15. Firewalls unter UNIX

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Gliederung

- Grundlagen
- Linux - iptables
- **OpenBSD - pf Toolkit (bis 4.6)**
- BSD, Solaris - Ipf Toolkit
- Löcher in Firewalls – Virtuelle Private Netzwerke

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

15.2 Firewall - OpenBSD

Historisches

Der Paket-Filter **pf** wurde ursprünglich von Pyun YoungHyeon für FreeBSD entwickelt. Gleichzeitig entwickelte Daniel Hartmeier einen Paket-Filter für OpenBSD. Nachdem P.YoungHyeon die Arbeiten einstellte wurde daraus bei FreeBSD das pf4freebsd Projekt. Dieses Projekt wurde wesentlich von Daniel Hartmeier unterstützt, der dafür sorgte daß im Weiteren ein einheitlicher Paket-Filter **pf** für OpenBSD und FreeBSD entwickelt wurde.

15.2 Firewall - OpenBSD

Merkmale

Der Paket-Filter **pf** ist Bestandteil des Kerns. Der Paketfilter wird mittels des Programms **pfctl** verwaltet:

- Starten
- Stoppen
- Regeln laden
- Regeln löschen

Jedes Paket durchläuft den vollständigen Regelsatz. Die **letzte** Bewertung ist die **ausschlaggebende**!!!! Das Durchlaufen des Regelsatzes kann zwangsweise beendet werden (**quick**).

Systems Architecture Group
http://sar.informatik.hu-berlin.de

Merkmale(2)

Der Paket-Filter pf unterstützt:

- Filtern in der IP-Schicht mit Routerfunktionalität oder Bridgefunktionalität
- NAT und Redirection (Port Forwarding)
- Load Balancing
- Logging
- Srubbing (Paketnormalisierung)
- Queueing und Prioritisierung
- Dynamische Tabellen, Listen und Makros für IP-Adressen
- Anachors - für dynamisches Laden von Regeln

Konfiguration(1)

Aktivieren per Hand:

Forwarding einschalten: sysctl -w net.inet.ip.forwarding=1
oder sysctl -w net.inet.ip6.forwarding=1

```
pfctl -e                    # einschalten des Firewalls
pfctl -f /etc/pf/pf.conf    # Laden der Filterregeln
```

Permanente Aktivierung:

/etc/rc.conf

```
pf=YES
pf_rules=/etc/pf.conf
pflogd_flags=
```

/etc/sysctl.conf

```
net.inet.ip.forwarding=1
```

15.2 Firewall - OpenBSD

Konfiguration(2)

Version 4.6!!!!!!!!

Konfigurationsfile:

- /etc/pf.conf
  - Struktur:
    - Makros
    - Tabellen
    - Optionen
    - Scrub (Paketnormalisierung)
    - Queueing
    - NAT und RDR
    - Filter-Regeln

Kleines Beispiel für pf.conf (1)

```
ext_if = "bge0"
int_if  = "bge1"
lan_net = "192.168.1.0/24"
table <firewall> const {self}
# Alle Pakete übersetzen
scrub in all
# vorläufig alle IN- und OUT-Pakete blockieren
block in all
block out all
# an lo0 alles sofort durchlassen
pass quick on lo0 all
# gefälschte Absenderadressen am internen Interface verbieten
antispoof quick for $int_if inet
```

Kleines Beispiel für pf.conf (2)

```
# ssh zum Firewall nur von 192.168.1.15 erlauben
block return in quick on $int_if proto tcp from ! 192.168.1.15 \
      to $int_if port ssh flags S/SA
# IN und OUT-Pakete am internen Interface erlauben
pass in on $int_if from $lan_net to any
pass out on $int_if from any to $lan_net
#  OUT für externes Interface
pass out on $ext_if proto tcp all  modulate state flags S/SA keep state
pass out on $ext_if proto { tcp,udp, icmp } all keep state
# protokollieren von ssh nach innen
pass in log on $ext_if proto tcp from any to ! <firewall> \
        port ssh flags S/SA synproxy state
```

pfctl – das Konfigurationstool (1)

Das Verhalten des Paket-Filters pf wird durch das Programm pfctl gesteuert. Mittels pfctl können alle Wartungsaufgaben für den Paket-Filter durchgeführt werden.

Vollständige Syntax des Kommandos:

pfctl [**-AdeghNnOqRrvz**] [**-a** *anchor*[:*ruleset*]]
[**-D** *macro=value*] [**-F** *modifier*] [**-f** *file*]
[**-i** *interface*] [**-k** *host*] [**-p** *device*] [**-s** *modifier*]
[**-T** *command* [address ...] [**-t** *table*] [**-x** *level*]

pfctl – das Konfigurationstool(2)

Die wichtigsten Parameter von **pfctl**:

-d - Paket-Filter ausschalten

```
pfctl -d
```

-e - Paket-Filter einschalten

```
pfctl -e
```

-f *file* - Laden der Filterregeln von dem File file.
Wenn keine weiteren Optionen angegeben sind, werden alle Regeln geladen.

```
pfctl -f /etc/pf.conf
pfctl -f /etc/pf.conf.neu
```


Systems Architecture Group
http://sar.informatik.hu-berlin.de

15.2 Firewall - OpenBSD

pfctl – das Konfigurationstool(3)

Reihenfolge im Regelfile:

- Macros: Benutzerdefinierte Variablen für IP-Adressen, Ports, Interfaces u.s.w.
- Tabellen: Mit Tabellen können Listen von IP-Adressen verwaltet werden. Die Einträge lassen sich während der Laufzeit ändern.
- Optionen: Konfigurieren die Arbeitsweise von pf
- Scrub: Dient zur Normalisierung oder Defragmentierung von Paketen
- Translation: Regeln für NAT und Redirektion
- Filterregeln: Selektive Regeln um Pakete an einem Interface durchzulassen oder zu stoppen.

pfctl – das Konfigurationstool(4)

Ergänzende Parameter für Option **-f** *file*:

- **-A** - Nur Queue-Regeln aus dem File *file* laden
- **-N** - Nur NAT-Regeln aus dem File *file* laden
- **-O** - Nur Optionen aus dem File *file* laden
- **-R** - Nur Filterregeln aus dem File *file* laden

Andere Optionen außer **-f** *file* werden bei den obigen Optionen jeweils ignoriert!

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

pfctl – das Konfigurationstool(5)

-F *modifier* - Aktivierte Paket-Filter-Regeln löschen

modifier:

- all - alle Regeln löschen
- nat - NAT Regeln löschen
- rules - Filterregeln löchen
- state - Status-Tabelle löschen
- info - Statistik-Tabellen lösche
- queue - Queue Regeln löschen
- Tables - Tabelle mit dyn. IP-Adressen löschen
- osfp - Operating system fingerprints löschen
- Sources - Source Tracking Tabelle löschen

pfctl – das Konfigurationstool(6)

-s *modifier* - Anzeigen der gesetzten Paket-Filter Regeln, Parameter, auch in Kombination mit **-v**

modifier:

all (**a**) - Alles ausgeben
nat (**n**) - NAT-Regeln ausgeben
rules (**r**) - Nur Filterregeln ausgeben
states (**s**) - Statusinformationen ausgeben
Interfaces (**I**) - Interfaces ausgeben
info (i) - Statistikinformationen

pfctl -v -s rules ; pfctl -v -s r

Systems Architecture Group
http://sar.informatik.hu-berlin.de

pfctl – das Konfigurationstool(6)

-s *modifier* - Anzeigen der gesetzten Paket-Filter Regeln,
Parameter, auch in Kombination mit **-v**

modifier:

osfp (**o**) - Fingerprints anzeigen
queue(**q**) - Queue-Regeln anzeigen
timeouts(**t**) - Timeouts anzeigen
memory(**m**) - Memory-Limits anzeigen
A**nchors**(**A**) - Benutzte Anker anzeigen
Sources, **labels**, **Tables**

pfctl -v -s timeouts ; pfctl -v -s t

Systems Architecture Group
http://sar.informatik.hu-berlin.de

pfctl – das Konfigurationstool(7)

-n - pfctl probeweise ausführen (Syntaxcheck)
pfctl -n -f /etc/pf.conf.test

-x *level* - Debug-Level

-i *interface* - Operationen nur für das Interface interface ausführen

-z - Löschen der Statistiken für die Filterregeln

-k *host* - Löschen der Statusinformationen für den Host *host*

-p *device* - Filter-Gerät umsetzen auf *device*. Standard ist /dev/pf

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

15.2 Firewall - OpenBSD

pfctl – das Konfigurationstool(8)

