ [Classification: Potentially Bad Traffic] [Priority: 2] {ICMP}
↳ 127.0.0.1 -> 127.0.0.1
```

Sämtliche Informationen befinden sich in einer Zeile, zu Beginn Datum und Uhrzeit. Anschließend eingefasst in `[**]` die Regelidentifikation `[1:528:2]` (Generator, Regel-Id, Version) und die Mitteilung der Regel. Hieran schließt sich die Klassifizierung und die Priorität der Regel an. Schließlich werden das Protokoll und die beteiligten IP-Adressen und Ports angegeben, wenn vorhanden. Eine darüber hinausgehende Protokollierung wird von `alert_fast` nicht durchgeführt.

alert_full

Dieses Output-Plug-In protokolliert jedes Paket mehrfach. Zunächst schreibt es ähnlich `alert_fast` einen Eintrag in der Alert-Protokolldatei. Dieser ist jedoch wesentlich ausführlicher und enthält zusätzlich den Header des Paketes:

```
[**] [1:528:2] BAD TRAFFIC loopback traffic [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
05/14-12:36:55.941462 127.0.0.1 -> 127.0.0.1
ICMP TTL:255 TOS:0x0 ID:30644 IpLen:20 DgmLen:84
Type:0 Code:0 ID:49441 Seq:768 ECHO REPLY
```

Darüber hinaus wird jedoch auch noch im Snort-Protokollverzeichnis (Default: `/var/log/snort`) ein Unterverzeichnis für diese Pakete erzeugt und dort entsprechend dem Protokoll und den Ports ebenfalls der Paket-Header protokolliert. In diesem Fall wird in `/var/log/snort/` ein Verzeichnis `127.0.0.1/` angelegt werden. In diesem Verzeichnis wird Snort eine Datei `ICMP_ECHO` erzeugen und dort diese Pakete anhängen. Im Falle einer TCP- oder UDP-Verbindung wird eine Datei angelegt werden, deren Name sich aus dem Protokoll und den verwendeten Ports ergibt, zum Beispiel: `TCP:62170-22`. Das bedeutet, dass für jede Netzwerkverbindung, die von Snort protokolliert wird, eine Datei erzeugt wird. Dies ist zum einen recht langsam und kann zum anderen im Falle eines Portscans zu extremen Verzeichnisgrößen führen. Dieser Modus sollte nicht in Produktionsumgebungen genutzt werden.

Rotation

Snort protokolliert so lange in diese Klartextdateien, wie das Betriebssystem Speicherplatz zur Verfügung stellt. Da moderne Linux-Distributionen nicht mehr an die Dateigrenze von 2 GByte gebunden sind, können diese Dateien sehr groß werden und auch sämtlichen freien Speicherplatz in Anspruch nehmen. Die Lösung für dieses Problem ist die Rotation der Protokolldateien. Bei dieser Rotation wird die aktuelle Protokolldatei umbenannt, eine neue leere Protokolldatei erzeugt und Snort mit dem Signal HUP dazu aufgefordert, die neue Protokolldatei zu nutzen.

Die meisten Distributionen liefern entsprechende Werkzeuge mit. Hier soll am Beispiel des Befehls `logrotate` der Red Hat Linux-Distribution die Konfiguration erklärt werden. Andere Distributionen verhalten sich ähnlich.

Der Befehl `logrotate` liest seine Konfiguration aus der Datei `/etc/logrotate.conf`. Bei Red Hat enthält diese Datei einen Verweis auf das Verzeichnis `/etc/logrotate.d` und liest so alle in diesem Verzeichnis befindlichen Konfigurationsdateien zusätzlich ein. Um `logrotate` nun anzuweisen, die zusätzlichen Snort-Protokolle zu rotieren, muss lediglich hier eine zusätzliche Konfigurationsdatei `snortlogs` erzeugt werden:

```
/var/log/snort/alert {
    weekly
    rotate 5
    compress
    notifempty
    missingok
    postrotate
        /usr/bin/killall -HUP snort
    }
}
```

Nun muss nur noch sichergestellt werden, dass `logrotate` regelmäßig aufgerufen wird. Dies erfolgt bei Red Hat Linux durch ein entsprechendes Script im Verzeichnis `/etc/cron.daily`. Dieses Script wird täglich aufgerufen und prüft, ob Protokolldateien rotiert werden müssen.

Wenn Ihre Distribution nicht über ein ähnliches Werkzeug verfügt, so befindet sich in der Snort-FAQ (<http://www.snort.org/docs/faq.html>) unter Punkt 6.8 ein Beispiel-Script, welches diese Aufgabe auch erfüllt.

Syslog-Protokollierung

Der zentrale Protokolldienst unter UNIX ist der `syslogd`. Dieser Dienst erhält seine Meldungen über einen so genannten Socket (`/dev/log`). Alle klassischen UNIX-Prozesse (nicht Apache, Squid oder Samba), die eine Protokollierung wünschen, senden ihre Meldungen an den Syslog. Der Syslog analysiert die Meldung und protokolliert sie in Abhängigkeit von der Quelle (`facility`) und der Priorität (`priority`) an unterschiedlichen Orten. Die Konfiguration des Syslog erfolgt in der Datei `/etc/syslog.conf`. Diese Datei besitzt folgende Syntax:

```
# facility.priority                                file
# Beispiel:
mail.*                                              /var/log/messages
# Protokollierung auf Geräten:
kern.*                                             /dev/console
# Protokollierung auf einem zentralen Syslogserver
local7.*                                           @snortlogserver
```

Das Snort-Output-Plug-In `alert_syslog` kann nun so konfiguriert werden, dass es eine bestimmte Quelle (*facility*) und Wertigkeit (*priority*) an den Syslog bei jeder Meldung übergibt.

Mögliche Quellen (*facility*): `LOG_AUTH`, `LOG_AUTHPRIV`, `LOG_DAEMON`, `LOG_LOCAL0` - `LOG_LOCAL7` und `LOG_USER`.

Verfügbare Prioritäten (*priority*): `LOG_EMERG`, `LOG_ALERT`, `LOG_CRIT`, `LOG_ERR`, `LOG_WARNING`, `LOG_NOTICE`, `LOG_INFO`, `LOG_DEBUG`.

Zusätzlich kann ab der Version 2.0 die Funktion des Output-Plug-Ins mit den folgenden Parametern modifiziert werden:

- `LOG_CONS`. Dieser Parameter führt die Protokollierung bei einem Fehler auf der Konsole durch.
- `LOG_NDELAY`. Dieser Parameter verzögert nicht das Öffnen des Kommunikationskanals zum Syslogd.
- `LOG_PERROR`. Dieser Parameter führt immer eine Protokollierung auf der Standardfehlerausgabe durch.
- `LOG_PID`. Hiermit wird auch immer die Prozessnummer mitprotokolliert.

Beispielkonfiguration:

```
output alert_syslog: LOG_LOCAL7 LOG_INFO LOG_PID
```

Auch bei einer Alert-Protokollierung über den Syslog sollte sichergestellt sein, dass eine Rotation der Protokolldatei durchgeführt wird. Einzelheiten sind im vorigen Abschnitt beschrieben.

WinPopUp-Alarmierung



Achtung:

Dieses Output-Plug-In steht seit der Version 2.1 nicht mehr zur Verfügung.

Bis Version 2.0 ist Snort in der Lage, über das Output-Plug-In `alert_smb` mit dem Kommando `smbclient` Mitteilungen an andere Microsoft Windows-Rechner zu senden, wenn auf diesen der Nachrichtendienst aktiviert wurde. Die Konfiguration des Output-Plug-Ins ist sehr einfach. Es benötigt lediglich eine Datei, in der die NET-BIOS-Namen aller zu alarmierenden Rechner aufgeführt sind.

```
output alert_smb: rechnerliste.txt
```

Wenn das Plug-In nicht zu funktionieren scheint, prüfen Sie zunächst, ob das Kommando `smbclient` tatsächlich installiert ist und sich im Suchpfad befindet. Bei einer Chroot-Installation muss sich das Kommando im Chroot-Verzeichnis befinden! Anschließend prüfen Sie bitte, ob Sie mit einem direkten Aufruf dieses Befehls ein WinPopUp senden können. Wenn dies nicht erfolgreich ist, prüfen Sie die Einstellungen des Nachrichtendienstes auf Ihrem Windows-Rechner. Seien Sie sich bewusst, dass Snort durchaus sehr viele Meldungen besonders in einer Angriffssituation erzeugt. Dies kann zu einem Denial of Service auf dem Windows-Rechner führen, da er mit PopUp-Meldungen überschwemmt wird.

```
echo "Linux WinPopUp" | smbclient -M NETBIOS-Name
```

Protokollierung über einen UNIX-Socket

Das Output-Plug-In `alert_unixsock` ermöglicht eine Protokollierung über einen UNIX-Socket an ein weiteres Programm.

Dies ermöglicht es einem weiteren Programm, die Ausgaben von Snort direkt online zu prozessieren. Bisher ist kein Programm bekannt, welches dieses Output-Plug-In nutzt.

Beispiel-Code für ein derartiges Programm ist in der Datei `doc/README.UNSOCK` enthalten.

Binärprotokollierung in eine Datei

Dieses Output-Plug-In (`log_tcpdump`) ermöglicht die Protokollierung des gesamten Paketes in der angegebenen Datei. Dieses Plug-In arbeitet sehr schnell, da es im Gegensatz zum `alert_full`-Plug-In nicht verschiedenste Dateien öffnen und auf externe Befehle oder Dienste warten muss. Das Format der resultierenden Protokolldatei ist das `libpcap`-Format. Snort ist, wie auch `tcpdump`, `ethereal` und weitere Befehle, in der Lage, diese Datei wieder einzulesen.

```
output log_tcpdump: binary.log
```

Sinnvollerweise wird in einer Umgebung, in der die Geschwindigkeit von Snort essentiell ist, dieses Plug-In genutzt. Die Analyse der protokollierten Pakete kann dann später offline durch Einlesen und Klartextausgabe dieser Datei erfolgen.

XML-Protokollierung

Es existieren zwei verschiedene Output-Plug-Ins, die die Informationen als XML aufbereiten. Dabei handelt es sich um die Plug-Ins `xml` und `idmef`.

xml-Plug-In

Das `xml`-Plug-In verwendet die Sprache Simple Network Markup Language (SNML, auch Snort Markup Language). Die DTD steht unter <http://www.cert.org/kb/snort.xml/>

snml-0.3.dtd zum Download bereit. Das Plug-In wurde am CERT Coordination Center als Teil des AIRCert-Projektes entwickelt.

Dieses Plug-In ist kein fester Bestandteil von Snort, sondern kann als Patch von der AIRCert-Homepage geladen werden (<http://aircert.sourceforge.net/>).



Achtung:

Dieses Plug-In ist nicht mit den aktuellen Snort-Versionen funktionsfähig.

Das *xml*-Plug-In kann sowohl von *alert*- als auch von *log*-Regeln genutzt werden. Des Weiteren können Parameter angegeben werden, die das weitere Verhalten definieren:

```
output xml: [log | alert] optionen
```

Folgende Optionen stehen zur Verfügung:

- **file.** Lokale Protokolldatei oder Script auf dem entfernten Rechner. Wenn ein Protokoll (nächste Option) angegeben wird, wird dieses Script als CGI auf dem entfernten Rechner in einer HTTP POST-Operation aufgerufen und die Daten werden übertragen.
- **protocol.**
 - **http.** HTTP POST unter Aufruf eines CGI-Scripts (file muss angegeben werden).
 - **https.** HTTPS POST (file, cert und key müssen angegeben werden)
 - **tcp.** Einfache TCP-Verbindung. Ein Server muss auf der anderen Seite gestartet werden (z.B. Netcat) (port ist erforderlich).
 - **iap.** Intrusion Alert Protocol – noch nicht implementiert.
- **host.** Protokollserver
- **port.** (Default: http-80, https-443, tcp-9000, iap-9000)
- **cert.** Client x509-Zertifikat in pem-Format
- **key.** Privater Schlüssel des Clients in pem-Format
- **ca.** Zertifikat der Zertifikatsautorität, um das Zertifikat des Servers zu validieren in pem-Format
- **server.** Datei, in der die Titel der Serverzertifikate abgelegt werden
- **sanitize.** Das angegebene Netzwerk/Netzmaske wird anonymisiert.

- `encoding`. Kodierung des binären Paketinhaltes (hex, base64, ascii)
- `detail`. *full/fast*. Full protokolliert alle Informationen über das Paket inklusive des Inhalts. Fast protokolliert lediglich ein Minimum an Daten.

Die Option `encoding` bedarf einer genaueren Betrachtung. Es stehen drei verschiedene Kodierungen zur Verfügung. Hex benötigt den doppelten Platz zur Speicherung der Daten, bietet jedoch gute Suchfunktionen. Base64 benötigt den 1,4-fachen Speicherplatz, lässt sich dafür aber kaum durchsuchen. Ascii benötigt keinen zusätzlichen Speicherplatz, führt dabei aber zu Datenverlust, da alle nicht druckbaren Zeichen durch einen ».« dargestellt werden. Es bietet dafür aber eine sehr gute Lesbarkeit der enthaltenen Texte.

Beispielkonfiguration:

```
output xml: log protocol=https host=snortlog.example.com file=snort.cgi \
           cert=sensor1_cert.pem key=sensor1_key.pem ca=ca_cert.pem \
           server=serverlist.txt
```

IDMEF-Plug-In

Das IDMEF-Plug-In wurde wie SPADE von Silicondefense (<http://www.silicondefense.com/idwg/snort-idmef/>) geschrieben. Seit die Firma Silicondefense nicht mehr existiert, wurde das Plug-In von Sandro Poppi auf <http://sourceforge.net/projects/snort-idmef/> weiterentwickelt. Die aktuelle Version ist verfügbar für die Versionen Snort 2.0.x und 2.1.x. Es gibt die Protokollmeldungen im Intrusion Detection Message Exchange-Format aus. IDMEF wurde von der Arbeitsgruppe Intrusion Detection Working Group (IDWG) der Internet Engineering Task Force (IETF) entwickelt. Dies ist der Versuch, eine allgemeine Sprache zur Behandlung von NIDS-Ereignissen zu entwickeln.



Achtung:

Der Snort-Quelltext muss für die Installation dieses Plug-Ins gepatcht werden.

Das Plug-In wird ähnlich den anderen Output-Plug-Ins aktiviert:

```
output idmef: $HOME_NET key=value ...
```

`$HOME_NET` definiert das eigene Netzwerk in der Notation-Netzwerk/CIDR-Netzmaske, z.B. 192.168.111.0/24. Folgende Parameter müssen zusätzlich angegeben werden:

- `logto`. Protokolldatei

- `dtd`. Ort der DTD-Datei
- `analyzer_id`. Eindeutige Identifikation dieses Sensors

Folgende Parameter sind optional:

- `category`. Netzwerkkategorie (`unknown`, `ads`, `afs`, `coda`, `dfs`, `dns`, `kerberos`, `nds`, `nis`, `nisplus`, `nt`, `wfw`)
- `name`. Voll qualifizierter Name dieses Sensors in der Domäne
- `location`. Physikalischer Ort des Sensors
- `address`. Netzwerkadresse dieses Sensors
- `netmask`. Dazugehörige Netzmaske
- `address_cat`. Art der Adresse (`unknown`, `atm`, `e-mail`, `lotus-notes`, `mac`, `sna`, `vm`, `ipv4-addr`, `ipv4-addr-hex`, `ipv4-net`, `ipv4-net-mask`, `ipv6-addr`, `ipv6-net`, `ipv6-net-mask`)
- `homenet_cat`. Kategorie des eigenen Netzes
- `homenet_loc`. Physikalischer Ort des eigenen Netzes
- `default`. Format der Meldungen vom Typ *default* (`disable`, `hex`, `ascii`, `base64`)
- `web`. Format der Meldungen vom Typ *web*
- `overflow`. Format der Meldungen vom Typ *overflow*
- `indent`. Einrücken der XML-Meldungen (`true` | `false`)
- `alert_id`. Datei, in der die Nummer der letzten Meldung zwischengespeichert wird

Um nun das IDMEF-Plug-In nutzen zu können, muss es zunächst aktiviert werden. Anschließend müssen die Regeln entsprechend angepasst werden. Regeln, die nun dieses Plug-In nutzen sollen, benötigen ein zusätzliches Attribut `idmef: default|web|overflow`:

```
output idmef: $HOME_NET logto=/var/log/snort/idmef_alert analyzer_id=Sensor1
↳ dtd=idmef-message.dtd category=dns location=Steinfurt
alert tcp any any -> any 80 (msg:"Directory Traversal"; flags: AP; content:
↳ "../";idmef: web;)
```

Diese Regel erzeugt folgende Ausgabe:

```
<IDMEF-Message version="0.1">
<Alert alertid="329440" impact="unknown" version="1">
<Time>
<ntpstamp>0x3a2da04c.0x0</ntpstamp>
<date>2002-05-11</date>
<time>16:41:30</time>
</Time>
<Analyzer ident="Sensor1">
<Node category="dns">
<location>Steinfurt</location>
```

```

</Node>
</Analyzer>
<Classification origin="vendor-specific">
<name>Directory Traversal</name>
</Classification>
<Source spoofed="unknown">
<Node>
<Address category="ipv4-addr">
<address>3.3.3.3</address>
</Address>
</Node>
</Source>
<Target decoy="unknown">
<Node category="dns">
<location>Steinfurt</location>
<Address category="ipv4-addr">
<address>192.168.111.50</address>
</Address>
</Node>
<Service ident="0">
<dport>80</dport>
<sport>1397</sport>
</Service>
</Target>
<AdditionalData meaning="Packet Payload" type="string">GET ../../etc/passwd
↳ </AdditionalData>
</Alert>
</IDMEF-Message>

```

Protokollierung in einer Datenbank

Snort ist in der Lage, die Protokollierung direkt in einer relationalen Datenbank durchzuführen. Dies ermöglicht eine Analyse der Protokolle in der Datenbank in Echtzeit. Das `database-Output` Plug-In unterstützt PostgreSQL-, MySQL-, unix-ODBC-, MS SQL- und Oracle-Datenbanken.

Snort ist nicht in der Lage, die Datenbank selbst zu erzeugen, jedoch befinden sich im Snort-Quelltextarchiv im `contrib`-Verzeichnis entsprechende `create_database`-Dateien, die diese Aufgabe übernehmen. Hier soll die Datenbankerstellung am Beispiel von MySQL, der am häufigsten für die Aufgabe verwendeten Datenbank, durchgespielt werden.

Stellen Sie zunächst sicher, dass die MySQL-Datenbank auf Ihrem System installiert ist und gestartet wurde. Wenn Ihre Distribution die MySQL-Datenbank nicht mitliefert, können Sie die neueste Version bei <http://www.mysql.org> herunterladen. Dort finden Sie üblicherweise auch RPM-Pakete, die die Installation vereinfachen. Anschließend prüfen Sie bitte, ob Snort mit MySQL-Unterstützung übersetzt wurde.

```
[root@kermit root]# ldd /usr/sbin/snort
libz.so.1 => /usr/lib/libz.so.1 (0x4002e000)
libm.so.6 => /lib/i686/libm.so.6 (0x4003d000)
libnsl.so.1 => /lib/libnsl.so.1 (0x4005f000)
libmysqlclient.so.10 => /usr/lib/mysql/libmysqlclient.so.10 (0x40074000)
libc.so.6 => /lib/i686/libc.so.6 (0x42000000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x400a9000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Wenn Snort keine Unterstützung für MySQL aufweist, übersetzen Sie Snort bitte mit der `configure`-Option `--with-mysql` erneut. Erzeugen Sie nun eine Datenbank für Snort in MySQL. Geben Sie nach Aufforderung das Kennwort von root für die Datenbank ein.

```
[root@kermit root]# echo "CREATE DATABASE snortdb;" | mysql -u root -p
```

Erzeugen Sie nun einen Benutzer, in dessen Kontext Snort die Datenbank füllen kann:

```
echo "grant INSERT,SELECT On snortdb.* to snortuser@localhost identified\
by 'geheim';" | mysql -u root -p
```

Erzeugen Sie anschließend das Datenbankschema. Hierzu existiert im `contrib`-Verzeichnis des Snort-Quelltextpaketes die Datei `create_mysql`:

```
mysql snortdb -u root -p < contrib/create_mysql
```

Nun können Sie das Output-Plug-In in Snort definieren. Hierbei geben Sie den Protokolltyp (`log` oder `alert`), den Datenbanktyp (`mysql`, `postgres`, `odbc`, `mssql` oder `oracle`), den Datenbanknamen (`dbname=`) und weitere optionale Parameter an. Folgende Parameter sind erlaubt:

- `host`. Rechner, auf dem der Datenbankserver läuft
- `port`. Port, auf dem der Datenbankserver zu erreichen ist
- `user`. Anmeldung erfolgt durch diesen Benutzer
- `password`. Anmeldung erfolgt mit diesem Kennwort
- `sensor_name`. Name des Sensors (wird sonst automatisch generiert)
- `encoding`. `hex` (default), `ascii`, `base64`
- `detail`. `full`, `fast`

Zur Diskussion der Optionen `encoding` und `detail` verweise ich auf das `XML-Plug-In`.

Die Beispielkonfiguration für obige MySQL-Datenbank sieht folgendermaßen aus:

```
output database: log, mysql, dbname=snortdb user=snortuser password=geheim \
host=localhost port=3306 encoding=hex detail=full
```

Ein Problem, welches leider immer wieder bei der Konfiguration von Snort vergessen wird, basiert auf der Tatsache, dass Snort kein multithreaded Programm ist. Das bedeutet, dass Snort immer nur eine Tätigkeit ausführen kann. Snort kann erst dann das nächste Paket analysieren, wenn das letzte Paket erfolgreich analysiert und im Zweifelsfall protokolliert wurde. Bei Verwendung des *database*-Plug-Ins muss Snort auf die erfolgreiche Bestätigung der Protokollierung durch die Datenbank warten. Während dieser Zeit können keine weiteren Pakete analysiert werden. In einem späteren Kapitel (Snort im Unternehmenseinsatz) wird diese Problematik wieder besprochen und werden Auswege aufgezeigt.

CSV – Kommaseparierte Klartextprotokollierung

Das CSV-Plug-In ist ein reines Alert-Plug-In. Es erlaubt die Protokollierung von Alert Ereignissen in einem kommaseparierten Format, welches sich für den Import in Datenbanken und Tabellenkalkulationen eignet.

Die Konfiguration des CSV-Plug-Ins ist recht einfach. Die Angabe eines Dateinamens und der zu protokollierenden Informationen genügt. Snort wird dann die geforderten Informationen kommasepariert in der angegebenen Reihenfolge in der Datei protokollieren:

```
output CSV: /var/log/snort/alert.csv timestamp, msg
```

Folgende Informationen können protokolliert werden:

- timestamp. Uhrzeit
- msg. Alarmmeldung
- proto. Protokoll
- src. Absender-IP-Adresse
- dst. Empfänger-IP-Adresse
- srcport, dstport. (entsprechende Ports)
- ethsrc, ethdst. (MAC-Adressen des Absenders und Empfängers)
- ethlen. Länge des Ethernet-Rahmens
- tcpflags. Gesetzte TCP-Flags
- tcpseq. TCP-Sequenznummer
- tcpack. TCP-Acknowledgenummer
- tcplen. Länge des TCP-Segments
- tcpwindow. Größe des TCP-Windows
- ttl. Time To Live
- tos. Type of Service
- id. IP-Fragment-Identifikationsnummer
- dgm1en. Länge des UDP-Datagramms
- iplen. Länge des IP-Paketes

- `icmptype`. ICMP-Typ (siehe Anhang)
- `icmpcode`. ICMP-Code (siehe Anhang)
- `icmpid`. ICMP-Identifikationsnummer
- `icmpseq`. ICMP-Sequenznummer

Unified Logging

Das `unified`-Plug-In tritt mit dem Anspruch an, das schnellste Output-Plug-In für Snort zu sein. Hierbei protokolliert das Plug-In in zwei Dateien: eine Alert-Datei und eine Paketprotokoll-Datei. Beide Dateien besitzen ein binäres Format. Das bedeutet, sie sind nicht direkt lesbar. Hierfür existieren die Werkzeuge *Barnyard* und *Mudpit*, welche dieses Format lesen und in unterschiedlichen Formaten wieder ausgeben können. Das Unified-Format wird das Format der Wahl in Umgebungen werden, die auf hohe Performanz angewiesen sind. Snort kann in diesen Umgebungen ohne Verzögerung alle Pakete analysieren, während *Barnyard* oder *Mudpit*unabhängig davon die Formatierung der Ausgabe übernimmt.

Die Konfiguration des Output-Plug-Ins ist recht unproblematisch:

```
output alert_unified: /var/log/snort/unified_alert
output log_unified: /var/log/snort/unified_log
```

Snort wird automatisch beim Start eine Zeitmarke in Form von `MonatTag@StundeMinute-` vor den Dateinamen hängen, um die verschiedenen Protokolle auseinanderzuhalten.

SNMP Traps

Snort ist in der Lage, Informationen mit SNMP über das `trap_snmp`-Output-Plug-In zu versenden. Hierzu verwendet Snort SNMP Version 2. Snort verwendet hierzu eine Management Information Base (MIB), die in den Dateien *SnortCommonMIB.txt* und *SnortIDAlertMIB.txt* abgelegt ist. Dieses Plug-In wird nicht unter der GPL, sondern unter der BSD-Lizenz vertrieben. Das Plug-In war bis Snort 1.9 dauerhafter Bestandteil von Snort und wird ab Snort 2.0 unter folgender Adresse vertrieben: <http://www.cysol.co.jp/contrib/snortsnmp/index.shtml>.

Das Plug-In erwartet die folgenden Angaben:

- `alert`. Bisher können nur alerts protokolliert werden.
- `SensorID`. Identifikation des Sensors
- `trap|inform`. Verwendung von SNMP Traps oder SNMP Informs
- `SNMP-Optionen`. `-v 2c` SNMPv2c Community, `-p` Port
- `Sink`. Trap-Empfänger
- `Community`. Community-Zeichenkette

Ein Beispielaufruf des Plug-Ins:

```
output trap_snmp: alert, Sensor1, trap, 192.168.111.50, private
```

Dieses Plug-In erlaubt die Integration von Snort in Netzwerkmanagement-Software wie HP Openview oder Scotty/Tkined. Diese erlauben häufig eine feinfühlig Alarmierung zum Beispiel mit SMS oder Pager.

log_null

Häufig gibt es den Bedarf, Regeln zu erzeugen, welche zwar einen Alert erzeugen, jedoch keine Protokollierung des Paketes. Für diese Fälle gibt es die Kommandozeilenoption `-N` und das Output-Plug-In `log_null`. Dies wird am sinnvollsten in einem eigenen Regeltyp eingesetzt:

```
ruletype info {
    type alert
    output alert_fast: snort.info
    output log_null
}
```

Thresholding und Suppression

Ab der Version 2.1 unterstützt Snort ein Thresholding und Event-Suppression. Das Thresholding erlaubt es dem Anwender, zu entscheiden, wie häufig eine Regel zutreffen darf oder muss, um eine Alarmmeldung zu erzeugen. Mit der Event-Suppression kann der Anwender bestimmte Regeln für bestimmte IP-Adressen unterdrücken.

Thresholding

Snort unterstützt drei verschiedene Formen des Thresholding:

- `limit`. Hierbei erzeugt eine Regel nur eine bestimmte Anzahl von Alarmmeldungen in einem Zeitraum. Selbst wenn die Regel häufiger zutrifft, werden keine Alarme mehr erzeugt. Beispiel: Snort gibt die ersten fünf Alarmmeldungen alle 30 Sekunden aus.
- `threshold`. Hierbei wird nur jede n-te Alarmmeldung in einem Zeitraum ausgegeben. Alle weiteren Alarmmeldungen einer Regel werden ignoriert. Beispiel: Snort gibt die 5., 10. und 15. Alarmmeldung in einem Zeitraum von 30 Sekunden aus.
- `both`. Dies erlaubt die Alarmierung durch die n-te Alarmmeldung in einem Zeitraum. Alle weiteren Alarmmeldungen in demselben Zeitraum werden ignoriert. Beispiel: Snort gibt nur die 5. Alarmmeldung in einem Zeitraum von 30 Sekunden aus. Die 10. und 15. Meldung wird auch ignoriert.

Die Threshold-Konfiguration kann sowohl in einer Regel angegeben werden als auch für sich alleine stehen. In einer Regel wird das folgende Format verwendet:

threshold: type <limit|threshold|both>, track *by_src,by_dst*, count *nummer*,
 ↳ seconds *sekunden*;

Hierbei haben die Argumente die folgende Bedeutung:

- type. Hier wird die Threshold-Form gewählt.
- track. Hier definieren Sie, ob die Quell-IP-Adresse oder die Ziel-IP-Adresse für das Thresholding verfolgt werden soll.
- count. Hiermit geben Sie die Häufigkeit an.
- seconds. Dieser Parameter spezifiziert den Zeitraum.

Sie können die Threshold-Konfiguration aber auch in einer eigenen Datei *threshold.conf* durchführen, die Sie dann mit dem `include`-Parameter in die *snort.conf* einlesen. Dort verwenden Sie dann die folgende Syntax:

```
threshold gen_id gen-id, sig_id sid, type limit|threshold|both,
↳ track by_src,by_dst, count nummer, seconds sekunden;
```

Die zusätzlichen Angaben definieren:

- *gen_id*. Die Nummer des Generators. Regeln haben die Generatoridentifikationsnummer 1 und zum Beispiel der BackOrifice-Präprozessor die Nummer 105. Sämtliche Generatoridentifikationsnummern finden Sie in der Datei *etc/gen-msg.map*. Möchten Sie ein generelles Thresholding für alle Generatoren definieren, so nutzen Sie hier die Null (0).
- *sid_id*. Dies ist die Signatur-ID. Jede Regel hat eine bestimmte Identifikationsnummer. Damit wirkt das Thresholding nur auf diese eine Regel. Sollen alle Signaturen vom Thresholding betroffen sein, nutzen Sie hier die Null (0).