-h - Hilfe ausgeben
-v - verbose
-q - nur Fehler ausgeben
-r - reverse DNS-Lookup ausführen

pfctl – das Konfigurationstool(9)

Tabellenverwaltung mittels **pfctl**:

-t *tabellenname* **-T** *kommando*

-t *tabllenname* - spezifiert den Namen der Tabelle

-T *kommando* - gibt das auszuführende Kommando *kommando* an

Kommandos:

kill (**-k**), **flush**(**-f**), **add**(**-a**),
delete(**-d**), **replace**(**-r**), **show**(**-s**),
test(**-t**), **zero**(**-z**), **load**(**-l**)

pfctl – das Konfigurationstool(10)

Anker(anchor) Verwaltung mittels **pfctl**:

-a *anchor-name*[:*ruleset-name*] [*flags*]

Anker dienen zum dynamischen laden von Regeln während der Arbeit des Paket-Filters.

anchor-name - Name des Ankers (vorher in pf.conf definiert oder Standard-Anker)

ruleset-name - Name des Regelsatzes innerhalb des Ankers.

flags - **-F -f**, und **-s** siehe oben

Systems Architecture Group
http://sar.informatik.hu-berlin.de

Struktur des Konfigurationsfiles

Im Konfigurationsfile gibt es folgende sieben Typen von Anweisungen. Mit Ausnahme der Definition von Makros und Tabellen müssen die Anweisungen in folgender Reihenfolge gruppiert sein.

1. **Makros**
2. Tabellen
3. Optionen
4. Verkehrsnormalisierung (Scrub)
5. Queueing
6. Adressumsetzung (NAT und RDR)
7. Paket Filterung

Makros und Listen(1)

Das Paket-Filter-Tool pfctl unterstützt einen einfachen Makromechanismus in den Konfigurationsfiles, z.B. /etc/pf.conf, für benutzerdefinierte Variable, IP-Adressen, Listen von IP-Adressen, Schnittstellennamen, Portnamen usw.. Weiterhin ist es möglich Mengen von IP-Adressen zu Listen zusammenzufassen und diese in den Regeln anstelle von IP-Adressen zu benutzen und so die Zahl der Regeln zu reduzieren.

Makros werden wie folgt definiert:

makroname **= "** Zeichenkette **"**

Makros können rekursiv definiert werden. **"**-Zeichen maskiert **$**-Zeichen!!!!

Aufgerufen werden die Makros wie folgt:

$*makroname*

Listen sind eine Aufzählung von IP-Adressen oder Portnummern, die durch Komma oder Leerzeichen getrennt sind und in geschweifte Klammern stehen.

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Makros und Listen(2)

Beispiele:

```
EXT_IF = "bge0"
INT_IF = "bge1"
block in on $EXT_IF from any to any
NISSERVER = "{ 141.20.20.18, 141.20.20.50, 141.20.21.0/24 }"
NFS1 = 141.20.20.52
NFS2 = 141.20.20.67
NFS3 = 141.20.21.68
NFSSERVER = "{" $NFS1 $NFS2 $NFS3 "}"
pass in on $INT_IF from $NISSSERVER to any
pass in on $INT_IF from $NFSSERVER to any
```

Struktur des Konfigurationsfiles

Im Konfigurationsfile gibt es folgende sieben Typen von Anweisungen. Mit Ausnahme der Definition von Makros und Tabellen müssen die Anweisungen in folgender Reihenfolge gruppiert sein.

1. Makros
2. **Tabellen**
3. Optionen
4. Verkehrsnormalisierung (Scrub)
5. Queueing
6. Adressumsetzung (NAT und RDR)
7. Paket Filterung

Tabellen(1)

Tabellen sind eine weitere Möglichkeit Gruppen von IP-Adressen zusammenzufassen. Im Unterschied zu Listen können Tabellen auch dynamisch verwaltet werden und sind wesentlich schneller in der Verarbeitung als Listen.

Tabellen werden wie folgt im Konfigurationsfile definiert:

table <*tabellen-name***>** [*type*] { *liste* } { **file** *filename* }

tabellen-name - Name der Tabelle, muß immer in spitzen Klammern stehen, auch bei der Dereferenzierung.

type - Type der Tabelle. Folgende Typen sind möglich:

persist - Tabelle bleibt immer im Speicher, auch wenn sie leer ist.

const - Der Inhalt der Tabelle ist konstant und kann durch **pfctl** nicht verändert werden.

Tabellen(2)

table <*tabellen-name***>** [*type*] { *liste* } { **file** *filename* }

liste - Liste von IP-Adressen oder IP-Netzen (141.20.20.0/24) in geschweiften Klammern und durch Kommas getrennt.

file *filename* - Das File mit dem Namen *filename* enthält eine Menge von IP-Adressen oder IP-Netzen - pro Zeile ein Eintrag.

Beispiele:

```
table <nfsserver> const { 141.20.20.55, 141.20.20.56/31 }
table <nisserver> const file /etc/nisserver
table <spam> persist file /etc/spammer
block in on bge0 from { <spam> } to any
```

Tabellen(3)

Tabellenverwaltung mit **pfctl**(1):

pfctl -t *tabellenname* **-T** *kommando* [*Adressen*]

-t *tabellenname* - spezifiert den Namen der Tabelle, der Name wird ohne spitze Klammern angegeben. Die Tabelle muß vorher definiert sein (außer bei add).

-T *kommando* - gibt das auszuführende Tabellensubkommando *kommando* an. Folgende Subkommandos sind möglich:

add - Hinzufügen von neuen Einträgen in eine Tabelle, existiert die Tabelle nicht, wird eine neue Tabelle mit dem Namen *tabellenname* erzeut.

```
pfctl -t spam -T add 194.177.20.10
```

kill - löschen der angegebenen Tabelle. Der Tabellename und die Eintragungen in der Tabelle gehen verloren.

```
pfctl -t spam -T kill
```

Tabellen(4)

Tabellenverwaltung mit **pfctl**(2):

pfctl -t *tabellenname* **-T** *kommando*

flush - Löschen aller IP-Adressen in der angegebenen Tabelle. Der Tabellenname bleibt erhalten.

```
pfctl -t spam -T flush
```

```
pfctl -t spam -T add 198.123.12.3
```

delete - Löschen der angegebenen IP-Adresse in der Tabelle.

```
pfctl -t spam -T delete 198.123.12.3
```

replace - Löschen aller IP-Adressen in der angegeben Tabelle und gleichzeitig die angegeben IP-Adressen hinzufügen (**flush** und anschließen **add**)

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Tabellen(5)

Tabellenverwaltung mit **pfctl**(2):

pfctl -t *tabellenname* **-T** *kommando*

load - Laden der Tabellendefinitionen von einem angegeben File. Standard ist /etc/pf.conf

pfctl -T load -f /etc/pf.table.conf

show - Anzeigen des Inhaltes der angegebenen Tabelle

pfctl -t spam - T show

test - Testen ob die angegeben Adressen in der Tabelle enthalten sind.

pfctl -t spam -T test 198.123.1.23

zero – Löschen der Statistikinformationen der Tabelle.

pfctl -t spam -T zero

Struktur des Konfigurationsfiles

Im Konfigurationsfile gibt es folgende sieben Typen von Anweisungen. Mit Ausnahme der Definition von Makros und Tabellen müssen die Anweisungen in folgender Reihenfolge gruppiert sein.

1. Makros
2. Tabellen
3. Optionen
4. Verkehrsnormalisierung (Scrub)
5. Queueing
6. Adressumsetzung (NAT und RDR)
7. **Paket Filterung**

Paket Filterung(1)

Allgemeine Syntax für ein Filter-Regel:

action direction [**log** | **log-all**] [**quick**] **on** *interface* [*af*] [**prot** *protocol*]
from *src_addr* [**port** *src_port*]
to *dst_addr* [**port** *dst_port*]
[**flags** *check/mask*] [*state*]

action - Aktion, die ausgeführt werden soll, wenn die nachfolgend aufgeführten Bedingungen erfüllt sind. Folgende Aktionen existieren:

pass - Das Paket kann passieren.

block - Das Paket soll blockiert werden. Es gibt folgende Möglichkeiten ein Paket zu blockieren: **drop** (standard) und **return**.

Paket Filterung(2)

Beispiele:

```
#  Alle Paket werden blockiert
block all
# Alle eingehenden Pakete werden blockiert
block in all
# Alle ausgehenden Pakete werden blockiert
block out all
# Alle Pakete dürfen passieren
pass all
# Alle ausgehenden Pakete dürfen passieren
pass out all
# Alle eingehenden Pakete dürfen passieren
pass in all
```

Paket Filterung(3)

direction - Flußrichtung der Pakete. Entweder **in** - eingehende Pakete oder **out** - ausgehende Pakete. Werden keine weiteren Parameter angegeben, ist auch **all** für eingehende und ausgehende Pakete zulässig.

log - Zusätzlich zu der eigentlichen Aktion wird ein LOG-Eintrag geschrieben. Bei status-behafteten Paketen wird nur das Startpaket protokolliert.

log-all - Bei status-behafteten Paketen werden alle Pakete protokolliert. Regeln enden mit: **keep state**, **modulate state**, **synproxy**

quick - die Aktion wird sofort ausgeführt. Alle weiteren Regeln werden nicht beachtet.

Paket Filterung(4)

interface - Name eines Interfaces oder einer Gruppe von Interfaces. Wenn das untersuchte Paket von diesem Interface stammt, kann die Aktion ausgeführt werden.

```
block in on bge0 all
```

af - Adressfamilie. Folgende Adressfamilien sind zulässig: **inet** und **inet6**. Wenn das Paket zu der spezifizierten Familie gehört, kann die Aktion ausgeführt werden.

```
block in on bge0 inet6 all
pass in on bge0 inet all
block out on bge0 inet6 all
```

Paket Filterung(5)

proto *protocol* – Protokoll-Type des Paketes. Wenn das Paket zu dem angegeben Protokoll gehört, kann die Aktion ausgeführt werden. Die Protokollangabe *protocol* kann wie folgt erfolgen:

- die Bezeichner tcp, udp, icmp, icmp6
- ein gültiger Protokollname aus dem dem File /etc/protocols oder entsprechenden Namensdiensten (NIS, LDAP)
- eine gültige Protokollnummer (0..255)
- eine Liste von Protokollen ({ 6, 17 })

```
block in log quick on bge0 inet proto { tcp, udp }
```

Paket Filterung(6)

from [**!**] *src_addr* [**port** *src_port*] **to** [**!**] *dst_addr* [**port** *dst_port*] (1)

Adressangaben für das Paket. Wenn die Adressenangaben übereinstimmen kann die Aktion ausgeführt werden. Die Parameter können wie folgt spezifiziert werden:

src_addr, dst_addr - IP-Adressangabe für Source- bzw. Destination-Adresse. Folgende Angaben sind möglich:

- *ipaddr* - IP-Adresse (141.20.20.20)
- *network* - IP-Netzwerk (141.20.20.0/24)
- *dns-name* - voll qualifizierter DNS-Name (xyz.xxx.de)
- **any** - alle IP-Adressen
- **no-route** - nicht-routebar IP-Adressen
- <*table*> - eine Tabelle *table*
- *interface***:network** – Netzwerk, dass zum dem Interface interface gehört

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Paket Filterung(7)

from [**!**] *src_addr* [**port** *src_port*] **to** [**!**] *dst_addr* [**port** *dst_port*] (2)

Die Parameter können wie folgt spezifiziert werden:

src_port, dst_port – Portangabe für Source- bzw. Destination-Port. Folgende Angaben sind möglich:

port *portnummer* [*op portnummer* | {,*portnummer*}]

op steht für einen der Operatoren:

=, != , <, <=, >, >=, : (von : bis),

>< (von:bis - ohne Grenzen), <> (außer)

portnummer steht für ein Zahl zwischen 0 und 65535 oder einen gültigen Portnamen aus /etc/services.

portliste – Liste von Ports, durch Komma getrennt.

Fehlt die Portangabe, sind alle Ports gültig.

Paket Filterung(8)

Beispiele:

```
block in log quick on bge0 inet proto tcp from any to any
# "all" ist gleich "from any to any"
block in log quick on bge0 inet proto tcp all
pass in from any to any
pass in proto tcp from any port <1024 to any
pass in proto tcp from bge0:network to any
block in proto tcp from ! 141.20.20.0/24 to any port ssh
block in proto tcp from ! 141.20.20.0/24 to 141.20.20.50 port ssh
pass in proto udp from { 141.20.20.50, 141.20.20.51 } port 2049 \
        to 141.20.20.67 port 1000><1024
```

Paket Filterung(9)

flags *check/mask* (1)

Mittels des Parameter flags lassen sich die Flags in TCP-Paketen testen. Damit ist es möglich den Zustand einer TCP-Verbindung abzufragen. Folgende Bits sind für *check* zulässig:

- **F**: FIN - Finish, Ende der Verbindung
- **S**: SYN - Synchronize, Verbindung soll aufgebaut werden
- **R**: RST - Reset, Abbruch der TCP-Verbindung
- **P**: PSH - Push, Paket sofort zustellen, nicht puffern
- **A**: ACK - Acknowledgement, gültige ACK-Nummer im Paket
- **U**: URG - Urgent, wichtig, sofort zuzustellen
- **E**: ECE - Explicit Congestion Notification Echo
- **W**: CWR - Congestion Window Reduced

Paket Filterung(10)

flags *check*/*mask* (2)

Für die Maske *mask* sind Kombinationen der für *check* zulässigen Bits anzugeben. Die Aktion wird ausgeführt, wenn genau das durch check angegebene Bit bei den durch Maske mask spezifizierten Bits gesetzt ist.

Beispiel:

```
pass in on bge0 proto tcp from any to any port ssh flags S/SA
# Paket passiert, wenn S gesetzt ist und  A nicht gesetzt ist.
# S/SA charakterisiert eine neue Verbindung
```

Paket Filterung(11)

Der Parameter *state* wird für das Anlegen einer Statusinformation für eine Verbindungen genutzt. Dies setzt eine erfolgreiche **pass**-Regel voraus (alle Bedingungen in der Regel erfüllt). Die Statusinformation wird beim Schließen der Verbindung (TCP) oder nach einem Timeout (ICMP, UDP) gelöscht. Ist für eine Verbindung eine Statusinformation angelegt worden, passieren Pakete, die zu dieser Verbindung gehören, den Paket-Filter ohne Anwendung des Regelwerkes. Folgende Angaben sind für state möglich:

keep state: Anlegen einer Statusinformation. Protokolle: TCP,UDP, ICMP. NAT, BINAT und RDR-Regeln erfordern **keep state**!!!

modulate state: Anlegen einer Statusinformation, Erhöhung der Sicherheit. Nur für TCP. Bessere ISN (initial sequence number).

synproxy state: Der Firewall übernimmt das komplett Handshaking und erst dann wird die Verbindung weitergereicht.

Paket Filterung(12)

Beispiele:

für keep state

```
block all
pass out proto tcp from any to any flags S/SA keep state
pass in proto tcp from any to any port 25 flags S/SA keep state
pass out inet proto icmp all icmp-type echoreq keep state
```

für modulate state

```
block all
# modulate state geht nur für neue Verbindungen
pass out proto tcp from any to any modulate state
# bei schnellen Netzwerken wird durch S/SA ein ACK-Sturm verhindert
pass in proto tcp from any to any port 25 flags S/SA modulate state
```

Paket Filterung(13)

Beispiele:

für synproxy state

```
block all
# hierdurch wird ein „SYN-Floods“ verhindert, weil der
# Verbindungsaufbau mit dem WWW-Server erst ausgeführt
# wird, wenn der Verbindungsaufbau zwischen Firewall und
# Client abgeschlossen ist.
pass in proto tcp from any to any port www flags S/SA synproxy state
```


Systems Architecture Group
http://sar.informatik.hu-berlin.de

Paket Filterung(14)

Dynamisches Laden von Regeln(1)

Damit man mit Hilfe vom pfctl Regeln dynamisch nachladen kann, müssen vorher Namen für die verschieden Ladepunkte (anchor attachment points) festgelegt werden. Die entsprechenden Anweisungen müssen dort stehen, wo die Regeln später eingefügt werden sollen. Es gibt folgende Ladepunkt-Typen:

- **anchor** *name* – Ladepunkt *name* für Filterregeln für die Paket-Filterung wird definiert.
- **rdr-anchor** *name* – Ladepunkt *name* für RDR-Regeln wird definiert
- **binat-anchor** *name* – Ladepunkt *name* für BINAT-Regeln wird definiert
- **nat-anchor** *name* – Ladepunkt *name* für NAT-Regeln wird definiert

Paket Filterung(15)

Dynamisches Laden von Regeln(2)

Anchors können während der Initialisierung (pfctl -f /etc/pf.conf) durch die **load**-Anweisung geladen werden. Dadurch erhalten die entsprechenden Abschnitte „Anfangsregeln“.

load anchor *name*:*ruleset* **from** *filename* – Laden von Regeln aus dem File *filename*. *rulset* ist der Name der geladenen Regeln. Dieser wird später für **pfctl** benötigt.