Ein Beispiel für ein globales Thresholding:

```
threshold gen-id 0, sig-id 0, type limit, track by_src, count 5, seconds 15
```

Mithilfe des Thresholds können auch neue Regeln erzeugt werden, die zum Beispiel nicht jeden Anmeldeversuch an einem Netzwerkdienst melden, sondern dies nur melden, wenn ein sehr häufiger Zugriff in kurzer Zeit erfolgt. So kann mit folgender Regel ein Brute-Force-Zugriff auf einen IMAP-Server erkannt werden:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 143 (msg:"IMAP login brute force
↳ attempt"; flow:to_server,established; content:"LOGIN"; nocase;
↳ threshold:type threshold, track by_src, count 30, seconds 30;
↳ classtype:suspicious-login; sid:2273; rev:2;)
```

Sobald eine Quell-IP-Adresse mindestens 30-mal in 30 Sekunden versucht, eine Anmeldung auf einem IMAP-Server durchzuführen, löst diese Regel einen Alarm aus.

Event-Suppression

Die Event-Suppression erlaubt die Unterdrückung von Alarmmeldungen in Abhängigkeit der beteiligten IP-Adressen, ohne die Regel aus dem Regelsatz zu entfernen. Hiermit kann sie eine ähnliche Aufgabe übernehmen wie die `pass`-Regeln.

Die Event-Suppression kann nicht direkt in der Regel konfiguriert werden, sondern nur alleine für sich (Stand-alone). Dabei verwendet sie das folgende Format:

```
suppress gen_id gen-id, sid_id sid, track by_src|by_dst, ip ip[/mask]
```

Die Angaben `gen_id`, `sid_id` und `track` haben die gleiche Bedeutung wie beim Thresholding. Mit dem Parameter `ip` können Sie die IP-Adresse oder das Netzwerk angeben, auf das die Event-Suppression angewendet werden soll. Wenn Sie zum Beispiel ein Nessus-System für regelmäßige Security-Audits verwenden und nicht von Alarmmeldungen überschwemmt werden möchten, die von diesem System erzeugt werden, können Sie folgende Zeile nutzen:

```
suppress gen_id 0, sid_id 0, track by_src, ip <nessus-ip>
```

MD5-Verkettung/GnuPG-Signatur

Wenn die Protokolldateien als späteres Beweismaterial genutzt werden sollen, sollte sichergestellt werden, dass diese Protokolle später nicht modifiziert werden können. Dies ist am einfachsten sicherzustellen, indem die Protokolle mit GPG signiert und mit MD5-Prüfsummen verkettet werden.

Die Signatur der Dateien garantiert die Echtheit der Dateien. Andere Personen können die Dateien nicht modifizieren. Zunächst müssen Sie sich einen GnuPG-Schlüssel erzeugen:

```
[root@kermit root]# gpg --gen-key
gpg (GnuPG) 1.0.6; Copyright (C) 2001 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.
```

Bitte wählen Sie, welche Art von Schlüssel Sie möchten:

- (1) DSA und ElGamal (voreingestellt)
- (2) DSA (nur signieren/beglaubigen)
- (4) ElGamal (signieren/beglaubigen und verschlüsseln)

Ihre Auswahl? 1

Der DSA Schlüssel wird 1024 Bit haben.

Es wird ein neues ELG-E Schlüsselpaar erzeugt.

kleinste Schlüssellänge ist 768 Bit

standard Schlüssellänge ist 1024 Bit

grösste sinnvolle Schlüssellänge ist 2048 Bit

Welche Schlüssellänge wünschen Sie? (1024) 2048

Brauchen Sie wirklich einen derartig langen Schlüssel? j

Die verlangte Schlüssellänge beträgt 2048 Bit

Nun kann später immer die Originalität der Dateien überprüft werden:

```
[root@kermit root]# gpg --verify /var/log/snort/alert.sig /var/log/snort/alert
gpg: Unterschrift vom Die 14 Mai 2002 20:48:09 CEST, DSA Schlüssel ID C1E4766C
gpg: Korrekte Unterschrift von "Ralf Spenneberg (Protokoll Signatur)
<ralf@spenneberg.de>"
```

Jedoch besteht noch die Möglichkeit, dass einzelne Dateien modifiziert und anschließend neu signiert werden. Um dies zu unterbinden, können die Dateien mit einer Prüfsumme verkettet werden. Dazu erzeugen Sie zunächst eine MD5-Prüfsumme aus Zufallszahlen, die anschließend mit der ersten Protokolldatei verkettet eine neue Prüfsumme ergibt. Dies wird für die nächste Protokolldatei wiederholt:

```
[root@kermit root]# dd if=/dev/random count=1024 bs=1 | md5sum >
↳ initialisierungsvektor
1024+0 Records ein
1024+0 Records aus
[root@kermit root]# cat /var/log/snort/alert initialisierungsvektor | md5sum >
↳ alert.md5
[root@kermit root]# cat /var/log/snort/alert.new alert.md5 | md5sum >
↳ alert.new.md5
```

Tuning

Dieses Kapitel zeigt die Performanzprobleme beim Einsatz von Snort auf und gibt Hinweise zur optimalen Konfiguration von Snort. Hauptsächlich die folgenden Aufgaben stellen die Nadelöhre bei Geschwindigkeitsproblemen dar:

- Sammeln der Pakete mit *libpcap*. Hier kann der Benutzer durch den Einsatz der modifizierten *mmap-libpcap* (siehe Abschnitt »*libpcap*-Bibliothek mit Ringpuffer« auf S. 246, <http://public.lanl.gov/cpw/>) eine hohe Geschwindigkeitssteigerung erreichen. Zusätzlich kann der Einsatz von Berkeley-Paketfiltern eine Reduktion der aufzunehmenden Pakete erzielen.
- Interne Speicherverwaltung. Auch hier kann der Benutzer keinen Einfluss nehmen.
- Mustererkennung. Hier wird Snort 2.0 einen neuen Mechanismus verwenden, der laut Marty Roesch eine bis zu fünffache Geschwindigkeitssteigerung erzielt.
- Anordnung der Regeln und der Attribute in der Regel. Bei Kenntnis der inneren Funktion kann durch eine intelligente Anordnung die Geschwindigkeit gesteigert werden.
- Prüfsummenverifizierung. Snort überprüft für jedes Paket sämtliche im Paket befindlichen Prüfsummen. Existieren bereits Netzwerkgeräte wie Router oder Swiches, die diese Aufgabe erfüllen, kann diese Funktion deaktiviert werden (siehe `checksum_mode`).
- Präprozessoren und Output-Prozessoren bearbeiten die Pakete vor bzw. nach der Detektion. Da Snort kein multithreaded Programm (bis jetzt) ist, kann das nächste

Paket erst bearbeitet werden, wenn das letzte vollständig bearbeitet wurde. Insbesondere komplizierte Output-Plug-Ins können Snort bremsen. Auf der anderen Seite kann jedoch mit den zustandsorientierten Präprozessoren Snort in gewissen Umständen stark beschleunigt werden. Angriffswerkzeuge wie `stick` oder `snort` führen nicht zu einer Überlastung von Snort. Diese Werkzeuge senden zusammenhangslos IP-Pakete. Diese Pakete gehören nicht zu aufgebauten Verbindungen. Ein zustandsloses NIDS muss jedes Paket analysieren und protokollieren. Ist der `stream4`-Präprozessor in Snort aktiviert, so filtert dieser bereits derartige Pakete aus.

Der Linux-Kernel 2.4 bietet die Möglichkeit, mit dem `ip_conntrack.o`-Modul sämtliche Pakete zu defragmentieren. Sobald dieses Modul geladen wurde, werden alle Pakete, die als Fragment den Rechner erreichen, vor der Analyse durch die Paketfilterrichtlinien defragmentiert. Snort greift jedoch über die `libpcap`-Bibliothek auf diese Pakete zu und sieht weiterhin die Fragmente. Ein Abschalten der Defragmentierung ist also nicht möglich.

Output-Tuning

Das Tuning der Protokollierung durch Snort stellt wahrscheinlich eine der effektivsten Formen in vielen Installationen dar. Viele Benutzer werden geblendet durch die Vielzahl der verfügbaren Output-Plug-Ins und aktivieren mehrere dieser Plug-Ins. Um grafische Analysewerkzeuge einsetzen zu können (Besprechung folgt im nächsten Kapitel), wird üblicherweise auch eine Datenbank als Protokollziel gewählt. Viele Anwender vergessen jedoch, dass diese Protokollierung sehr viel Zeit kostet, da momentan für jedes einzelne Paket eine (Netzwerk-)Verbindung mit der Datenbank geöffnet wird, das neue Objekt in die Datenbank geschrieben und committet und anschließend die Verbindung geschlossen wird. Es existieren Dokumentationen im Internet, in denen zusätzlich die Übertragung des Paketes in einem SSL-Tunnel empfohlen und beschrieben wird. Dies erzeugt einen zusätzlichen Overhead und Verzögerungen.

Ein weiterer häufiger Fehler ist die Protokollierung auf dem Bildschirm. Es ist zwar recht nett, wilde Zeilen über den Bildschirm fliegen zu sehen, jedoch darf nicht vergessen werden, dass üblicherweise unter Linux (auch andere UNIXe) der Bildschirm als virtuelles Terminal emuliert wird. Dieses Terminal hat üblicherweise eine Geschwindigkeit in der Größenordnung von 38.400 Baud. Das bedeutet, dass Informationen mit 38,4 Kbit/s geschrieben werden können. Es ist leicht einzusehen, dass es hier zu Problemen kommen kann, wenn Snort ein Netzwerk mit 10 Mbit/s (10.000 Kbit/s) oder gar 100 Mbit/s (100.000 Kbit/s) beobachtet. So angenehm diese Ausgabeformate auch sind, sie brauchen sehr viel Zeit in ihrer Erzeugung.

Sinnvollerweise wird auf dem eigentlichen Sensor lediglich im Binärformat oder im Unified-Format protokolliert. Snort ist dann in der Lage, ohne Verzug die notwendigen Informationen zu protokollieren. Für die Ausgabe werden diese Dateien in regelmäßigen Abständen auf einen anderen Rechner transferiert und dort von einem zweiten Snortprozess (`snort -r`) oder Barnyard analysiert. Im nächsten Kapitel wird ein derartiges Szenario beschrieben und implementiert.

Wird eine Echtzeit-Alarmierung gewünscht, so sollte zusätzlich auf dem Sensor ein *alert_fast*-Plug-In die entsprechenden Ausgaben in einer Datei erzeugen. Ein weiteres Protokollanalysewerkzeug kann die Datei überwachen und zum Beispiel E-Mails oder WinPopUps versenden. In diesem Fall erfolgt das aber unabhängig vom Snort-Prozess.

Optimierung der Präprozessoren

Bei der Optimierung der Präprozessoren kann im Allgemeinen festgestellt werden, dass Snort umso schneller arbeitet, je weniger Präprozessoren aktiviert sind. Daher sollte darauf geachtet werden, dass nur die Präprozessoren aktiviert werden, deren Funktionalität benötigt wird.

Zwei Präprozessoren lassen sich jedoch darüber hinaus optimieren: *frag2* und *stream4*.

frag2 ist der Defragmentierungspräprozessor. Er ist wesentlich schneller als der alte defrag-Präprozessor. Er erwartet als Option die Angabe des verwendbaren Speichers und einen Timeout-Wert. Diese Werte erlauben die Optimierung. Der Timeout-Wert gibt die Zeit an, die *frag2* auf sämtliche Fragmente eines Paketes wartet. Je größer dieser Wert ist, desto mehr Speicher benötigt *frag2*. Defaultwerte sind 4 Mbyte und 60 Sekunden. Steht auf dem Sensor genügend Speicher zur Verfügung, so sind sinnvollerweise 16 Mbyte oder mehr zu wählen. In Abhängigkeit von der Geschwindigkeit des zu beobachtenden Netzwerkes (der Internetanbindung) kann der Timeout-Wert auf 30 Sekunden reduziert werden.

stream4 erweitert Snort um die Fähigkeit, TCP-Pakete Verbindungen zuzuordnen. Er ersetzt *stream* und ist in der Lage, für mehrere tausend Verbindungen gleichzeitig die Reassemblierung und für bis zu 64.000 Verbindungen die zustandsorientierte Überwachung durchzuführen. Auch dieser Präprozessor benötigt die Angabe eines Timeouts und des zu verwendenden Speichers. Der Timeout ist bei Snort als Standardwert mit 30 Sekunden sehr klein gewählt. Dieser Wert führt dazu, dass Snort eine Verbindung aus der Zustandstabelle entfernt, wenn in den letzten 30 Sekunden kein weiteres Paket dieser Verbindung gesehen wurde. Der Speicher ist als Standard auf 8 Mbyte eingestellt. Auch dieser Wert sollte erhöht werden, wenn genügend Speicher zur Verfügung steht. Soll der *stream4*-Präprozessor nun eingesetzt werden, um die Detektionsmaschine zu entlasten, so muss Snort mit der Option `-z est` gestartet werden oder die Konfigurationsdatei muss einen Eintrag `config stateful est` tragen.

Der *stream4*-Präprozessor besitzt noch eine andere Option, die die Geschwindigkeit beeinflussen kann. Hierbei handelt es sich um die Einstellung, für welche Protokolle die in den Paketen hinterlegten Prüfsummen überprüft werden sollen. Häufig überprüfen bereits andere Netzwerkgeräte wie Router oder Switches diese Prüfsummen und verwerfen bereits die Pakete, die fehlerhafte Prüfsummen aufweisen. In diesem Fall kommen derartige Pakete nicht mehr bei Snort an. Dann macht es auch keinen Sinn, bei den bekannt korrekten Paketen die Prüfsummen erneut zu überprüfen. Dieser Vorgang kann komplett oder für einzelne Protokolle abgeschaltet werden und es kann die entsprechende Zeit zur Paketanalyse genutzt werden:

```
# entweder noip notcp noudp noicmp oder none  
config checksum_mode: none
```

Die Reassemblierung benötigt sehr viel Speicher (und Zeit). In einer Hochgeschwindigkeitsumgebung sollte die Reassemblierung auf dem Sensor deaktiviert werden. Dies erzeugt aber möglicherweise einen höheren Anteil an falsch-positiven Meldungen.

Regel-Tuning

Um die Regeln möglichst gut an das zu überwachende Netzwerk anzupassen, sind genaue Informationen über dieses erforderlich. Der Regelsatz, der mit Snort ausgeliefert wird, ist für die meisten Anwendungen zu groß und enthält sehr viele Regeln, deren Anwendung entweder nicht nötig ist (weil kein Webserver existiert) oder viele falsche Alarmierungen verursacht (weil zum Beispiel Netzwerküberwachungswerkzeuge ping zur Überwachung von Netzwerkgeräten einsetzen und Ping-Pakete von Snort gemeldet werden).

Um eine Optimierung des Regelsatzes zu erreichen, sollten zunächst nur Regeln, deren Anwendung sinnvoll ist, aktiviert werden. Weitere Optimierungen werden in den nächsten beiden Abschnitten besprochen.

Attribut-Reihenfolge

Snortregeln bestehen aus einem Rumpf und weiteren Optionen. Der Rumpf der Regel wird immer zuerst überprüft. Nur wenn die Angaben im Rumpf mit dem Paket übereinstimmen, werden die Attribute getestet. Daher sollten bereits möglichst viele Informationen im Rumpf spezifiziert werden. Der IMAP-Server-Bufferoverflow ist zum Beispiel nur gefährlich, wenn er an Port 143 gerichtet ist.

Anschließend werden die Attribute in der Reihenfolge überprüft, in der sie spezifiziert werden. Die einzige Ausnahme von dieser Regel stellen `content`, `uricontent` und die diskreten Attribute (s.u.) dar. Diese werden seit Snort 2.0 von der Hi-Performance-Multi-Rule-Inspection-Engine zu Beginn gleichzeitig getestet. Dabei sucht diese Snort-Funktion nach allen Vorkommen der Suchmuster in dem Paket und stellt diese in eine Warteschleife. Alle weiteren Attribute werden anschließend in ihrer Reihenfolge getestet.

Hierzu nimmt Snort das erste Vorkommen des Suchmusters im Paket und prüft anschließend, ob alle weiteren Attribute auch zutreffen. Ist das nicht der Fall, so sucht Snort nach dem zweiten Vorkommen des Suchmusters im Paket und führt den gleichen Test erneut durch. Snort arbeitet sich so rekursiv durch das Paket.

Diese Arbeitsweise kann in besonderen Fällen von großem Nachteil sein. Daher verfügt Snort über einige diskrete Attribute, die in der Attributreihenfolge als Erstes aufgeführt werden, und vor der Multi-Rule-Inspection-Engine ausgewertet werden. Hierbei handelt es sich um die folgenden Attribute:

- `ack`. siehe S. 268
- `dsize`. siehe S. 267
- `flags`. siehe S. 267
- `flow`. siehe S. 268
- `fragbits`. siehe S. 266
- `icmp_id`. siehe S. 267
- `icmp_seq`. siehe S. 267
- `icode`. siehe S. 267
- `id`. siehe S. 266
- `ipopts`. siehe S. 266
- `ip_proto`. siehe S. 266
- `itype`. siehe S. 267
- `resp`. siehe S. 277
- `sameip`. siehe S. 266
- `seq`. siehe S. 268
- `session`. siehe S. 276
- `tos`. siehe S. 266
- `ttl`. siehe S. 266
- `window`. siehe S. 268

Diese Attribute sollten daher in den Regeloptionen immer zu Beginn angegeben werden. Dann führt ein Test dieser Attribute in vielen Fällen direkt zum Abbruch der Regel. Hierdurch kann die Detektionsmaschine von Snort stark beschleunigt werden (siehe auch: Abschnitt »Sortierung der Regeln – Aufbau der Detektionsmaschine« auf S. 348).

Es sollte bei Einsatz des `content`-Attributes auch immer geprüft werden, ob die Angabe eines Offsets (`offset`) oder einer maximalen Tiefe (`depth`) für die Suche möglich ist.

False Positives/Negatives

Ein sehr großes Problem beim Einsatz von Intrusion-Detection-Systemen sind die Falschmeldungen. Man unterscheidet zwischen falsch-positiven und falsch-negativen Meldungen.

Die falsch-positiven Meldungen sind eigentlich harmlos, aber lästig. Sie können jedoch auf lange Sicht auch gefährlich werden. Hierbei handelt es sich um Meldungen des NIDS, die behaupten, ein gefährliches Paket gesehen zu haben, welches in Wirklichkeit vollkommen gültig und korrekt war. Wenn Snort mit dem mitgelieferten Regelsatz aktiviert wird, werden zunächst in den meisten Umgebungen wahrscheinlich Hunderte falsch-positive Meldungen erzeugt. Hier ist der Administrator gefordert,

der die Meldungen analysieren und ihren Wahrheitsgehalt überprüfen muss und anschließend entscheidet, ob es sich um ein normales erlaubtes Verhalten in diesem Netzwerk handelt. Ist dies der Fall, sollte die entsprechende Regel deaktiviert oder modifiziert werden. Die große Gefahr bei einer Vielzahl von falsch-positiven Meldungen liegt in der Wahrscheinlichkeit, wichtige Meldungen zu übersehen, oder in der Erzeugung eines Desinteresses bei dem Benutzer, da er mit Meldungen überschwemmt wird.

Die falsch-negativen Meldungen sind wesentlich gefährlicher, da es sich hierbei um fehlende Meldungen tatsächlich ereigneter Einbrüche handelt. Das NIDS kann nur die Einbrüche erkennen, für die es konfiguriert wurde. Pakete und Einbrüche, die in den Regeln nicht berücksichtigt werden, werden auch nicht detektiert. Dies kann nur durch eine dauernde Pflege des Regelsatzes vermieden werden. Mehrere Webseiten im Internet bieten aktuelle Regelsätze für Snort an. Die URLs werden im Anhang aufgeführt.

Wenn Sie selbst Regeln entwickeln, sollten Sie darauf achten, dass Sie die Regeln nicht speziell auf den Exploit anzupassen versuchen, sondern eine Regel zu entwickeln, die die Sicherheitslücke erkennt. Bei einem Bufferoverflow sollten Sie also versuchen, ob es möglich ist, generell den Bufferoverflow zu erkennen. Wenn Sie spezielle Bytes im Shellcode testen, kann durch eine leichte Modifikation desselben Ihre Regel den Angriff nicht mehr erkennen!

Die vorgestellte Regel für die Erkennung des Bufferoverflows im IMAP-Server (siehe Abschnitt »Erzeugung der Regeln« auf S. 288) ist zwar funktionstüchtig, kann aber bei einer leichten Modifikation des Exploits den Angriff nicht mehr erkennen. Daher handelt es sich um eine schlechte Regel. Wesentlich besser ist zum Beispiel die folgende in Snort ab 2.1 enthaltene Regel:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 143 (msg:"IMAP login buffer overflow
↳ attempt"; flow:established,to_server; content:"LOGIN"; isdataat:100,
↳ relative; pcre:"\/sLOGIN\s[^\n]{100}/smi"; reference:cve,1999-0005;
↳ reference:nessus,10125; classtype:attempted-user; sid:1842; rev:9;)
```

Diese Regel prüft, ob es sich um ein IMAP-LOGIN-Paket handelt. Anschließend prüft das Attribut `isdataat`, ob dieses Paket 100 Bytes nach dem LOGIN noch Daten aufweist. Wenn auch dies erfüllt ist, wird mithilfe des `pcre`-Attributes geprüft, ob auf das `LOGIN` innerhalb der nächsten 100 Bytes ein Newline folgt. Wenn ja, ist die Anmeldung in Ordnung. Ansonsten wird eine Alarmmeldung ausgegeben. Das Newline schließt den `LOGIN`-Befehl ab.

Sortierung der Regeln – Aufbau der Detektionsmaschine

Die Reihenfolge der Regeln in der Konfigurationsdatei stimmt nicht unbedingt mit der internen Sortierung der Regeln durch Snort überein. Um dies zu verstehen, sind tiefere Kenntnisse des Aufbaus der Detektionsmaschine erforderlich.

Dieser interne Aufbau hat sich mit der Version 2.0 geändert. Während vorher die Regeln laut Abbildung 9.19 angeordnet wurden, wird ab Snort 2.0 nun die in Abbildung 9.20 dargestellte Methode genutzt. Diese neue Architektur ermöglicht theoretisch auch mehrfache Alarme pro Paket. Snort 2.0 kann diese Funktion jedoch noch nicht nutzen. Erst Snort 2.1 bietet mit der Konfigurationsdirektive `event_queue` eine Einstellungsmöglichkeit.

Zunächst werden alle Regeln entsprechend ihres Regeltyps (ihrer Aktion) sortiert. Die Regeln werden dann in der Reihenfolge `activation`, `dynamic`, `alert`, `pass`, `log` und anschließend die benutzerdefinierten Regeltypen abgearbeitet. Diese Reihenfolge kann mit der Konfigurationsoption `config order: modifiziert` werden:

```
# info sei von dem Benutzer definiert
config order: pass activation dynamic info alert log
```

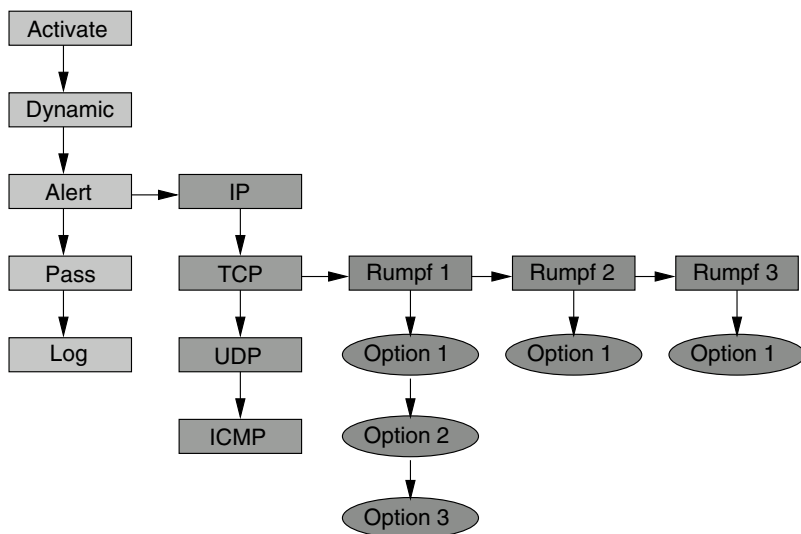


Abbildung 9.19 Aufbau der Snort-Detektionsmaschine vor Version 2.0

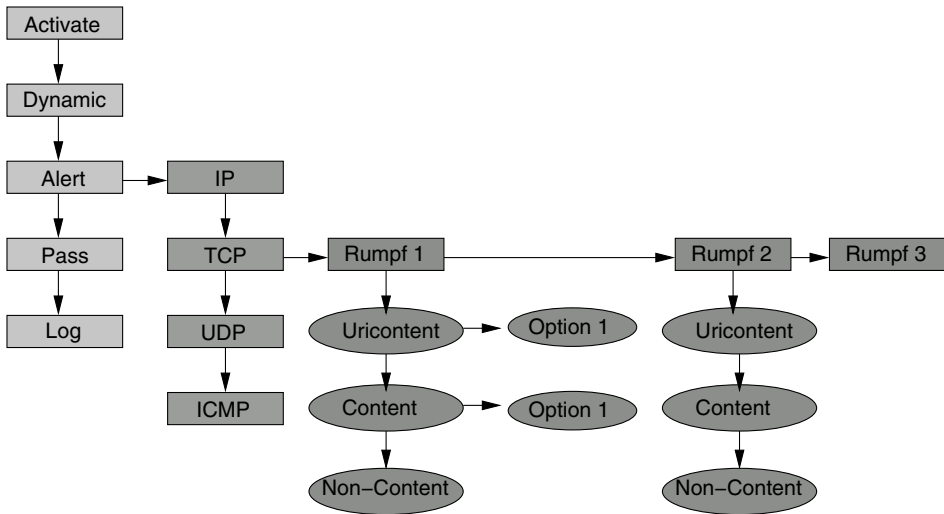


Abbildung 9.20 Neue Detektionsmaschine ab Snort 2.0

Standardmäßig führt Snort also die Alert-Regel vor der Pass-Regel aus. Das bedeutet, dass die Pass-Regel eigentlich ihre gewünschte Wirkung verfehlt. Dies ist so beabsichtigt, damit diese Regeln nur von erfahrenen Snort-Nutzern eingesetzt werden können.

Sind die Regeln nach Typen sortiert, werden sie anschließend nach den vier Snort bekannten Protokollen `ip`, `tcp`, `udp` und `icmp` sortiert. Für jeden dieser vier Bäume werden nun die Regeln nach ihrem Rumpf sortiert. Jeder Rumpf erzeugt einen (Regel-)Knoten im Baum.



Achtung:

Das ist nicht ganz korrekt. Teilweise werden auch noch Attribute hinzugezogen. So erzeugt das Attribut `itype` für ICMP-Pakete auch einen Knoten! Ein weiteres Attribut, welches einen eigenen Knoten erzeugt ist `ip_proto`.

Unterhalb des Knotens wird dann die Regeln in `uricontent`-, `content`- und nicht-`content`-Regeln gruppiert. Die Regeln werden dann auch in dieser Reihenfolge abgearbeitet. Innerhalb dieser Gruppen werden sie in der Reihenfolge angelegt, die durch die Konfigurationsdatei vorgegeben wird.

Analysiert Snort nun ein Paket, so prüft es zunächst das Protokoll und springt in den entsprechenden Baum. Dort werden die IP-Adressen des Paketes mit dem Inhalt je-

des Regelknoten verglichen. Stimmen sie überein, erfolgt der Vergleich der Ports. Stimmen IP-Adressen oder Ports nicht mit dem Inhalt des Regelknotens überein, springt Snort zum nächsten Regelknoten. Stimmen die Angaben im Regelknoten mit dem Paket überein, werden die Optionsknoten der Reihe nach getestet. Wird hier keine Übereinstimmung gefunden, springt Snort ebenfalls zum nächsten Regelknoten. Findet Snort hier eine Übereinstimmung, so wird die mit der Regel verbundene Aktion durchgeführt und die Detektionsmaschine verlassen.

Dieser interne Aufbau führt zu Konstellationen, die zunächst schwer zu verstehen sind. Betrachten Sie folgende Regeln:

```
# Regel 1
alert tcp any any -> $HOME_NET 23 (content: "su - root"; msg: "su to
↳ root attempt");
# Regel 2
alert tcp any any -> $HOME_NET 1:1023 (flags: S; msg: "Connectionattempt to
↳ privileged port");
# Regel 3
alert tcp any any -> $HOME_NET 23 (flags: S; msg: "Connectionattempt to telnet)
```

Unvoreingenommen würde ein Benutzer annehmen, dass die oben angegebenen Regeln in der Reihenfolge 1, 2 und 3 von Snort abgearbeitet würden. Dies würde dazu führen, dass die Regel 3 nie abgearbeitet würde, da die Regel 2 derartige Pakete mit einschließt. Mit Kenntnis der internen Struktur der Snort-Detektionsmaschine ist jedoch klar, dass Snort die Regeln zunächst nach ihrem Rumpf gruppieren wird. Das bedeutet, die Regeln 1 und 3 und die Regel 2 werden getrennt gruppiert. Da die Regel 1 sich vor der Regel 2 in der Konfiguration befand, werden alle Regeln, welche einen zu Regel 1 identischen Regelrumpf verwenden, vor Regel 2 getestet, so also auch die Regel 3. Das bedeutet, Snort arbeitet die Regel in der Reihenfolge 1, 3 und dann 2 ab.

Um nun über die Reihenfolge eine Geschwindigkeitssteigerung zu erreichen, sollte versucht werden, die häufiger zutreffenden Regeln zu Beginn anzuordnen, dann wird die Detektionsmaschine früh die Regel finden, die Aktion ausführen und die Abarbeitung für dieses Paket beenden.

Sicherheit – Chrooting Snort

Snort ist ein Netzwerkdienst. Wie andere Netzwerkdienste nimmt Snort Pakete ungewisser Herkunft an. Diese Pakete können bösartig sein. Snorts Aufgabe ist es, sogar diese bösartigen Pakete zu erkennen und zu melden. Es wurde bereits der Bufferoverflow als Sicherheitslücke erwähnt und im Anhang finden sich weitere Informationen über die Funktionsweise des Bufferoverflows. Auch Snort kann Sicherheitslücken aufweisen, wenn auch im Moment keine öffentlich bekannt sind. Entweder wurden sie noch nicht entdeckt oder eine neue Funktionalität einer zukünftigen Version wird sie in Snort einführen.