Paket Filterung(15)

Dynamisches Laden von Regeln(3)

Beispiel:

/etc/pf.conf.spam:

```
block in quick from 220.160.10.0/24 to any
```

/etc/pf.conf

```
block on bge0 all
anchor spam
load anchor spam:base file /etc/pf.conf.spam
pass out on bge0 all keep state
pass in on bge0 proto tcp from any to bge0 port smtp keep state
```

/etc/pf.conf.spam.neu:

```
block in quick from 194.170.10.0/24 to any
pfctl -f /etc/pf.conf
pfctl -a spam:neu -f /etc/pf.conf.spam.neu
pfctl -a spam:neu -s rules     # Regeln in spam:neu anschauen
pfctl -a spam:neu -F rules     # Regeln spam:neu löschen
```

Paket Filterung(16)

„spoofed“ Pakete blocken

antispoof wird benutzt um „Address-spoofing“ (verfälschen von Absenderadressen) zu unterdrücken. Der Paketfilter bietet mittels **antispoof** einen gewissen Schutz dagegen.

anitspoof [**log**] [**quick**] for *interface* [*af*]

log, **quick,** *interface* und *af* haben die üblichen Bedeutungen.

Beispiel:

```
# bge1 192.168.1.1 mit Netzmaske 255.255.255.0
 anitspoof for bge1 inet
 #entspricht etwa
 #   block in on ! bge1 inet from 192.168.1.0/24 to any
 #   block in inet from 192.168.1.1 to any
 #   blockieren von Paketen vom loopback-Interface zu lokalen Adressen
 #       „pass quick on lo0 all“   hilft dagegen
```

Paket Filterung(17)

OSFP

OSFP (Operating System Finger Printing) ist eine Methode zur passiven Bestimmung des Betriebssystems des Partnerhosts. Dies erfolgt mit Hilfe bestimmter Verhaltensmustern bei den TCP-SYN Paketen. Der Paketfilter unterstützt OSFP. Die dazu notwendigen Informationen werden in der Datei /etc/pf.os abgelegt und können nach dem Start des Paketfilters mittels pfctl -s osfp besichtigt werden. OSFP kann in den Regeln in Erweiterung des from-Parameters benutzt werden.

Beispiele:

```
pass in on bge0 from any os "Linux"
block in on bge0 from an os "Windows 2000"
block in on bge0 from any os "Windows 95"
block in on bge0 from any os "Windows 98"
```

Struktur des Konfigurationsfiles

Im Konfigurationsfile gibt es folgende sieben Typen von Anweisungen. Mit Ausnahme der Definition von Makros und Tabellen müssen die Anweisungen in folgender Reihenfolge gruppiert sein.

1. Makros
2. Tabellen
3. **Optionen**
4. Verkehrsnormalisierung (Scrub)
5. Queueing
6. Adressumsetzung (NAT und RDR)
7. Paket Filterung

Optionen(1)

Mit Hilfe der Anweisung **set** können verschiedenste Eigenschaften des Paketfilter eingestellt werden. Mit wenigen Ausnahmen ist dies im Allgemeinen aber nicht notwendig.

set block-policy [**drop** | **return**] - Setzen des Blockverhaltens. **drop** bewirkt ein wegschmeißen des Paketes, **return** bewirkt ein TCP RST und ein entsprechendes ICMP-Paket.

set debug [**none** | **urgent** | **misc** | **loud**] - Setzen des Debug-Niveaus.

- **none** - keine Debug-Nachrichten erzeugen
- **urgent** - Debug-Nachrichten bei schweren Fehlern
- **misc** - Debug-Nachrichten bei Fehlern
- **loud** - Debug-Nachrichten „geschwätzig“

Optionen(2)

set fingerprints "*filename*" – Setzen des Filenamens für Fingerprint-Informationen. *filename* ist der Name der Datei, die die entsprechenden Informationen enthält. Standard ist /etc/pf.os.

set limit *type value* - Setzen von Grenzen für verschiedene Felder *type* im Hauptspeicher (Memory Pool).

states *anzahl* - Anzahl der gleichzeitigen Verbindungen in der State-Tabelle (keep state).

frags *anzahl* - Anzahl der maximalen Verbindungen bei Verkehrsnormalisierung (scrub).

src-nodes *anzahl* - Source Tracking

Beispiel: set limit state 20000

set limit { state 20000, frags 2000 }

Optionen(3)

set loginterface *device* – Festlegen des Interfaces *device* für das der Paketfilter eine Statistik anlegen soll. Die Statistik kann mit pfctl -s info angezeigt werden.

Beispiele:

set loginterface bge0

keine Statistik anlegen

set loginterface none

set state-policy [**if-bound** | **group-bound** | **floating**] -Festlegen der Bindungsregeln für Statusinformationen:

if-bound – Binden an Interface (ppp0)

group-bound – Binden an Interface-Gruppe (ppp)

floating – Keine Bindung an Interface (standard)

Optionen(4)

set optimization [**normal | high-latency | aggressiv | conservativ**] - Festlegen der Optimierungsstrategien des Paketfilters in Abhängigkeit vom Netzwerktyp. Standard ist **normal**.

set timeout *timer value* - Setzen des Timeout Wertes *value* für den Zeitgeber *timer.* Die Werte werden in Sekunden angegeben

interval - Timeout für Statusinformationen und Paketfragmente

frag - Timeout für nicht benutzte Paketfragmente

tcp.[**first**|**opening**|**established**|**closed**| ....] - Spezielle TCP-Timeouts

udp.[**first**|**single**|**multiple**] - Spezielle UDP-Timeouts

icmp.[**first**|**error**] - Spezielle ICMP-Timeouts

other.[**first**|**single**|multiple] - Spezielle Timeouts für andere Protokolle

Optionen(5)

Beispiele für **Optionen:**

```
set optimization normal
# Satelitenverbindung
set optimization high-latency
set timeout interval 200
set timeout tcp.first 10
# schweigsamer Paketfilter
set block-policy drop
# geschwätziger Paketfilter
set block-policy return
```

Struktur des Konfigurationsfiles

Im Konfigurationsfile gibt es folgende sieben Typen von Anweisungen. Mit Ausnahme der Definition von Makros und Tabellen müssen die Anweisungen in folgender Reihenfolge gruppiert sein.

1. Makros
2. Tabellen
3. Optionen
4. **Verkehrsnormalisierung (Scrub)**
5. Queueing
6. Adressumsetzung (NAT und RDR)
7. Paket Filterung

Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de

Verkehrsnormalisierung(1)

Verkehrsnormalisierung (scrub) dient dazu eingehende Pakete auf Standardtreue zu überprüfen und Pakete, die den Standard verletzen, standardkonform umzuschreiben. Dadurch verlassen den Paketfilter nur standardkonforme Pakete. Angriffe, die auf Standardverletzungen basieren, werden dadurch abgewehrt.

Verkehrsnormalisierung wird wie folgt aktiviert:

scrub in all

Alle Pakete werden überprüft. Kann zu Problemen bei NFS und Spielen führen. Trotzdem empfehlenswert. Abhilfe bei NFS-Problemen: scrub in all no-df

scrub [**in** | **out**] [**on** *interface*] [*source* | **all**] {*option*}

Verkehrsnormalisierung(2)

Allgemein:

scrub [**in** | **out**] [**on** *interface*] [*source* | **all**] {*option*}

interface - Interface an dem „scrub“ eingeschaltet werden soll.

source - Quelladresse – nicht empfehlenswert

option - zusätzliche Optionen

no-df - Löschen des „don't fragment“-Bits. Sonst werden solche Paket gelöscht (drop)

random-id - Ersetzen des „IP-Identifikation“-Feldes durch Zufallszahl. Nicht benutzen bei Fragmentierung. Verschleiert das Quell-Betriebssystem.

min-ttl *num* - Minimaler Wert für TTL

max-mss *num* - Maximaler Wert für Segmentgröße

Verkehrsnormalisierung(3)

scrub [**in** | **out**] [**on** *interface*] [*source* | **all**] {*option*}

option - zusätzliche Optionen

fragments reassemble - Eingehende Fragmente puffern und gesamtes Paket übersetzen.

fragment crop - Doppelte oder überlappende Fragmente wegwerfen (drop).

fragment drop-ovl - wie **fragment crop**, aber zukünftige Pakete werden auch weggeworfen (drop)

reassemble tcp - Übersetzung der TCP-Verbindung. Überwachung von IP TTL und Ersetzen von RFC1321 Zeitstempeln durch zufällige Werte (Einschaltzeiten von Hosts werden verschleiert)

Verkehrsnormalisierung(4)

Beispiele:

```
scrub in all
scrub in on bge0 all no-df
# in | out bei reassemble nicht notwendig
scrub on bge0 all reassemble tcp
scrub in all random-id fragment reassemble
```

Struktur des Konfigurationsfiles

Im Konfigurationsfile gibt es folgende sieben Typen von Anweisungen. Mit Ausnahme der Definition von Makros und Tabellen müssen die Anweisungen in folgender Reihenfolge gruppiert sein.

1. Makros
2. Tabellen
3. Optionen
4. Verkehrsnormalisierung (Scrub)
5. Queueing
6. **Adressumsetzung (NAT und RDR)**
7. Paket Filterung

Adressumsetzung NAT, BINAT und RDR(1) (neu match)

NAT (Network Adress Translation) ist ein Weg ein ganzes Netzwerk auf eine einzige IP-Adresse abzubilden. Dies wird typischer Weise dazu benutzt ein internes Netz (Netzwerkadressen 10.0.0.0/8, 172.16.0.0/12 oder 192.168.0.0/16) an das Internet anzuschließen. Der Paket-Filter braucht dafür mindestens zwei Netzwerkadapter!!! net.inet.ip.forwarding muß aktiviert sein. NAT-Regeln stehen vor den Filterregeln, so daß die Filterregeln schon die übersetzten Adressen und Ports in den Paketen „sehen“. Eine NAT-Regel sieht im Allgemeinen wie folgt aus:

nat [**pass**] **on** *interface* [*af*] **from** *src_addr* [**port** *src-port*] **to** \
dst_addr [**port** *dst_port*] **->** *ext_addr* [*pool_type*] [*static-port*]

match out on *interface* [*af*] **from** *src_addr* [**port** *src-port*] **to **
dst_addr [**port** *dst_port*] **nat-to** *nat_ext_addr*

Adressumsetzung NAT,BINAT und RDR(2)

nat [**pass**] **on** *interface* [*af*] **from** *src_addr* [**port** *src_port*] **to** \
dst_addr [**port** *dst_port*] **->** *ext_addr* [*pool_type*] [**static-port**]

pass - Nach der Adressumsetzung die Filterregeln überspringen

on *interface* - Interface von dem das zu übersetzende Paket kommt

af - Adressfamilie (**inet** oder **inet6**)

from [**!**] *src_addr* - Source-Adresse des Paketes, das umgeleitet werden soll. Folgende Adressangaben sind möglich: einzelne IP-Adresse, Block von IP-Adressen (141.20.20.0/24), voll qualifizierter DNS-Name, Name eines Netzwerkes (/etc/network), *interface***:network**, ein Tabellenname, eine Liste, **any**.

"**!**" bedeutet die Verneinung der Adressangabe.

Adressumsetzung NAT,BINAT und RDR(3)

nat [**pass**] **on** *interface* [*af*] **from** *src_addr* [**port** *src_port*] **to** \
dst_addr [**port** *dst_port*] **->** *ext_addr* [*pool_type*] [**static-port**]

port *src_port* - Source-Portadresse des Paketes, das umgeleitet werden soll. Folgenden Angaben sind für *src_port* zulässig: einzelne Portnummer, Name aus /etc/services, Liste von Portnummern, ein Bereich von Portnummern (!=, <, >, >=, >=, <>, ><, :)

to *dst_addr* - Zieladresse des Paketes. Wie *src_addr* spezifizierbar.

port *dst_port* - Zielport des Paketes. Wir *src_port* spezifizierbar.

Adressumsetzung NAT, BINAT und RDR(4)

nat [**pass**] **on** *interface* [*af*] **from** *src_addr* [**port** *src_port*] **to** \
dst_addr [**port** *dst_port*] **->** *ext_addr* [*pool_type*] [**static-port**]

ext_addr - Adresse in die übersetzt werden soll (neue Absenderadresse). Folgende Angaben sind möglich: IP-Adresse, Block von IP-Adressen (141.20.20.50/31), DNS-Name, externes Interface in runden Klammern z.B. (bge1), interfacename:network, Liste

pool_type - Falls für ext_addr eine Liste spezifiziert wurde, wird hier angegeben, wie die Adressen ausgewählt werden: **source-hash**, **bitmask**, **round-robin**, **random**

static-port - Source-Port des Paketes bei TCP und UDP benutzen.

Adressumsetzung NAT, BINAT und RDR(4)

nat [**pass**] **on** *interface* [*af*] **from** *src_addr* [**port** *src_port*] **to** \
dst_addr [**port** *dst_port*] **->** *ext_addr* [*pool_type*] [**static-port**]

Beispiel:

```
ext_if="bge0"
int_if="bge1"
ExtAddr="141.20.23.63"
nat on $ext_if from $int_if:network to any -> $ExtAddr
```

Adressumsetzung NAT, BINAT und RDR(5)

Bidirektionales Mapping (**binat**) erzeugt eine 1:1 Abbildung zwischen einer externen Adresse und einer internen Adresse. Dies ist sinnvoll wenn ein Server aus dem internen Netz vollständig im externen Netz sichtbar werden soll. TCP- und UDP-Ports bleiben beim BINAT erhalten.

binat [**pass**] **on** *interface* [*af*] **from** *src_addr* [**port** *src_port]* \
to *dest_addr* [**port** *dest_port*] -> *ext_addr*

match out on *interface* **from** *src_addr* [**port** *srcort*] \
to *dest_addr [* **port** *dest_port]* **binat-to** *ext_addr*

Parameterbeschreibung: siehe **nat**

Adressumsetzung NAT, BINAT und RDR(5)

binat [**pass**] **on** *interface* [*af*] **from** *src_addr* [**port** *src_port]* \
to *dest_addr* [**port** *dest_port*] -> *ext_addr*

Beispiel:

www_serv_int = „192.168.1.11“
www_serv_ext= „141.20.23.12“
binat on bge0 from $www_serv_int to any -> $www_serv_ext

Adressumsetzung NAT,BINAT und RDR(6)

RDR wird für die Weiterleitung von Paketen bei eingehendem Verkehr vom externen Netz in das interne Netz benötigt, wenn das interne Netz durch NAT vom externen Netz getrennt ist, die IP-Adresse und der Port eines Dienstes im internen Netz ist vom externen Netz aus nicht sichtbar, soll aber dort zur Verfügung gestellt werden, z.B. WWW-Server.

rdr [**pass**] **on** *interface* **proto** *protocoll* **from** *src_addr* [**port** *src_port*]\
to *dst_addr* [**port** *dst_port*] -> *int_addr* [**port** *int_port*]
match in on *interface* **proto** *protocoll* **from** *src_addr* [**port** *src_port*]\
to *dst_addr* [**port** *dst_port*] **rdr-to** *intern_dst_addr*

pass - Filterregeln passieren
on *interface* - Interface, an dem das Paket bereitgestellt wird.
proto *protocoll* - Protokoll-Type

Adressumsetzung NAT,BINAT und RDR(7)

rdr [**pass**] **on** *interface* **proto** *protocoll* **from** *src_addr* [**port** *src_port*]\
to *dst_addr* [**port** *dst_port*] **->** *int_addr* [**port** *int_port*]

from *src_addr* – Quelladresse, wie bei **nat** auch **any** zulässig

port *src_port* – Quellport, wie bei **nat**

to *dst_addr* – IP-Adresse des Ziels, wie bei **nat** auch **any** zulässig

port *dst_port* – Zielport, kann auch ein Bereich von Ports sein

int_addr – interne IP-Adresse, zu der umgeleitet werden soll.

port *int_port* – interner Zielport, kann auch ein Bereich sein. Muß dem Bereich von *dst_port* entsprechen. **port** *anfangsport***:*** ist zulässig.

Adressumsetzung NAT,BINAT und RDR(8)

rdr [**pass**] **on** *interface* **proto** *protocoll* **from** *src_addr* [**port** *src_port*]\
to *dst_addr* [**port** *dst_port*] **->** *int_addr* [**port** *int_port*]

Achtung!!

rdr steht im Regelwerk vor den Filterregeln. Deshalb werden die Pakete nach der Adressumsetzung noch gefiltert!!