Um die Auswirkungen des Bufferoverflows möglichst gering zu halten, bietet Snort zwei Verfahren an: Abgabe der *root*-Privilegien und Chrooting. Schließlich soll Snort ein Werkzeug zur Einbruchererkennung sein und nicht Einbrüche fördern. Snort ist in der Lage, nach dem Start seinen Benutzerkontext zu ändern. Es muss mit *root*-Privilegien gestartet werden, damit es in der Lage ist, die Netzwerkkarte in den promiscuous-Modus zu schalten und Pakete zu sammeln. Sobald dies geschehen ist, werden die *root*-Privilegien nicht mehr benötigt und können abgegeben werden. Dies erfolgt entweder mit Kommandozeilenoptionen

```
snort -u snortuser -g snortgroup ...
```

oder mithilfe von Konfigurationsdirektiven:

```
config setuid: snortuser
config setgid: snortgroup
```

Ein Einbrecher, der mit einem Bufferoverflow in der Lage ist, die Kontrolle über Snort zu übernehmen, könnte nun nur noch Aktionen mit den Rechten des Benutzers *snortuser* und der Gruppe *snortgroup* ausführen. Dennoch hätte er noch Zugriff auf den gesamten Rechner und könnte jede Datei lesen/modifizieren, auf die dieser Benutzer Zugriff hat!

Das zweite Verfahren zur Beschränkung der Auswirkungen eines möglichen Angriffes auf Snort ist das Chroot-Verzeichnis. Hierbei modifiziert der Snort-Prozess den Verweis auf sein *root*-Verzeichnis so, dass er nicht mehr auf den gesamten Rechner zugreifen kann, sondern nur noch auf einen Teilbaum.

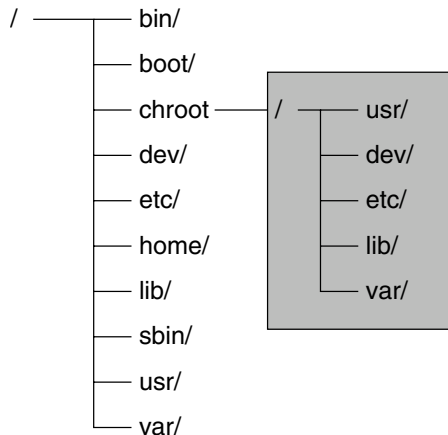


Abbildung 9.21 Chrooting-Snort. Wenn Snort gestartet wird, wechselt es sein */*-Verzeichnis und kann auf Dateien außerhalb nicht mehr zugreifen.

Dazu muss jedoch jede von Snort benötigte Datei sich innerhalb dieses Verzeichnisses befinden. Dazu zählen Bibliotheken, Konfigurationsdateien und Protokolldateien.

Die Erstellung dieses Verzeichnisses ist recht einfach. Hierzu ermitteln Sie zunächst die von Snort benötigten Bibliotheken:

```
#ldd /usr/sbin/snort
  libz.so.1 => /usr/lib/libz.so.1 (0x4002e000)
  libm.so.6 => /lib/i686/libm.so.6 (0x4003d000)
  libnsl.so.1 => /lib/libnsl.so.1 (0x4005f000)
  libmysqlclient.so.10 => /usr/lib/mysql/libmysqlclient.so.10 (0x40074000)
  libc.so.6 => /lib/i686/libc.so.6 (0x42000000)
  libcrypt.so.1 => /lib/libcrypt.so.1 (0x400a9000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Erzeugen Sie nun ein Verzeichnis, welches anschließend als *chroot* genutzt werden soll, erzeugen Sie die Verzeichnisstruktur und kopieren Sie die Dateien:

```
# mkdir -p /chroot_snort/usr/lib
# mkdir -p /chroot_snort/lib
# mkdir -p /chroot_snort/dev
# mkdir -p /chroot_snort/usr/sbin
# mkdir -p /chroot_snort/etc/snort
# mkdir -p /chroot_snort/lib/i686
# mkdir -p /chroot_snort/usr/lib/mysql
# mkdir -p /chroot_snort/var/log/snort

# cp /usr/lib/libz.so.1 /chroot_snort/usr/lib/
# cp /lib/i686/libm.so.6 /chroot_snort/lib/i686/
# cp /lib/libnsl.so.1 /chroot_snort/lib/
# cp /usr/lib/mysql/libmysqlclient.so.10 /chroot_snort/usr/lib/mysql/
# cp /lib/i686/libc.so.6 /chroot_snort/usr/lib/i686
# cp /lib/libcrypt.so.1 /chroot_snort/lib/
# cp /lib/ld-linux.so.2 /chroot_snort/lib/
# cp /etc/snort/* /chroot_snort/etc/snort/
# cp /etc/passwd /chroot_snort/etc/

### Der Vollständigkeit halber auch das Snort binary (streng genommen nicht
↳ nötig!)
# cp /usr/sbin/snort /chroot_snort/usr/sbin/
```

Damit nun Snort dieses Verzeichnis auch nutzt, kann Snort auf der Kommandozeile mit der Option *-t* das Verzeichnis übergeben werden oder es wird ein Eintrag in der Konfigurationsdatei vorgenommen:

```
config chroot: /chroot_snort
```

Achten Sie darauf, dass nun die Protokolle sich in `/chroot_snort/var/log/` befinden. Wünschen Sie eine Protokollierung durch den Syslog, so benötigt Snort das Socket `/dev/log`. Wenn Sie den Syslog starten, können Sie ihn mit der Option `-a` anweisen, weitere Sockets zu öffnen.

```
syslog -m 0 -a /chroot_snort/dev/log
```

Stealth – Konfiguration des Snortsensors als passives Element

Als Stealth-Modus bezeichnet man die Tatsache, dass das Interface, welches von Snort verwendet wird, um den Netzwerkverkehr zu protokollieren, nicht in der Lage ist, Pakete zu versenden. Dies ist ein Sicherheitsfeature, da es (fast) unmöglich zu erkennen ist, dass dieses Interface existiert und bei einem Angriff einer Sicherheitslücke in Snort der Angreifer keine Antwort erhält.

Ein derartiger Stealth-Modus kann mit Software- und Hardwaremitteln erreicht werden.

Softwarebasierte Stealth-Konfiguration

Die Konfiguration ist recht einfach auf dem Linux-Betriebssystem. Dazu kann die Netzwerkkarte aktiviert werden, ohne dass ihr eine IP-Adresse zugewiesen wird. Dies erfolgt mit:

```
ifconfig eth1 up
```

Diese Netzwerkkarte kann nun nicht mehr IP-Pakete versenden. Jedoch kann diese Netzwerkkarte noch ARP-Anfragen senden (und theoretisch auch beantworten). Um auch dies zu unterdrücken, kann das ARP-Protokoll auf der Netzwerkkarte deaktiviert werden.

```
ifconfig eth1 -arp up
```

Hardwarebasierte Stealth-Konfiguration

Es ist wesentlich schwieriger, Kabel herzustellen, welche nur den Empfang von Paketen ermöglichen und ein Senden unterbinden. Als noch AUI-Konnektoren eingesetzt wurden, konnten einfach die beiden Sendekabel unterbrochen werden. Wenn noch AUI-Transceiver vorhanden sind, können einfach die Kontakte 3 und 10 unterbrochen werden. Diese Transceiver werden nur noch das Empfangen von Paketen ermöglichen. Leider kann man kaum noch AUI-Transceiver kaufen. Außerdem sind mir keine 100-Mbit/s-Transceiver bekannt.

Wird als Medium 10Base2 (ThinCoax) eingesetzt, so ist es vollkommen unmöglich, ein ausschließliches Empfangskabel zu erzeugen, da lediglich zwei Adern existieren und diese zum Senden und Empfangen benötigt werden.

Auch im Falle von 10BaseT und 100BaseTX liegt der Fall nicht so einfach. Ein Durchtrennen der Sendeleitungen führt dazu, dass der Hub oder Switch, an dem das Kabel angeschlossen wird, den entsprechenden Port deaktiviert. Ein Hub erwartet über die

Sendekabel einen regelmäßigen Impuls, ansonsten wird ein toter Link angenommen. Eine Möglichkeit der Modifikation des Kabels besteht darin, das Kabel so abzuändern, dass die Fehlerrate stark ansteigt. Der Hub wird weiterhin den Link erkennen, jedoch die Pakete wegen fehlerhafter Prüfsummen verwerfen. Dies kann erfolgen, indem im Kabel zwei Adern entdrillt werden. Netzwerkkabel sind paarweise verdrillt (twisted pair), um ihre Unempfindlichkeit gegen Umgebungseinflüsse zu erhöhen (<http://www.robertgraham.com/pubs/sniffing-faq.html>).

Eine zweite Möglichkeit ist die Einführung eines Kondensators (http://personal.ie.cuhk.edu.hk/~msg0/sniffing_cable/).

Ist Snort an einen alten Hub angeschlossen, so besteht auch die Möglichkeit, die Signale, die vom Hub gesendet werden, an den Hub zurückzusenden. Dies ist jedoch bei einem Switch meist nicht möglich. Dazu werden am Hub-Ende die Pins 1 und 2 aufgetrennt und mit den Pins 3 und 6 verbunden.

Eine letzte Möglichkeit ist die Verbindung von Pin 1 und 2 mit den Pins 3 und 6 eines weiteren Ports des Hubs. Dies funktioniert auch mit allen mir bekannten Switches (http://www.theadamsfamily.net/~erek/snort/ro_cable_and_hubs.txt).

```

HUB PORT 1                HUB PORT 2
-----
x x r r                    r r x x
6 3 2 1                    1 2 3 6
| | | |                    | |
| | | -----            | |
| | -----              | |
| |
| |
| |
| |
6 3 2 1
r r x x
-----

```

Auf der Snort-Homepage ist auch der einfache Selbstbau eines passiven »WireTaps« beschrieben (<http://www.snort.org/docs/tap/>). Der Autor hatte aber noch nicht die Gelegenheit, dieses zu testen.

Weiterhin gibt es natürlich auch kommerziell erhältliche »WireTaps« (z.B. Shomiti), die in der Lage sind, diese Aufgabe zu übernehmen.

Hogwash

Hogwash (<http://hogwash.sourceforge.net>) ist der Versuch, die Fähigkeiten von Snort in einer Firewall zu nützen. Der Autor bezeichnet Hogwash als Paketschrubber. Dazu besteht die Möglichkeit, das Linux-Betriebssystem so zu erweitern, dass bei einem weiterzuleitenden Paket nicht oder nicht nur die Paketfilterregeln (iptables/netfilter) abgearbeitet werden, sondern auch Hogwash aufgerufen wird. Besonders interessant

ist die Möglichkeit, Hogwash komplett unsichtbar auf der Ebene 2 des Netzwerkprotokollstapels laufen zu lassen. Dann benötigt das zugrunde liegende Betriebssystem keinen IP-Stack und arbeitet als filternde Bridge im Netzwerksegment. Hogwash ist eine besondere Variante von Snort, momentan basierend auf der Version 1.8.1, die die Pakete testet und über ihre Weiterleitung entscheidet. Hogwash ist noch im Beta-Stadium, wird jedoch bereits weltweit genutzt. Laut seinem Autor Jed Haile skaliert Hogwash bis 100 Mbit/s. Weitere Informationen über Hogwash finden Sie auf der Homepage.

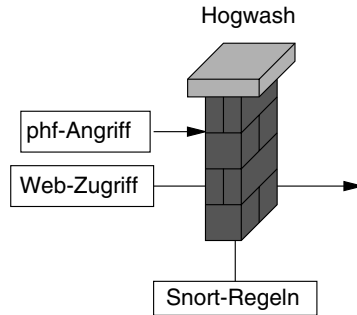


Abbildung 9.22 Funktionsweise von Hogwash



Achtung:

Hogwash wird nicht mehr aktiv weiterentwickelt. Seine Funktionalität ist im Wesentlichen aber in dem Produkt Snort-Inline enthalten, welches im Teil III, »Verfügbare Open-Source-Intrusion-Prevention-Systeme« besprochen wird.

Snort in einer geswitchten Umgebung

In einem Netzwerk, welches über einen Switch kommuniziert, kann Snort lediglich den Verkehr sehen, der vom Switch an den Snort-Sensor geschickt wird. In derartigen Umgebungen kann Snort daher nicht zufriedenstellend eingesetzt werden.

Zwei Möglichkeiten bestehen dennoch für den Einsatz von Snort. Möglicherweise besitzt der Switch einen so genannten Monitor oder Spanning-Port. Dieser Port kann so konfiguriert werden, dass er sämtlichen Verkehr oder den Verkehr einer bestimmten Anzahl von Ports sieht. Snort wird dann an diesen Port angeschlossen. Bei der Auswahl der Ports sollten die interessanten Rechner berücksichtigt werden. Nicht alle Switches unterstützen diese Konfiguration. Werden mehrere Switch-Ports an den

Monitor-Port weitergeleitet, so besteht auch die Gefahr, dass bei einer Auslastung der überwachten Ports die Geschwindigkeit des Monitor-Ports nicht ausreicht.

Als zweite Möglichkeit existiert der Anschluss von Snort mittels eines Hubs am Uplink des Switches. Häufig sollen sowieso nur die Pakete, die das Netzwerk von außen erreichen, von Snort untersucht werden. Wenn dies der Fall ist, so besteht die Möglichkeit, die Uplink-Verbindung des Switches über einen Hub zu gewährleisten, an dem dann auch der Snort-Sensor angebunden wird.

Snort-Addons

Es gibt eine ganze Reihe von Snort-Addons und Patches, die die Funktion von Snort erweitern. Die im Folgenden vorgestellten Werkzeuge und Patches sind sicherlich keine vollständige Liste und geben nur eine kleine Anzahl von Werkzeugen wieder, die der Autor nutzt.

Oinkmaster

Oinkmaster von Andreas Östling ist ein Werkzeug, um die Snort-Regeln automatisch zu aktualisieren. Hierbei handelt es sich um ein Perl-Skript, welches regelmäßige aktuelle Regelsätze herunterlädt, anpasst und Snort zur Verfügung stellt. Dieses Regel-Management-Werkzeug ist unter <http://oinkmaster.sourceforge.net> verfügbar.

Die Installation ist sehr einfach. Nach dem Download und dem Auspacken des Archivs werden die Dateien manuell in die entsprechenden Verzeichnisse kopiert:

```
$ tar xvzf oinkmaster-<version>.tar.gz
$ cd oinkmaster-<version>
$ sudo cp oinkmaster.conf /etc
$ sudo cp oinkmaster.pl /usr/local/bin
$ sudo cp oinkmaster.1 /usr/share/man/man1
```

Anschließend müssen Sie die Oinkmaster-Konfiguration anpassen. Dies kann direkt in der Konfigurationsdatei oder mit der GUI erfolgen. Um die GUI zu benutzen, benötigen Sie eine funktionierende Perl/TK-Umgebung. Sie starten die GUI durch den Aufruf des Befehls `./contrib/oinkgui.pl`. Anschließend können Sie die Konfiguration vornehmen (Abbildung 9.23).

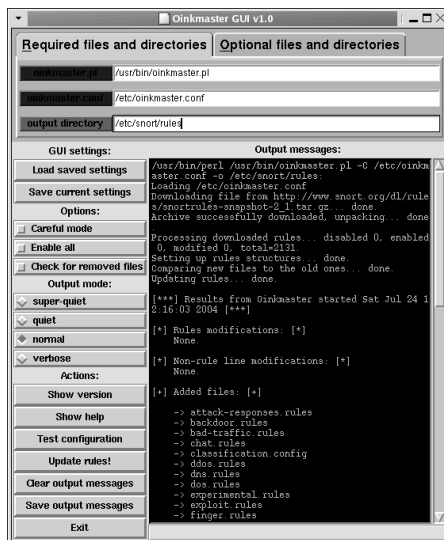


Abbildung 9.23 Die Oinkmaster-Gui erlaubt die grafische Konfiguration.

Alternativ kann die Konfiguration aber auch in der Konfigurationsdatei `/etc/oinkmaster.conf` durchgeführt werden. Dann kann Oinkmaster mit *Cron* regelmäßig täglich aufgerufen werden, um neue Regeln zu laden.

In der Konfigurationsdatei können Sie die folgenden Parameter verwenden:

- `url`. Dieser Parameter gibt den Ort des zu ladenden Snort-Regelarchives an. Oinkmaster unterstützt die folgenden Protokolle: `http://`, `https://`, `ftp://`, `file://` und `scp://`. Die zu ladende Datei muss auf `tar.gz` enden. Sie können hiermit auch sehr komfortabel eigene Regelsätze auf Ihren Sensoren verteilen.
- `scpkey`. Mit diesem Parameter können Sie für das Secure-Copy-Protokoll den privaten Schlüssel für die Anmeldung angeben.
- `path`. Der Inhalt der Pfad-Variablen während der Ausführung.
- `use_external_bins`. Dieser Parameter definiert, ob die externen Befehle `wget`, `tar` und `gzip` verwendet werden oder Perl-Funktionen genutzt werden. Auf Linux-Systemen werden per Default die externen Befehle genutzt.
- `tmpdir`. Dies ist ein temporäres Verzeichnis, in dem Oinkmaster während der Prozessierung die Regeln ablegt. Oinkmaster legt hier ein eigenes Unterverzeichnis an.
- `umask`. Mit dieser Option kann die aktuelle `umask` für die Ausführung von Oinkmaster überschrieben werden. Diese Option definiert die Rechte der neu angelegten Dateien (siehe `man umask`).
- `update_files`. Dieser reguläre Ausdruck beschreibt die von Oinkmaster zu modifizierenden Dateien. Normalerweise muss diese Option nicht geändert werden.

- `rule_actions`. Dieser reguläre Ausdruck beschreibt die Regelaktionender Regeln, die Oinkmaster aktualisieren soll. Wenn Sie eigene Regeltypen definiert haben, sollten Sie diese hier möglicherweise aufnehmen, wenn auch diese Regeln automatisch aktualisiert werden sollen.
- `min_files|min_rules`. Hiermit können Sie Sicherheitsüberprüfungen für das Update definieren. Wenn in dem neuen Regelarchiv weniger Regeldateien als `min_files` oder Regeln als `min_rules` gefunden werden, sieht Oinkmaster das Archiv als defekt an und bricht das Update ab.
- `use_path_checks`. Dies ist ebenfalls ein Sicherheitscheck, bei dem die Dateinamen und -pfade auf Sonderzeichen geprüft werden.
- `include`. Hiermit können Sie die `oinkmaster.conf`-Datei auf verschiedene Dateien aufteilen und gemeinsam laden.
- `skip_file`. Diese Option gibt die Dateien an, die dem regulären Ausdruck `update_files` entsprechen, aber dennoch nicht aktualisiert werden sollen. Diese Option kann mehrfach verwendet werden. Typische Dateien sind `local.rules` und `snort.conf`.
- `modifysid`. Diese Option erlaubt Ihnen, bestimmte Regeln in Abhängigkeit ihrer `SID` zu modifizieren. So ersetzt der folgende Ausdruck zum Beispiel die Variable `$EXTERNAL_NET` durch `$HOME_NET` in den angegebenen Regeln: `modifysid 302,429,1821 "\$EXTERNAL_NET" | "\$HOME_NET"`.
- `localsid`. (Ab Oinkmaster 1.1). Dieser Parameter erlaubt Ihnen die Angabe der Snortregeln, die nie aktualisiert werden sollen.
- `enablesid|disablesid`. Hiermit können sehr einfach bestimmte Regeln aktiviert oder deaktiviert werden. Dabei werden die Kommentarzeichen (`#`) entweder entfernt oder hinzugefügt.
- `define_template`. (Ab Oinkmaster 1.1). Mit diesem Parameter können Sie sich Templates definieren, die Sie anschließend auf verschiedenste Regeln anwenden können. Damit wird die Oinkmaster-Konfiguration bei vielen individuellen Einstellungen einfacher.

```
define_template name "modifythis" | "withthis"
```

- `use_template`. (Ab Oinkmaster 1.1). Hiermit wenden Sie das definierte Template auf bestimmte Regeln an (siehe auch die Datei `README.template`).

```
use_template name sid [ "arg1" "arg2" ... ]
```

Um den eigentlichen Update durchzuführen, müssen Sie nun Oinkmaster aufrufen. Hierbei unterstützt das Perl-Skript die folgenden Optionen:

- `-b dir`. Diese Option veranlasst Oinkmaster, die alten Regeln zunächst in einem Backup-Verzeichnis zu sichern.
- `-c`. Hier prüft Oinkmaster lediglich, ob ein Update verfügbar ist, aber führt es nicht durch (`careful`).
- `-C file`. Oinkmaster lädt diese alternative Konfigurationsdatei.
- `-e`. Schaltet alle Regeln an.

- -h. Gibt eine kurze Hilfe aus.
- -i. Schaltet den interaktiven Modus an, in dem vor jedem Update der Benutzer um eine Bestätigung gebeten wird.
- -m. (ab Version 1.1). Diese Option minimiert die Diff-Ausgabe, indem identische Anteile in der Ausgabe nicht angezeigt werden.
- -o *outputdir*. Kopiert die Regeln in dieses Verzeichnis. Üblicherweise wird hier das Snortregelverzeichnis angegeben. Dies ist die einzige verpflichtende Option.
- -q. Reduziert die Ausgaben auf ein Minimum (quiet).
- -Q. Unterdrückt alle Ausgaben (superquiet).
- -r. Prüft, ob Regeln in dem neuen Regelsatz entfernt wurden (removed).
- -T. Testet die Syntax der Konfiguration.
- -u. Oinkmaster verwendet diese alternative Download-URL.
- -U *file*. Fügt die Variablen aus der heruntergeladenen *snort.conf* der angegebenen Datei hinzu.
- -v. Verbose-Modus.
- -V. Version.

Bevor Sie Oinkmaster produktiv einsetzen, ist es immer sinnvoll, es mit der Option `-c` im *Careful*-Modus aufzurufen. Hier sehen Sie nur die Änderungen, die Oinkmaster an den Regeln vornehmen würde, ohne dass diese tatsächlich umgesetzt werden würden.

```
# oinkmaster.pl -c -o /etc/snort/ -C /usr/local/etc/oinkmaster.conf
Loading /usr/local/etc/oinkmaster.conf
Downloading file from http://www.snort.org/dl/rules/
↳ snortrules-snapshot-2_2.tar.gz... done.
Archive successfully downloaded, unpacking... done.
Setting up rules structures... done.
Processing downloaded rules... disabled 0, enabled 0, modified 0, total=2236
Setting up rules structures... done.
Comparing new files to the old ones... done.
Note: Oinkmaster is running in careful mode - not updating anything.

[***] Results from Oinkmaster started Sat Sep  4 09:04:38 2004 [***]

[+++]           Added rules:           [+++]

-> Added to exploit.rules (2):
  alert tcp $EXTERNAL_NET any -> $HOME_NET 443 (msg:"EXPLOIT SSLv2
  ↳ Client_Hello Challenge Length overflow attempt"; flow:to_server,
  ↳ established; flowbits:isnotset,sslv3.client_hello.request;
  ↳ byte_test:1,>,127,0; content: "|01|"; offset:2; depth:1;
  ↳ byte_test:2,<,768,3; flowbits:isnotset,sslv2.client_hello.request;
  ↳ flowbits:set,sslv2.client_hello.request;byte_test:2,>,32,9;
  ↳ classtype:attempted-admin; sid:2656; rev:2;)
```

... gekürzt ...

```
-> Added to sid-msg.map (82):
 120 || BACKDOOR Infector 1.6 Server to Client || cve,1999-0660 ||
    ↳ nessus,11157
 121 || BACKDOOR Infector 1.6 Client to Server Connection Request ||
    ↳ cve,1999-0660 || nessus,11157
```

... gekürzt ...

```
[---]      Removed non-rule lines:      [---]
```

```
-> Removed from sid-msg.map (80):
 120 || BACKDOOR Infector 1.6 Server to Client
 121 || BACKDOOR Infector 1.6 Client to Server Connection Request
 286 || POP3 EXPLOIT x86 BSD overflow || bugtraq,133 || cve,1999-0006
```

... gekürzt ...

```
[*] Added files: [*]
    None.
```

Sobald die Ausgabe des Testlaufes den Wünschen entspricht, kann die Option `-c` beim Aufruf entfernt werden. Oinkmaster wird dann den Snortregelsatz tatsächlich anpassen und erweitern.

Bleeding-Snort-Rules

Wenn Sie immer die neuesten Regeln für Ihren Snort-Sensor herunterladen möchten, so können Sie Oinkmaster als Download-Quelle das Bleeding-Snort-Clearinghouse (<http://bleedingsnort.com>) angeben. Hier werden alle fünf Minuten neue Regelsätze aus einem CVS-Repositorium erzeugt. Diese Regelsätze sind Oinkmaster-kompatibel.



Achtung:

Diese Regeln enthalten natürlich viele nur wenig getestete Regeln. Sie erzeugen daher häufig auch falsch-positive Meldungen. Nutzen Sie diese Regeln mit Vorsicht.

Snort!(FP)

Viele IDS-Systeme versuchen einen Mehrwert zu bieten, indem sie eine automatische Aktivierung der Regeln in Abhängigkeit von der Netzwerkumgebung bieten. So de-

aktivieren diese Systeme alle IIS-Regeln, wenn sich kein Internet Information Server in dem Netzwerk befindet. Ist einer vorhanden, so werden die Regeln nur für dieses System angewendet. Hierzu benötigt das IDS zusätzliche Informationen über die eingesetzten Systeme. Sourcefire hat hierfür die Realtime-Network-Awareness (RNA) für das kommerzielle auf Snort basierende IDS entwickelt. Während RNA für Open Source Snort nicht verfügbar ist, versuchen verschiedene Entwickler dies zu implementieren. Der erfolgversprechendste Ansatz ist im Moment Snort!(FP).

Snort!(FP) basiert auf dem passiven Fingerprinting-Werkzeug *p0f*.



Tipp:

Ein weiterer Ansatz wird gegenwärtig unter <http://www4.big.or.jp/~kanai/unix/snort/> entwickelt.

Snort!(FP) wird als Patch für Snort-2.1.2 unter <http://mysite.verizon.net/sdreed/> vertrieben. Dieser Patch enthält auch jeweils Patche für das *mysql*-Output-Plug-In und ACID. Damit ist es möglich, die Ergebnisse von Snort!(FP) direkt in ACID anzuzeigen.

Dieser Patch erweitert die Snort-Detektmaschine um die folgenden Attribute:

- `ttl`. Dieses Attribut ist um die Vergleichsoperatoren \leq und \geq erweitert worden.
- `window`. Dieses Attribut ist um die folgenden Vergleichsoperatoren erweitert worden:
 - `%nummer`. Dieser Operator berechnet den Modulus mit der angegebenen Nummer.
 - `Snummer`. Dieser Operator berechnet den Modulus der Maximum Segment Size (MSS).
 - `Tnummer`. Dieser Operator berechnet den Modulus der Maximum Transmission Unit (MTU).
- `length`. Dieses Attribut testet das Length-Feld im IP-Header des Paketes.
- `tcptopts`. Diese Option erlaubt den Test auf das Vorhandensein bestimmter TCP-Optionen.
- `quirks`. Hiermit können TCP-Unregelmäßigkeiten geprüft werden.

Snort!(FP) kommt bereits mit einer Vielzahl von vordefinierten Fingerprinting-Regeln, die genutzt werden können.

Testwerkzeuge

Jede Snort-Installation sollte anschließend auch getestet werden. Im Laufe der Jahre sind verschiedene Werkzeuge geschrieben worden, die auf der Basis der Snortregelsätze Pakete erzeugen, die ein Auslösen der Regeln provozieren. Da diese Werkzeuge aber nur einzelne Pakete und nicht komplette Verbindungen simulieren, konnten diese Werkzeuge in der Vergangenheit auch verwendet werden, um einen Denial-of-Service-Angriff gegen ein IDS auszuführen. Heute bietet Snort einen gewissen Schutz bei Anwendung der `stream4`-Zustandsüberwachung für Angriffe, die das TCP-Protokoll nutzen. Daher werden bei konsequenter Anwendung dieses Präprozessors simulierte TCP-Angriffe erfolglos bleiben. Auch für diesen Test eignen sich diese Testwerkzeuge.

Die bekanntesten Werkzeuge dieser Art sind `Stick` (<http://www.eurocompton.net/stick/projects8.html>), `Sneeze` (<http://snort.sourceforge.net/sneeze-1.0.tar>) und `Snot`. Im Weiteren soll die Anwendung von `Snot` genauer erläutert werden.

`Snot` kann von <http://www.stolenshoes.net/sniph/index.html> heruntergeladen werden. Es benötigt für eine erfolgreiche Übersetzung die `Libnet`-Bibliothek in der Version 1.0.2a von <http://www.packetfactory.net/Projects/Libnet/>.

Nach der Übersetzung kann `snort` aufgerufen werden und mit den folgenden Optionen können Sie den Test spezifizieren:

- `-r file`. Diese Option gibt die Snortregeldatei an.
- `-s address`. Diese Option gibt die zu verwendene Quell-IP-Adresse an.
- `-d address`. Diese Option gibt die zu verwendene Ziel-IP-Adresse an.
- `-l delay`. Diese Option fügt eine zufällige Verzögerung mit der maximalen Länge `delay` ein.
- `-n count`. Diese Option definiert die Gesamtanzahl der zu sendenden Pakete. Ist diese Option nicht gesetzt, so läuft `Snot` unendlich.
- `-p`. Diese Option deaktiviert die Erzeugung zufälligen Paketinhaltes.

Beispiel:

```
snort -r snort.rules -s 192.168.0.0/24 -d www.target.com -l 10
```



Achtung:

`Snot` wurde im Vergleich zu `Snort` nicht wesentlich weiterentwickelt. Daher kann es nur die Regelsprache von `Snort` 1.8 verwenden.

Mudpit

Mudpit (<http://www.fidelissecurity.com/techtalk/mudpit.asp>) ist ein alternativer Spool-Prozessor für Snort. Er kann an Stelle von Barnyard (siehe Abschnitt »Barnyard« auf S. 366) genutzt werden und bietet einige über Barnyard hinausgehende Funktionen.

Mudpit liest wie Barnyard die `unified-Log`-Dateien. Im Gegensatz zu Barnyard kann Mudpit aber sowohl die `alert`- als auch die `log`-Datei lesen und wieder zusammenführen. Dies ist umso wichtiger, da einige Informationen nur in der `alert`- oder in der `log`-Datei geschrieben, viele Informationen jedoch auch in beiden Dateien von Snort gespeichert werden. Als Ausgabe unterstützt Mudpit eine Textausgabe (`fast_alert`) und die Ausgabe in eine ACID-Datenbank (`acid`). Die Ausgabe in eine Sguil-Datenbank, wie Barnyard sie unterstützt, bietet Mudpit im Moment nicht.

Mudpit kann darüber hinaus die Protokolle mehrerer Snort-Sensoren einlesen und prozessieren.

Die Übersetzung und Installation von Mudpit erfolgt wie gewohnt mit den folgenden Befehlen:

```
$ tar xvzf mudpit-<version>.tar.gz
$ cd mudpit-<version>
$ ./configure
$ make
$ sudo make install
```

Beim Start kann das Verhalten von Mudpit mit einigen Optionen modifiziert werden:

- `-c file`. Hiermit kann die Konfigurationsdatei angegeben werden. Default: `/etc/mudpit.cf`
- `-v [-v [-v]]`. Dies erhöht die Anzahl der Meldungen von Mudpit (verbose).
- `-D|--daemon`. Hiermit startet Mudpit im Hintergrund.
- `--once`. Führt die Prozessierung der Log-Dateien nur einmalig durch und beendet sich.
- `-h|--help`. Gibt eine Hilfe aus.

Die wesentliche Konfiguration erfolgt in der Datei `/etc/mudpit.cf`. Diese Datei enthält mindestens zwei Abschnitte: `global` und `spool`. Eine Beispielkonfigurationsdatei ist im Folgenden abgedruckt:

Listing 9.11 Mudpit-Konfigurationsdatei

```
# Global parameters

global {
    daemon
    # verbose = 4
    class_file = "classification.config"
```

```

    sid_file = "sid-msg.map"
    gen_file = "gen-msg.map"
    ref_file = "reference.config"
#   run_once
#   nice = 1
#   pid_file = "/tmp/snortman.pid"
}

#spool configuration

spool "/var/log/snort" {
    lock = "mysql"
#   delete_processed
    user = "snort"
#   output = "/home/gvs/mudpit/output/fast_alert/mp_fast_alert.so"
    output = "/home/gvs/mudpit/output/acid/mp_acid_out.so",
    "server sensor, user sensor, database snortdb, interface eth0"
}

```

Nun werden die Parameter erläutert:

■ Globale Parameter

- `daemon`. Mudpit startet im Hintergrund (Default: Nein).
- `verbose=zahl`. Verbosity Level (Default: 0). Kann nicht im Daemon-Modus verwendet werden.
- `class_file|sid_file|gen_file|ref_file=file`. Diese Parameter definieren die entsprechenden Dateien der Snort-Konfiguration. Wenn die Dateinamen relativ angegeben werden, wertet Mudpit diese relativ zu dem Verzeichnis aus, in dem sich die Mudpit-Konfigurationsdatei befindet.
- `pid_file`. Mudpit speichert seine Prozess-ID in dieser Datei. Diese Datei kann anschließend genutzt werden, um den Prozess zu beenden (Default: `/var/run/mudpit.pid`).
- `nice=zahl`. Ändert die Priorität des Mudpit-Prozesses (Default: 0).
- `run_once`. Führt nur einen Durchlauf durch (Default: Nein).

■ Spool-Parameter

Diese Parameter definieren nun die verschiedenen Snort-Spool-Verzeichnisse und wie diese verarbeitet werden sollen.

- `lock=ressource`. Dies weist Mudpit an, die angegebene Ressource exklusiv zu locken (Default: None).
- `delete_processed`. Dies weist Mudpit an, prozessierte Einträge zu löschen (Default: Nein).
- `arch_dir=dir`. Hiermit kann ein Archiv-Verzeichnis angegeben werden, in das prozessierte Spool-Dateien kopiert werden.
- `user=user`. Mudpit verwendet diesen Benutzer zur Prozessierung.

- `checkpoint=file`. Dies ist die Checkpoint-Datei, die Mudpit nutzt.
- `output=plugin`. Hiermit wird das Output-Plug-In spezifiziert. Mit einem Komma getrennt können Argumente an das Output-Plug-In übergeben werden.

Barnyard

Um nun die Protokolle zu analysieren, können sie von Barnyard verarbeitet werden. Barnyard ist unter <http://www.snort.org/dl/barnyard/> zu finden.

Barnyard unterstützt drei verschiedene Modi:

- Einmalige Verarbeitung der Dateien (One Shot)
- Fortlaufende Verarbeitung der Dateien (Continual)
- Fortlaufende Verarbeitung der Dateien mit Statusdatei (Waldofile) (Continual with checkpoint)

Barnyard verwendet eine Konfigurationsdatei. In dieser Datei können die folgenden Optionen definiert werden:

```
# Eingabeprozessoren
# Dieser Prozessor verarbeitet die unified_alert Datei
processor dp_alert
# Dieser Prozessor verarbeitet die unified_log Datei
processor dp_event
#
#
# Ausgabeprozessoren
# Dieser Prozessor erzeugt Ausgaben analog dem alert_fast Output Plug-In
output alert_fast: /var/log/snort/snort_alert
# Dieser Prozessor erzeugt menschenlesbare Ausgaben inklusive Paketinhalten
output log_dump: /var/log/snort/snort_dump
# Dieser Prozessor erzeugt Protokolle in pcap-Format
output log_pcap: /var/log/snort/snort_pcap
# Dieser Prozessor erzeugt Datenbankeinträge für ACID (lediglich für MySQL)
output alert_acid_db: mysql, database snortdb, server host, user snortuser, \
↳ password geheim; sensor_id Sensor1; detail full
# Dieser Prozessor protokolliert via Syslog
output alert_syslog: LOG_LOCAL7 LOG_INFO
```

Beim Start von Barnyard können nun weitere Optionen definiert werden:

- `-a Archiv`. Archiviert die Spool-Dateien nach ihrer Prozessierung (noch nicht implementiert (Version 0.1.0-beta4)).
- `-c Konfig`. Gibt die Konfigurationsdatei für Barnyard an.
- `-d Spool-Verzeichnis`. In diesem Verzeichnis sucht Barnyard nach den Spool-Dateien (Default `/var/log/snort`).
- `-f Spooldatei`. Diese Option definiert den Basisnamen der zu analysierenden Spool-Dateien.

- `-g` *Generatordatei*. Diese Option definiert die Kartendatei *gen-msg.txt*. Diese Datei ordnet Präprozessorereignissen Texte zu (bei Barnyard dabei).
- `-h/-?`. Hilfe
- `-o`. One Shot-Modus
- `-s` *SidMapDatei*. Diese Datei enthält die Meldungen entsprechend den Snort-IDs (bei Snort dabei).
- `-t` *Zeit*. Beginnen mit Dateien, deren Zeitstempel im Namen jünger ist als **Zeit**.
- `-w` *WaldoDatei*. Statusdatei, damit Barnyard bei einem Neustart kontrollieren kann, wo es sich in einer wachsenden Protokolldatei befand.
- `-L` *Datei*. Ausgabeprotokoll
- `-R`: *Trockenlauf*. Barnyard testet die übergebenen Optionen und die Konfigurationsdatei auf Fehler.

Ein Beispielaufruf von Barnyard kann wie folgt aussehen:

```
barnyard -c /etc/barnyard/barnyard.conf -d /var/log/snort -f unified_alert\
-g /etc/barnyard/gen-msg.map -s /etc/barnyard/sid-msg.map
```

Barnyard-Installation

Hierzu wird das neueste Quelltextarchiv von <http://www.snort.org/dl/barnyard> benötigt. Dieses wird ausgepackt, konfiguriert und übersetzt:

```
# cd /usr/local/src
# tar xvfz /path/barnyard-version.tar.gz
# ./configure --prefix=/usr --bindir=/usr/sbin --sysconfdir=/etc/snort \
--enable-mysql --enable-postgresql
# make
```

Anschließend kann das Paket erneut in einem Archiv eingepackt und auf die Sensoren transferiert werden.

```
# cd ..
# tar cvzf barnyard-compiled.tar.gz barnyard-version/
```

Auf den Sensoren ist nun die Installation mit `make install` durchzuführen bzw. sind von Hand die Dateien an die entsprechenden Stellen zu kopieren.

```
# cp barnyard /usr/sbin/
# mkdir /etc/barnyard
# cp etc/* /etc/barnyard
```

FLoP – Das Fast Logging Project

Das Fast Logging Project für Snort (FLoP, <http://www.geschke-online.de/FLoP/>) von Dirk Geschke ist ein alternativer Mechanismus, um die Alarmmeldungen von Snort in einer zentralen Datenbank zu protokollieren. Während aber Barnyard oder Mudpit die

unified-Log-Dateien liest und jedes Ereignis einzeln in die Datenbank einfügt, arbeitet FLoP anders.

FLoP patched Snort, so dass Snort über ein neues weiteres Output-Plug-In verfügt: `alert_unixsock_db`. FLoP enthält dann ein Programm, welches sich an diesen Socket bindet, von Snort die Alarmmeldungen entgegennimmt und schnell (in zwei Paketen) über das Netzwerk an den zentralen Datenbankrechner schickt. Dort nimmt FLoP das Paket entgegen und fügt die Meldung in die zentrale Datenbank. Das Hinzufügen der Meldung in die Datenbank erfolgt dann lokal über schnelle UNIX-Sockets.

Eine Blockade von Snort kann nicht erfolgen, da diese immer in den UNIX-Socket schreiben kann.

Die FLoP-Plug-In-Installation ist recht einfach:

```
$ tar xvzf snort-2.1.x.tar.gz
$ tar xvzf FLoP-1.x.x.tar.gz
$ cd snort-2.1.x
$ patch -p1 <../FLoP-1.x.x/patches/snort-2.1.x_patch
$ ./configure
$ make
$ sudo make install
```

Das anschließend installierte Snort-Binary verfügt nun über das FLoP-Output-Plug-In. Dieses muss noch konfiguriert werden. Hierzu fügen Sie die folgende Zeile Ihrer Snort-Konfiguration hinzu:

```
output alert_unixsock_db: /tmp/snort.sock
```

Die Installation von FLoP selbst ist ähnlich einfach:

```
$ cd FLoP-1.x.x
$ ./configure --with-snort=/path/to/snort-2.1.x \
              --with-postgres=/path/to/postgres \
              --with-mysql=/path/to/mysql \
              --enable-drop \
              --enable-alert \
              --enable-getpacket \
              --enable-fpg \
$ make
$ sudo make install
```

Bei der Konfiguration des Quelltextes können die folgenden Optionen angegeben werden:

- `--prefix=dir`. Hiermit kann der spätere Installationsort angegeben werden.
- `--with-snort=dir`. Dies ist das Quelltextverzeichnis von Snort.

- `--with-mysql=dir`. Dies aktiviert MySQL-Unterstützung. Wenn der `configure`-Befehl die MySQL-Datenbank nicht findet, kann ihr Verzeichnis angegeben werden.
- `--with-postgres=dir`. Dies aktiviert die PostgreSQL-Unterstützung. Wenn der `configure`-Befehl die PostgreSQL-Datenbank nicht findet, kann ihr Verzeichnis angegeben werden.
- `--enable-drop`. Hiermit wird auch das Programm `drop` gebaut und die Schnittstellen in `sockserv` und `servsock` werden aktiviert.
- `--enable-alert`. Hiermit wird auch das Programm `alert` gebaut und die Schnittstelle in `sockserv` und `servsock` werden aktiviert.
- `--enable-getpacket`. Hiermit wird auch das Programm `getpacket` gebaut. Dieses Programm ist in der Lage ein Netzwerkpaket aus der Datenbank wiederherzustellen. Hierfür ist jedoch eine Erweiterung des Datenbankschemas erforderlich.
- `--enable-fpg`. Hiermit wird das Programm `fpg` gebaut. Dieses benötigt zusätzlich die Libnet-Bibliothek. Hierbei handelt es sich um einen False-Positive-Generator.

Auf dem Sensor starten Sie nun den Prozess `sockserv`. Ein typischer Aufruf wäre:

```
sockserv -S address.of.your.server -p port -s /tmp/snort.sock
```

Das Programm `sockserv` unterstützt bei seinem Aufruf die folgenden Optionen:

- `-A delay`. Diese Option veranlasst `sockserv` in den angegebenen Zeitabständen statistische Informationen auszugeben.
- `-b`. Diese Option startet das Programm im Hintergrund. Gleichzeitig wird die Option `-l` aktiviert.
- `-D dropsocket`. Wenn mehr Ereignisse als `<Highwater>` in der Warteschlange sind, werden so viele Ereignisse an den Dropsocket geschickt, bis `<Lowwater>` erreicht wird.
- `-h`. Gibt eine Hilfe aus.
- `-H Highwater`. Hiermit wird die Hochwassermarke gesetzt (Default: 10000).
- `-l`. Dies protokolliert die statistischen Informationen (`-A`) an Syslog anstelle auf der Konsole.
- `-L Lowwater`. Setzt die Niedrigwassermarke.
- `-m mode`. Hiermit wird die Umask für den Daemon-Modus gesetzt.
- `-M MaxTry`. Anzahl der maximalen Wiederholungen der Serververbindung.
- `-p port`. Gibt den Port auf dem `servsock`-Server an.
- `-P pidfile`. Der Prozess speichert hier seine Prozess-ID.
- `-s snortsocket`. Diese Option gibt den zu erzeugenden Socket für Snort an.
- `-S Server`. Der Prozess verbindet sich mit dem angegebenen Server.
- `-w dir`. Dies ist das Arbeitsverzeichnis im Daemon-Modus.
- `-W waittime`. Diese Zeit wartet der Prozess zwischen zwei Wiederholungen des Verbindungsaufbaus mit dem Server.

Auf dem Datenbankserver wird anschließend der Prozess *servsock* gestartet. Dieser Prozess unterstützt identische Optionen. Zusätzlich bietet dieser Prozess die folgenden Optionen:

- `-d`. Gibt die Konfiguration aus.
- `-c configfile`. Liest die Konfiguration aus einer Datei ein.
- `-f`. Diese Option speichert zusätzliche Optionen in der Datenbank, so dass das Programm *getpacket* die Pakete wiederherstellen kann.
- `-M priority`. Hiermit werden Alarmmeldungen mit einer höheren Priorität an einen Alert-Socket geschrieben. Diese können mit dem Program *alert* gelesen und als E-Mail verschickt werden.
- `-n`. Führt keine Reverse-DNS-Auflösung der IP-Adresse der Sensoren durch.
- `-s dbsocket`. Gibt den Datenbank-Socket an.
- `-U alertsocket`. Hiermit wird der Alert-Socket angegeben.
- `-u`. Schaltet den Alert-Socket ab.
- `-W swapdir`. Das Swap-Verzeichnis wird genutzt, wenn die Datenbank nicht erreicht werden kann.

Das Programm *servsock* benötigt zusätzlich auch noch eine Konfigurationsdatei. Hier werden zum Beispiel die Datenbankinformationen gespeichert. In diesen Konfigurationsdateien können die folgenden Parameter genutzt werden:

```
DBuser: user
DBpassword: password
DBname: name
DBtype: MySQL|Postgres
DBencoding: hex|ascii|base64
DBtrust: 0|1
DBtrans: 0|1
PIDfile: file
SocketName: socket
ServerName: name
ServerPort: port
AlarmDelay: seconds
Syslog: 0|1
FQNSensor: 0|1
AlertSocket: socket
UnixPriority: priority
DropSocket: socket
HighWater: value
LowWater: value
DaemonMode: 0|1
Umask: mode
SwapDir: directory
FullPayload: 0|1
```

Die Programme *alert* und *drop* sind in der Lage, über einen Socket Meldungen entgegenzunehmen und per E-Mail zu versenden.

Das Programm *fpg* ist ein False-Positive-Generator. Es ist wie *snot* in der Lage, Pakete auf der Basis von Snortregeln zu generieren.

Für die `FullPayload`-Funktion (`-f`) ist eine Erweiterung der Datenbank erforderlich. Diese wird in der Dokumentation von FLoP genau beschrieben.

**Tipp:**

FLoP ist im Moment das mächtigste Output-Plug-In für Snort!

9.2.11 Snort-Wireless

Snort bietet einen eingeschränkten Wireless-Modus an. Dieser wird mit der Option `-w` aktiviert. Snort setzt dann die Wireless-Karte in den RFMON-Zustand. Nun kann Snort auch alle Wireless-Pakete anzeigen: Management, Control und Data. Diesen Modus muss jedoch der Kartentreiber unterstützen.

Da Snort jedoch keine Attribute besitzt, um Wireless-Pakete zu analysieren, kann dieser Modus nur verwendet werden, um derartige Pakete anzuzeigen.

Mit dem Snort-Wireless-Projekt wird jedoch ein IDS für Wireless-Netzwerke auf der Basis von Snort erzeugt (<http://snort-wireless.org/>).

Leider basiert das Projekt im Moment noch auf Snort 2.0. Allerdings gibt es einen experimentellen Patch für Snort 2.1.1.

Snort-Wireless Optionen

Der Snort-Wireless-Patch fügt im wesentlichen ein weiteres Protokoll zu Snort hinzu. Hierbei handelt es sich um das `wifi`-Protokoll, welches spezifisch 802.11-Rahmen erkennt. Die allgemeine Syntax einer `wifi`-Regel ist:

Aktion `wifi` *mac* *Richtung* *mac* (*Regel-Attribute*)

Snort-Wireless unterstützt die gleichen Aktionen wie Snort. Die Angabe der MAC-Adressen erfolgt entweder einzeln (`00:0F:1F:1C:BD:15`) oder als Liste (`[00:0F:1F:1C:BD:15, 00:90:4B:88:D0:17]`).

Snort-Wireless verfügt auch über einige zusätzliche Regel-Attribute:

- `frame_control:[!]nummer`. Dieses Attribut prüft das Frame-Control-Feld. Die Nummer darf einen Wert zwischen 0 und 65535 annehmen.
- `type:[!]wifi-typ`. Hiermit können Sie den Typ des Paketes testen. Folgende Typen sind möglich: `STYPE_MANAGMENT`, `STYPE_CONTROL` und `STYPE_DATA`.
- `stype:[!]wifi-subtyp`. Dies erlaubt die Angabe des Wifi-Subtyps. Folgende Subtypen sind erlaubt:
 - **Management-Frame-Subtypen:**
 - `STYPE_ASSOCREQ`
 - `STYPE_ASSOCRESP`
 - `STYPE_REASSOC_REQ`
 - `STYPE_REASSOC_RESP`
 - `STYPE_PROBEREQ`
 - `STYPE_PREBERESP`
 - `STYPE_BEACON`
 - `STYPE_ATIM`
 - `STYPE_DISASSOC`
 - `STYPE_AUTH`
 - `STYPE_DEAUTH`
 - **Control-Frame-Subtypen**
 - `STYPE_PS`
 - `STYPE_RTS`
 - `STYPE_CTS`
 - `STYPE_ACK`
 - `STYPE_CFPEND`
 - `STYPE_CFPEND_CFACK`
 - **Data-Frame-Subtypen**
 - `STYPE_DATA`
 - `STYPE_CFACK`
 - `STYPE_CFPOLL`
 - `STYPE_CFACK_CFPOLL`
 - `STYPE_NULL`
 - `STYPE_CFACK_NULL`
 - `STYPE_CFPOLL_NULL`
 - `STYPE_CFACK_CFPOLL_NULL`

- `from_ds`: TRUE|FALSE|ON|OFF. Hiermit können Sie prüfen, ob das Paket von einem Accesspoint (Distribution System) an einen Client gesendet wird.
- `to_ds`: TRUE|FALSE|ON|OFF. Dieses Attribut testet ob ein Paket an den Accesspoint gesendet wird.
- `more_frags`: TRUE|FALSE|ON|OFF. Dieses Attribut testet, ob das 802.11-Paket fragmentiert wurde.
- `retry`: TRUE|FALSE|ON|OFF. Dieses Attribut betrachtet nur erneute Sendungen verlorener Pakete.
- `pwr_mgmt`: TRUE|FALSE|ON|OFF. Dieses Argument prüft, ob sich das sendende Gerät im Stromsparmmodus befindet.
- `more_data`: TRUE|FALSE|ON|OFF. Dieses Attribut prüft das *more-data-control-flag* im 802.11-Paket.
- `wep`: TRUE|FALSE|ON|OFF. Dieses Attribut prüft, ob das Paket mit Wired-Equivalent-Privacy (WEP) geschützt wird.
- `order`: TRUE|FALSE|ON|OFF. Hiermit können Sie prüfen, ob die Pakete in der *strictly-ordered* Serviceklasse gesendet werden.
- `duration_id`: [!] *nummer*. Hiermit können Sie das Duration/ID-Feld in 802.11-Paketen prüfen.
- `bssid`: [!] *nummer*. Hiermit können Sie die Basic-Service-Set-ID (BSSID) des Paketes testen.
- `seqnum`: [!] *nummer*. Dieses Attribut testet die Sequenznummer. Die angegebenen Nummer darf einen Wert von 0 bis 4095 besitzen.
- `fragnum`: [!] *nummer*. Dieses Attribut testet die Fragmentnummer (0-15).
- `addr4`: [!] <mac>. Hiermit testen Sie das vierte Addressfeld in dem 802.11-Paket
- `ssid`: [!] *SSID*. Dieses Attribut testet die Service-Set-Identifizier (SSID) des Paketes. Diese darf 32 Buchstaben lang sein.

Zusätzlich unterstützt Snort-Wireless auch einige neue Präprozessoren:

- *RogueAP*. Diese Option erlaubt es unbekannte Accesspoints und AdHoc-Netzwerke zu erkennen. Hierzu müssen Sie Ihre Accesspoints und die verwendeten Kanäle konfigurieren. Das erfolgt sinnvollerweise in einer Variablen. Anschließend können Sie den Präprozessor aktivieren.

```
var ACCESS_POINTS [XX:XX:XX:XX:XX:XX, YY:YY:YY:YY:YY:YY, ...]
var CHANNELS [X, Y, ...]
preprocessor rogue_ap: $ACCESS_POINTS, $CHANNELS, scan_flag [0 | 1], \
scan_timeout [num], expire_timeout [num]
```

Die Optionen `scan_flag` und `scan_timeout` (Default 60 Sekunden) sind im Moment noch nicht implementiert. Dieser Präprozessor wird nicht-autorisierte Accesspoints erkennen und protokollieren. Anschließend wird er den Accesspoint in einer Liste aufnehmen, so dass keine erneute Alarmierung ausgelöst wird. Erst nachdem der Accesspoint für eine bestimmte Zeit (`expire_timeout`, Default

12 Stunden) keine weiteren Pakete mehr gesendet hat, wird er von der Liste gestrichen. Ein erneutes Auftauchen des Accesspoints löst dann eine erneute Alarmierung aus: *Rogue AP detected*.

- **Antistumbler.** Dieser Präprozessor versucht Netstumbler-ähnlichen Verkehr und Kismet-Verkehr zu erkennen. Netstumbler (<http://www.netstumbler.com/>) wie Kismet (<http://www.kismetwireless.net/>) sind Werkzeuge, welche nach Wireless-LANs und Accesspoints sucht. Dieser Präprozessor kann den Einsatz des Werkzeuges an den versendeten Paketen mit NULL-SSID-Felder erkennen. Sie aktivieren den Präprozessor mit:

```
preprocessor antistumbler: probe_reqs [num], probe_period [num], \
expire_timeout [num]
```

Dabei gibt der Parameter `probe_reqs` (Default 5) die erforderliche Anzahl der Netstumbler-Pakete in dem Zeitraum `probe_period` (Default 3 Sekunden) für einen Alarm an. Die gefundenen Netstumbler werden in einer Liste gespeichert, um erneute Meldungen zu unterdrücken. Nach der angegebenen Zeit (`expire_timeout`, Default 60 Sekunden) werden die Einträge wieder gelöscht.

- **Authflood.** Dieser Präprozessor versucht die Flutung eines Accesspoints mit Authentifizierungspaketen zu erkennen. Dieser Präprozessor ist lediglich in dem Snort-2.1.1-Patch enthalten. Die Aktivierung ist sehr einfach.

```
preprocessor authflood: auth_threshold [num], expire_timeout [num], \
target_limit [num], prune_period [num]
```

Dabei meldet der Präprozessor einen Alarm, wenn mehr Authentifizierungspakete als `auth_threshold` (Default 100) innerhalb der Zeit `expire_timeout` (Default 60 Sekunden) an einen Accesspoint gesendet werden. Der Präprozessor erzeugt hierfür eine Liste der gefluteten Accesspoints im Speicher. Die maximale Anzahl der Accesspoints geben Sie mit `target_limit` (Default 10) an. Nach einer bestimmten Zeit (`prune_period`, Default 30 Sekunden) werden die Accesspoints wieder aus der Liste entfernt.

- **Deauthflood.** Dieser Präprozessor versucht die Flutung eines Accesspoints mit Deauthentifizierungs-Paketen zu erkennen. Dieser Präprozessor ist lediglich in dem Snort-2.1.1-Patch enthalten. Er unterstützt die identischen Optionen wie der Authflood-Präprozessor.

```
preprocessor deauthflood: auth_threshold [num], expire_timeout [num], \
target_limit [num], prune_period [num]
```

- **Macspoof.** Dieser Präprozessor ist lediglich in dem Snort-2.1.1-Patch enthalten. Er versucht die Verwendung gespoofter MAC-Adressen zu erkennen. Dieser Präprozessor weist jedoch im Betrieb Probleme auf, da einige Netzwerkgeräte hier falsch-positive Meldungen erzeugen. Deren MAC-Adressen sollten daher von der Überwachung ausgeschlossen werden.

```
preprocessor macspooft: <ausgeschlossene MAC-Adressen>, \
tolerate_gap [num], threshold [num], expire_timeout [num], \
spoofed_addr_limit [num], prune_period [num]
```

Der Präprozessor analysiert die Sequenznummern der versendeten Pakete jeder MAC-Adresse. Treten hierbei Sprünge oder Lücken auf, so kann er einen Alarm auslösen. Der Parameter `threshold` (Default 10) definiert, wieviele ungewöhnliche Lücken in dem Zeitraum `expire_timeout` (Default 120 Sekunden) auftreten dürfen, bevor ein Alarm ausgelöst wird. Der Parameter `spoofed_addr_limit` (Default 100) gibt die maximale Anzahl der überwachten MAC-Adressen an. Für diese Anzahl legt der Präprozessor eine Liste im Speicher an. Der Parameter `prune_period` (Default 30) spezifiziert, wann eine MAC-Adresse wieder aus diesem Speicher entfernt wird.

9.2.12 Konfigurationswerkzeuge für Snort

Die Konfiguration von Snort kann wie beschrieben mit einem Texteditor vorgenommen werden. Viele Anwender wünschen jedoch menübasierte oder sogar grafische Werkzeuge.

Leider wurden viele Werkzeuge, die in der ersten Auflage vorgestellt wurden, wie Snort-Center, SnortConf, SneakyMan etc., von ihren Entwicklern nicht mehr weiterentwickelt, so dass als einzige brauchbare Alternative das Windows-Werkzeug IDS Policy Manager übrig blieb.

Win32 IDS Policy Manager

Der IDS Policy Manager wird unter <http://www.activevortex.com/idspm/index.htm> gepflegt. Jeff Dell veröffentlicht das Programm als Freeware. Dieses Programm wird aktiv weiterentwickelt und gepflegt und stellt im Moment sicherlich das mächtigste Programm für diesen Zweck dar. Leider ist es nur für Windows verfügbar.

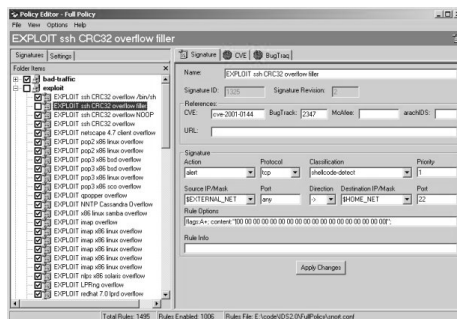


Abbildung 9.24 Win32 IDS Policy Manager-Bildschirm

9.3 ARPwatch

ARPwatch ist sicherlich kein vollständiges Network Intrusion Detection System, aber es kann einen wichtigen Baustein darstellen. ARPwatch wird unter der GNU GPL-Lizenz vertrieben.

Im Snort-Kapitel und auch im Anhang finden Sie Informationen über ARP-Spoofing. Kurz zusammengefasst stellt ARP-Spoofing eine Methode dar, mit der ein Angreifer, der die Kontrolle über einen Rechner in einem Netzwerk erlangt hat, trotz Einsatz eines Switches beliebigen Netzwerkverkehr belauschen und zum Beispiel auf Kennwörter hin analysieren kann. Hierbei fälscht der Angreifer die im Netzwerk vorhandenen MAC-Adressen, indem er vorgetäuschte ARP-Antworten im Netzwerk verteilt. Dies führt zu falschen IP-Adresse/MAC-Adresse-Paarungen in den ARP-Tabellen der kommunizierenden Rechner.

Dieser Angriff ist mit den üblichen Methoden nur sehr schwer zu bemerken. ARPwatch kann jedoch die Paarungen in einer Tabelle protokollieren und kontrollieren. Ähnlich einem Switch liest *ARPwatch* alle ARP-Pakete mit und pflegt eine Tabelle (*arp.dat*) mit den Informationen. Tauchen widersprüchliche Informationen auf, so protokolliert *ARPwatch* dies und sendet zusätzlich eine E-Mail an den Administrator *root*.

ARPwatch ist wahrscheinlich bereits Teil der von Ihnen eingesetzten Distribution. Ansonsten kann ARPwatch von <ftp://ftp.ee.lbl.gov/arpwatch.tar.gz> heruntergeladen werden.

Die Konfiguration von *arpwatch* ist recht einfach. Es besitzt folgende Optionen:

- `-f arp.dat`. MAC/IP-Datenbank
- `-i eth0`. Interface
- `-n Netzwerk/Maske`. Weitere »lokale« Netzwerke
- `-u user`. Verwendet die Rechte des Benutzers *user*

Vor dem Start von *ARPwatch* ist es wichtig, die Datei *arp.dat* als leere Datei anzulegen.

```
# touch /var/arpwatch/arp.dat
# arpwatch -u arpuser -i eth1 -f /var/arpwatch/arp.dat
```

ARPwatch wird dann eine neue MAC/IP-Paarung wie folgt mit dem Syslog protokollieren

```
May 15 14:44:51 kermit arpwatch: new station 192.168.111.101 0:e0:7d:7d:70:69
```

und folgende E-Mail an *root* senden:

```
Delivered-To: postfix@kermit.spenneberg.de
From: arpwatch@kermit.spenneberg.de (Arpwatch)
```

To: root@kermit.spenneberg.de
Subject: new station (grobi)
Date: Wed, 15 May 2002 14:44:51 +0200 (CEST)
X-Virus-Scanned: by AMaViS perl-11

hostname: grobi
ip address: 192.168.111.101
ethernet address: 0:e0:7d:7d:70:69
ethernet vendor: Encore (Netronix?) 10/100 PCI Fast ethernet card
timestamp: Wednesday, May 15, 2002 14:44:51 +0200