Ausnahme: In der **rdr**-Anweisung ist **pass** spezifiziert.

Adressumsetzung NAT,BINAT und RDR(8)

rdr [**pass**] **on** *interface* **proto** *protocoll* **from** *src_addr* [**port** *src_port*]\
to *dst_addr* [**port** *dst_port*] -> *int_addr* [**port** *int_port*]

Beispiele:

```
rdr on bge0 proto tcp from any to any port 80 -> 192.168.1.2
rdr on bge0 proto tcp from 141.20.0.0/15 to any port 80 -> 192.168.1.2
rdr on bge0 proto tcp from any to any port 80 -> 192.168.1.2 port 8000
rdr pass on bge0 proto tcp from any to any port 5000:5500 ->
                                        192.168.1.2 port 3000:*
```

Adressumsetzung NAT,BINAT und RDR(9)

Probleme bei NAT und RDR

ext_if =“bge0“

int_if=“bge1“

www=“192.168.1.2“

rdr on $ext_if proto tcp from any to $ext_if port 80 -> $www port 80

Der WWW-Server ist von Außen erreichbar aber von Innen nicht, da die Pakete von Innen nie am Interface ext_if vorbeikommen. Dafür gibt es mehrere Lösungen:

- Privater DNS-Server für die Auflösung der WWW-Adresse
- Server in ein separates lokales Netz verschieben (DMZ) und dadurch auch gleichzeitig die Sicherheit erhöhen.

Adressumsetzung NAT,BINAT und RDR(10)

- TCP-Proxy: Auf dem Firewall einen lokalen TCP-Proxy z.B. nc oder netcat einrichten, der die Weitervermittlung der Pakete übernimmt:

  inetd.conf:

  ```
  5000 stream tcp nowait nobody /usr/bin/nc -w 20 192.168.1.2 80
  ```

  rdr-Regel:

  ```
  rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> \
          127.0.0.1 port 5000
  ```

- RDR- und NAT-Kombinationen: Mit zusätzlicher Adressübersetzung kann man den Übergang erzielen:

  ```
  rdr on $int_if proto tcp from $int_net to $ext_if port 80 ->  $www
  no nat on $int_if proto tcp from $int_if to $int_net
  nat on $int_if proto tcp from $int_net to $www port 80 -> $int_if
  ```

  NICHT EMPFEHLENSWERT!!!!!

Struktur des Konfigurationsfiles

Im Konfigurationsfile gibt es folgende sieben Typen von Anweisungen. Mit Ausnahme der Definition von Makros und Tabellen müssen die Anweisungen in folgender Reihenfolge gruppiert sein.

1. Makros
2. Tabellen
3. Optionen
4. Verkehrsnormalisierung (Scrub)
5. **Queueing**
6. Adressumsetzung (NAT und RDR)
7. Paket Filterung

Queueing(1)

Queueing und Priorisierung

Der Paketfilter unterstützt die Bildung von verschiedenen Warteschlangen (Queues) für den ausgehenden Verkehr (und nur für den ausgehenden Verkehr). Außerdem ist es möglich den einzelnen Warteschlangen unterschiedliche Prioritäten zuzuweisen. Für die Einordnung in die Warteschlangen gibt es verschiedene Schedulingalgorithmen: Class Base Queueing (CBQ), Priority Queueing (PRIQ), Random Early Detection (RED) und Explicit Congestion Notification (ECN). Für das Aktivieren des Queueing-Mechanismus im Paketfilter sind folgende zwei Anweisungen zuständig:

altq on *interface scheduler* [**bandwith** *bw*] [**qlimit** *qlim*] \
[**tbrsize** *size*] **queue {** *queue_list* **}**

queue *name* [**on** *interface*] **bandwith** *bw* [**priority** *pri*] [**qlimit** *qlim*] \
[**scheduler (** *sched_options* **)**] [**{** *queue_list* **}**]

Queueing(2)

altq on *interface scheduler* [**bandwith** *bw*] [**qlimit** *qlim*] \
[**tbrsize** *size*] **queue {** *queue_list* **}**

Mittels **altq on** wird das Queueing für ein Interfaces *interface* aktiviert - Root-Warteschlange definiert. Gleichzeitig wird durch *scheduler* der benutzte Scheduler (**cbq**, **priq**, **red**, **ecn**) festgelegt. Außerdem müssen die Namen der Child-Warteschlangen, die durch **queue** definiert werden, angegeben werden. *queue_list* ist eine durch Kommas getrennte Liste von Namen. Folgende optionale Parameter sind möglich:

bandwith *bw* - Bandbreite für die Root-Warteschlange in b, Kb, Mb, Gb (Angabe im Bits pro Sekunde) oder nn% der Bandbreite des Interfaces.

qlimit *qlim* - Maximale Paketanzahl in der Warteschlange

tbrsize *size* – Größe des „token bucket regulator“ in Bytes

Queueing(3)

queue *name* [**on** *interface*] **bandwith** *bw* [**priority** *pri*] [**qlimit** *qlim*] \
scheduler (*sched_options* **) {** *queue_list* **}**

queue definiert den Namen und die Eigenschaften einer Child-Warteschlange.

queue *name* – Name der child-Warteschlange (in altq benutzt)

on *interface* – Name des Interfaces (optional)

bandwith *bw* – Bandbreite in b, Kb, Mb, GB oder in nn% der Bandbreite der Root-Warteschlange

priority *pri* – Priorität der Warteschlange. **cbq**: 0-7, **priq** 0-15. Standardwert ist 1. 0 ist die niedrigste Priorität.

qlimit *qlim* – Maximale Anzahl der Pakete, die in der Warteschlange gehalten werden kann

Queueing(4)

queue *name* [**on** *interface*] **bandwith** *bw* [**priority** *pri*] [**qlimit** *qlim*] \
[*scheduler* **(** *sched_options* **)**] [**{** *queue_list* **}**]

scheduler **(** *sched_options* **)** - *scheduler* ist der Name des für die Warteschlange benutzen Schedulers. sched_options sind durch Leerzeichen getrennte Optionen für den jeweiligen Scheduler: default, ecn, borrow red, ...)

{ *queue_list* **}** - Liste von Namen (durch Kommas getrennt) von weiteren child-warteschlangen. Hierdurch ist eine Hierarchie von Warteschlangen möglich.

Queueing(5)

Beispiel:

```
altq on bge0 cbq bandwidth 100% queue { ssh, std }
altq on bge1 cbq bandwidth 500 Mb queue { q1 }
queue ssh bandwidth 25% priority 7
queue std bandwidth 75% priority 5 queue { s1, s2 }
    queue s1 bandwith 50% priority 3 cbq(ecn)
    queue s2 bandwith 50% priority 2 cbq(ecn
queue q1 bandwith 400Kb priority 3
pass in on bge2 from any to any port ssh flags S/SA keep state \
                                    queue ssh
pass in on bge2 from any to any port ftp flags S/SA keep state  \
                                    queue s2
```

Sonstiges(1)

Markierungen

Innerhalb der Filterregeln ist es möglich ein Paket zu markieren und anschließend auf die Markierungen zuzugreifen. Markierungen werden durch **tag** *zeichenkette* in einer Regel an ein Paket angebracht. Mehrere Markierungen sind zulässig. Eine Markierung eines Paketes kann durch **tagged** *zeichenkette* in einer Regel getestet werden.

Beispiele:

```
pass in on bge0 tag bge0_in keep state
pass in on bge1 tag bge1_in keep state
pass out on bge2 tagged bge0_in keep state
block out on bge2 tagged  bge1_in
```

Sonstiges(2)

LOG-Informationen anschauen

LOG-Informationen werden ständig auf das Gerät /dev/pflog0 ausgegeben. Außerdem werden die Informationen über den Syslog-Daemon protokolliert, wenn dies in der /etc/syslog.conf aktiviert ist:

```
local0.info          /var/log/pflog.txt
```

Die LOG-Informationen kann man sich auch wie folgt anschauen:

```
tcpdump -n -e -ttt -i pflog0    # die LOG-Informationen erscheinen live
tcpdump -n -e -ttt -r /var/log/pflog # anschauen der LOG-Informationen,
                  # die durch pflogd aufgezeichnet wurden.
```

Ausgabe filtern:

```
tcpdump -n -e -ttt -i pflog0 host 141.20.20.20
tcpdump -n -e -ttt -i pflog0 port 80
```

Sonstiges(3)

authpf – Benutzer-Shell für authentifizierende Gateways

Der Paketfilter unterstütz mittels der Anker-Technologie die Möglichkeit, spezielle Regeln für einen Nutzer zu laden, der sich zuvor eingeloggt hat. Der Nutzer loggt sich mittels ssh oder telnet ein. Die Benutzershell ist hierbei /usr/sbin/authpf. **authpf** lädt bei einem erfolgreichen Login spezielle Regeln in den Filter, der den Nutzer den Durchgang durch den Paketfilter ermöglicht. Wenn der Nutzer sich ausloggt werden die entsprechenden Regeln entladen. Die Konfiguration von **authpf** erfolgt mittels **/etc/authpf/authpf.conf**. Die nutzerspezifischen Regeln werden in **/etc/authpf/users/$USER/authpf.rules** und **/etc/authpf/authpf.rule**s abgelegt.

Sonstiges(3)

authpf - Dateien

/etc/authpf/authpf.conf - Allgemeines Konfigurationsfile für **authpf**, kann leer sein - muß aber existieren(anchor-Name und table-Name)

/etc/authpf/authpf.rules - Allgemeines Regelfile für den Paket-Filter. Enthält Regeln, die beim Login eines Nutzers geladen werden, wenn kein nutzerspezifisches Regelfile gefunden wird.

/etc/authpf/users/<*user*>**/authpf.rules** - Nutzerspezifisches Regelfile für den Paket-Filter. Enthält Regeln, die beim Login eines Nutzers <*user*> geladen werden und den Durchgang durch den Firewall ermöglichen.

/etc/authpf/banned/<*user*> - gesperrter Nutzer <*user*>.

Sonstiges(3)

authpf - Beispiel(1)

Nutzer tbell soll sich über Port 22 auf dem Rechner 192.168.1.15 hinter dem Firewall einloggen.

- /etc/shells modifizieren: /usr/sbin/authpf hinzufügen
- Nutzer auf Firewall einrichten mit Shell /usr/sbin/authpf
- pf.conf modifizieren und laden pfctl -e -f /etc/pf.conf
- mkdir /etc/authpf
- mkdir /etc/authpf/users
- mkdir /etc/authpf/banned
- touch /etc/authpf/authpf.rules # leeres File reicht
- touch /etc/authpf/authpf.conf # leeres File reicht

Sonstiges(3)

authpf - Beispiel(2)

/etc/pf .conf(1)

```
ext_if="bge0"
int_if="em0"
ext_addr="141.20.20.14"
priv_nets = "{ 127.0.0.0/8, 192.168.1.0/24 }"
tcp_services = "{ 22, 113 }"
icmp_types = "echoreq"
set block-policy return
set loginterface $ext_if
scrub in all
nat on $ext_if from $int_if:network   to any -> $ext_addr
nat-anchor "authpf/*"
rdr-anchor "authpf/*"
binat-anchor "authpf/*"
```


Systems Architecture Group
http://sar.informatik.hu-berlin.de

Sonstiges(3)

authpf - Beispiel(3)

/etc/pf .conf(2)

```
block all
pass quick on lo0 all
block drop in quick on $ext_if from $priv_nets to any
block drop out quick on $ext_if from any       to $priv_nets
# ssh
pass in on $ext_if inet proto tcp from any to ($ext_if) port $tcp_services \
      flags S/SA keep state
pass in inet proto icmp all icmp-type $icmp_types keep state
pass in on $int_if   from $int_if:network   to any keep state
pass out on $int_if   from any to $int_if:network keep state
pass out on $ext_if proto tcp all modulate state flags S/SA
pass out on $ext_if proto { udp, icmp } all keep state
anchor "authpf/*"
```

Sonstiges(3)

authpf - Beispiel(4)

/etc/authpf/users/tbell/authpf.rules

```
external_if = "bge0"
internal_if = "em0"
external_addr = "141.20.20.214"
nat on $external_if from $user_ip to any tag $user_ip -> $external_addr
rdr on $external_if proto tcp from $user_ip to $external_addr port 22 -> \
        192.168.1.15 port 22
pass in log quick on $internal_if proto tcp from $user_ip to any keep state
pass in quick on $internal_if from $user_ip to any
pass in on $external_if proto tcp from $user_ip to 192.168.1.15 port 22 \
        flags S/SA synproxy state
```

Sonstiges(3)

authpf - Beispiel(5)

Auf Rechner amsel:

```
# Freischalten
ssh -l tbell 141.20.20.14   # Wenn diese Sitzung beendet wird,
                            # werden die Firewallregeln für tbell
                            # entladen.
```

Ebenfalls auf Rechner amsel:

```
# Einloggen auf internen Rechner 192.168.1.15:
ssh -l root 141.20.20.214
```

Sonstiges(4)

Firewall Redundanz(1)

Der Paketfilter unterstützt das Protokoll CARP (**C**ommon **A**ddress **R**edundancy **P**rotocoll). CARP ermöglicht es mehreren Rechnern sich eine IP-Adresse zu teilen. Ein Master sendet regelmäßig Kontollpakete. Wenn dies ausbleiben, übernimmt ein anderer Rechner des CARP-Verbundes die Masterfunktion. CARP wird durch die Interfaces **carp***N* realisiert, das durch ifconfig konfiguriert werden muß.

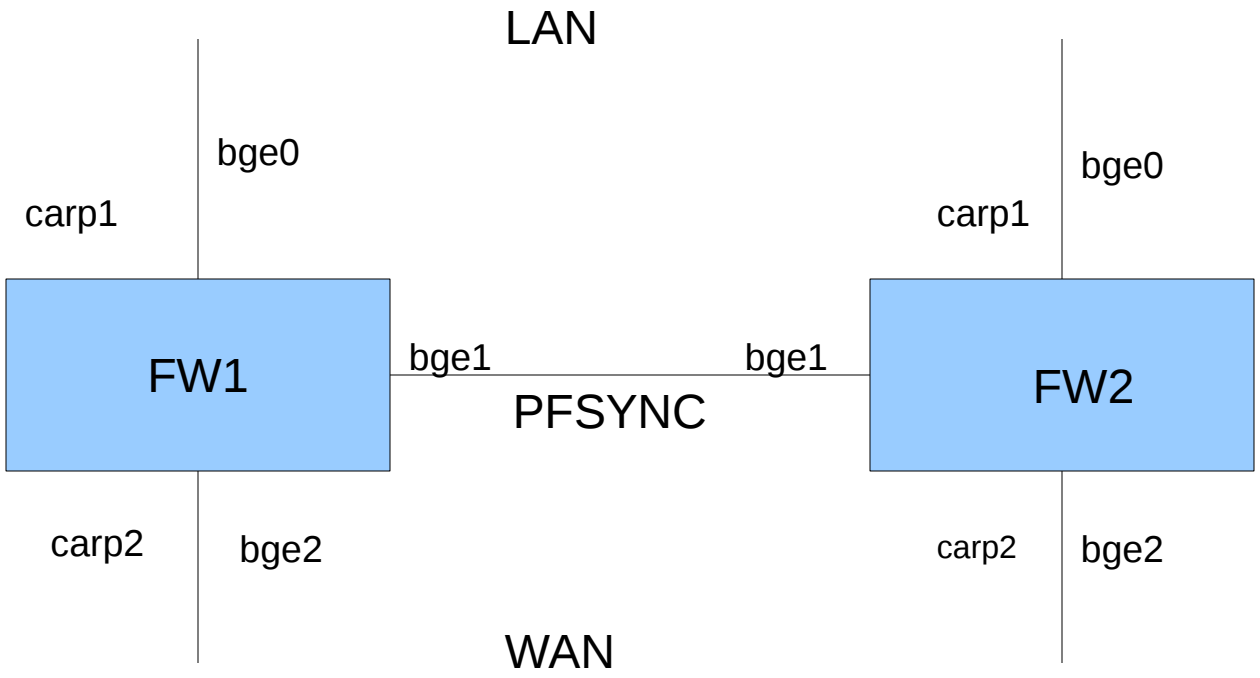
Mit Hilfe von **pfsync** können Firewalls über ein eigenes Netzwerk Statusinformationen austauschen. **pfsync** wird durch das Interface **pfsync***N* realisiert, das durch ifconfig konfiguriert werden muß.

Für **carp***N* und **pfsync***N* müssen pass-Regeln in den Firewallregeln stehen.

Sonstiges(5)

Firewallredundanz(2)

Beispiel:



Sonstiges(5)

Firewallredundanz(3)

Beispiel:

```
# fw1: bge0: 192.168.1.100   - LAN
# fw1: bge1: 10.10.10.1      - PFSYNC
# fw1: bge2: 141.20.23.100   - WAN
# fw2: bge0: 192.168.1.101   -LAN
# fw2: bge1: 10.10.10.2      - PFSYNC
# fw2: bge2: 141.20.23.101   - WAN
# shared LAN IP-Address: carp1:   192.168.1.1 - LAN
# shared WAN IP-Address: carp2:  141.20.23.63 - WAN
```

Sonstiges(6)

Firewallredundanz(4)

fw1 konfigurieren:

```
sysctl -w net.inet.carp.preempt=1
# pfsync aktivieren
ifconfig bge1 10.10.10.1 netmask 255.255.255.0
ifconfig pfsync0 syncdev bge1
ifconfig pfsync0 up
# CARP für LAN aktivieren Master
ifconfig carp1 create
ifconfig carp1 vhid 1 carpdev bge0 pass lanpasswd 192.168.1.1 \
                255.255.255.0
# CARP für  WAN aktivieren
ifconfig carp2 create Master
ifconfig carp2 vhid 2 carpdev bge2 pass wanpasswd 141.20.23.63\
                 255.255.255.0
```

Sonstiges(7)

Firewallredundanz(5)

fw2 konfigurieren:

```
sysctl -w net.inet.carp.preempt=1
ifconfig bge1 10.10.10.2 netmask 255.255.255.0 # pfsync
ifconfig pfsync0 syncdev bge1
ifconfig pfsync0 up
# CARP für LAN aktivieren Slave
ifconfig carp1 create
ifconfig carp1 vhid 1 carpdev bge0 pass lanpasswd  advskew 128 \
                192.168.1.1  255.255.255.0
# CARP für WAN aktivieren Slave
ifconfig carp2 create
ifconfig carp2 vhid 2 carpdev bge2 pass wanpasswd  advskew 128    \
                141.20.23.63    255.255.255.0
```

Beispiel(1)

```
        /etc/pf.conf:
#       $OpenBSD: pf.conf,v 1.27 2004/03/02 20:13:55 cedric Exp $
#  Aenderungen:
#  25.01.2005   JPB     - Umlenkung 2222 nach ssh(22) auf merkur(192.168.4.199)
#  12.07.2005   JPB     - sarsvn.informatik.hu-berlin.de
#                         141.20.23.60(tcp/udp:3690) -->  192.168.2.14
##############################
#   Hardwareinterfaces   #
##############################
#       141.20.23.63 - Internet
ext_if="bge0"
#       192.168.2.1 - DMZ
int_if="bge1"
#       192.168.3.1 - internes Netz
int_ifm3="em0"
```

Beispiel(2)

```
#        Macros                          #
tcp_services = "{ 22, 113 }"
icmp_types = "echoreq"
ExtAddr="141.20.23.63"
ExtAddrExch="141.20.23.61"
ExtAddrSvn="141.20.23.60"
priv_nets = "{ 127.0.0.0/8, 192.168.2.0/24, 192.168.3.0/24, 192.168.10.0/24, \
                192.168.11.0/24 }"
#  interne Server nach aussen sichtbar
www = "192.168.2.2"
dns = "192.168.2.3"
mail = "192.168.2.5"
svn = "192.168.2.14"
```

Beispiel(3)

```
#      VPN   config Makros       #
# Mitarbeiter
VPN0="tun0"
vpn_port0 = "{ 5000 }"
vpn_net0 = "{ 192.168.10.0/24 }"
# Studenten
VPN1="tun1"
vpn_port1 = "{ 5001 }"
vpn_net1 = "{ 192.168.11.0/24 }"
#         Options          #
set block-policy return
set loginterface $ext_if
# scrub
scrub in all
```

Beispiel(4)

```
nat on $ext_if from $int_if:network    to any -> $ExtAddr
nat on $ext_if from $int_ifm3:network to any -> $ExtAddr
nat on $ext_if from $vpn_net0 to any -> $ExtAddr
rdr on $ext_if    proto tcp from any to $ExtAddr      port  25 -> $mail
rdr on $int_ifm3 proto tcp from any to $ExtAddr      port  25 -> $mail
rdr on $ext_if    proto tcp from any to $ExtAddr      port  80 -> $www
rdr on $int_ifm3 proto tcp from any to $ExtAddr      port  80 -> $www
rdr on $ext_if    proto tcp from any to $ExtAddr      port 443 -> $www
rdr on $int_ifm3 proto tcp from any to $ExtAddr      port 443 -> $www
rdr on $ext_if    proto tcp from any to $ExtAddrExch port 443 -> $mail
rdr on $int_ifm3 proto tcp from any to $ExtAddrExch port 443 -> $mail
rdr on $ext_if    proto tcp from any to $ExtAddrSvn  port 3690 -> $svn
rdr on $int_ifm3 proto tcp from any to $ExtAddrSvn  port 3690 -> $svn
rdr on $ext_if    proto udp from any to $ExtAddrSvn  port 3690 -> $svn
rdr on $int_ifm3 proto udp from any to $ExtAddrSvn  port 3690 -> $svn
rdr on $ext_if proto tcp from any to any port 53 -> $dns
rdr on $ext_if proto udp from any to any port 53 -> $dns
rdr on $int_if    proto tcp from any to any port 21 -> 127.0.0.1 port 8021
rdr on $int_ifm3 proto tcp from any to any port 21 -> 127.0.0.1 port 8021
```

Beispiel(5)

```
#       Firewall config                         #
block all
pass quick on lo0 all
block drop in quick on $ext_if from $priv_nets to any
block drop out quick on $ext_if from any       to $priv_nets
#           VPN0 rules                          #
pass in on $ext_if inet proto udp from any to ($ext_if) port $vpn_port0
pass in quick on $VPN0  all
pass out quick on $VPN0  all
pass in  on $VPN0 from $vpn_net0 to any      keep state
pass out on $VPN0 from any       to $vpn_net0 keep state
#           VPN1 rules                          #
pass in on $ext_if inet proto udp from any to ($ext_if) port $vpn_port1
pass in quick on $VPN1  all
pass out quick on $VPN1  all
pass in  on $VPN1 from $vpn_net1 to any      keep state
pass out on $VPN1 from any      to $vpn_net1 keep state
```

Beispiel(6)

```
#          SSH rules        #
pass in on $ext_if inet proto tcp from any to ($ext_if) port $tcp_services \
                    flags S/SA keep state
#          ftp rules        #
pass in on $ext_if inet proto tcp from any to $ext_if port >= 49152 \
                   flags S/SA keep state
pass out quick on $ext_if inet proto tcp from any to any port >= 49152 keep state
pass in on $ext_if   proto tcp from any to $mail port  25 flags S/SA synproxy state
pass in on $int_ifm3 proto tcp from any to $mail port  25 flags S/SA synproxy state
pass in on $ext_if   proto tcp from any to $mail port 443 flags S/SA synproxy state
pass in on $int_ifm3 proto tcp from any to $mail port 443 flags S/SA synproxy state
pass in on $ext_if   proto tcp from any to $www  port  80 flags S/SA synproxy state
pass in on $int_ifm3 proto tcp from any to $www  port  80 flags S/SA synproxy state
pass in on $ext_if   proto tcp from any to $www  port 443 flags S/SA synproxy state
pass in on $int_ifm3 proto tcp from any to $www  port 443 \
                        flags S/SA synproxy state
```

Beispiel(7)

```
#          svn rules        #
pass in on $ext_if   proto tcp from any to $svn port  3690 flags S/SA synproxy state
pass in on $int_ifm3 proto tcp from any to $svn port  3690 flags S/SA synproxy \ state
pass in on $ext_if   proto udp from any to $svn port  3690 keep state
pass in on $int_ifm3 proto udp from any to $svn port  3690 keep state
#          dns rules        #
pass in on $ext_if proto tcp from any to $dns port 53 flags S/SA synproxy state
pass in on $ext_if proto udp from any to $dns port 53 keep state
#          ssh rules        #
pass in on $ext_if proto tcp from any to $merkur port 22 flags S/SA synproxy state
```

Beispiel(8)

```
pass in inet proto icmp all icmp-type $icmp_types keep state

pass in on $int_if   from $int_if:network   to any keep state
pass in  on $int_ifm3 from $int_ifm3:network to any keep state
pass out on $int_if   from any to $int_if:network keep state
pass out on $int_ifm3 from any to $int_ifm3:network keep state

pass out on $ext_if proto tcp all modulate state flags S/SA
#          icmp rules         #
pass out on $ext_if proto { udp, icmp } all keep state
```


Humboldt Universität Informatik
Systems Architecture Group
http://sar.informatik.hu-berlin.de