die von der Umstellung betroffenen Programme im Sourcecode vorliegen.
Einer der bekanntesten Vertreter dieser Gattung ist **SOCKS**. In seiner neuesten Version 5 kann das Protokoll während des Verbindungsaufbaus den Benutzer zusätzlich authentisieren, z.B. über seine Kennung und sein Passwort. Dem leidigen Problem der Anpassung der Client Software kann über das kostenlose Programm SocksCap von NEC [soc 02] begegnet werden. Es arbeitet zwischen den vorhandenen Internet-Anwendungen und dem TCP/IP Stack (z.B. bei Windows: WinSock). Dort unterbricht SocksCap alle Aufrufe und konvertiert sie zur Laufzeit in das SOCKS-Protokoll. Programme werden so durch einfaches Drag-and-Drop in den Fenstercontainer **socksified**.

9.1.1 Die Plug Gateways beim Firewall Tool Kit

Das im letzten Kapitel beschriebene Firewall Tool Kit bietet neben den reinrassigen Proxies auch die Möglichkeit, mit Hilfe des sogenannten Plug Gateways, `plug-gw`, eines **generischen Proxies**, TCP-Dienste freizugeben, für die kein Proxy existiert. Die Handhabung dieses Plug Gateways ist relativ einfach. In der `/etc/services` wird der für diese Plug-Verbindung vorgesehene Port definiert und der Dienst in der `/etc/inetd.conf` gestartet:

```
#
# /etc/services
#
plug-12345    12345/tcp     # Plug fuer Telnet zur 10.1.1.1 und 10.2.2.2
#


# /etc/inetd.conf
#
plug-12345 stream tcp nowait root /usr/local/etc/plug-gw plug-gw plug-12345
#
```

Normalerweise realisiert ein Plug Gateway eine n-zu-1-Verbindung. Wenn aber genau abgegrenzte Source-IP-Bereiche auf unterschiedliche Destinations zugreifen wollen, kann dies über den gleichen Port abgewickelt werden:

```
plug-12345: port plug-12345 192.168.1.0:255.255.255.0 -plug-to 10.1.1.1
                                                                -port 23
plug-12345: port plug-12345 192.168.2.0:255.255.255.0 -plug-to 10.2.2.2
                                                                -port 23
```

Erfolgt nun der Verbindungsaufbau von einem Rechner aus dem Netz `192.168.1.0/24` mit `telnet sectest.muc.meinedomain.de 12345`, so wird die Anfrage zur `10.1.1.1` Port 23 weitergeleitet. Im Logfile `/var/log/messages` sieht das dann so aus

```
Jun 13 14:41:53 sectest plug-gw[29102]: connect host=unknown/192.168.1.1
                    destination=10.1.1.1/23
Jun 13 14:42:01 sectest plug-gw[29102]: disconnect host=unknown/192.168.1.1
                    destination=10.1.1.1/23 in=187 out=103 duration=8
```

Nichterlaubte Verbindungen werden so mitprotokolliert:

```
Jun 13 14:39:51 sectest plug-gw[29097]: deny host=unknown/192.168.100.100
                                                     service=plug-12345
```

Kommt die Anfrage von einem Client im Netz `192.168.2.0/24`, so wird die Anfrage laut Konfiguration zur `10.2.2.2` weitergeleitet.

9.1.2 SOCKS

[Bart 01] Das SOCKS-Protokoll ist zwischen der Anwendungsschicht und der Transportschicht angesiedelt. Für den Verbindungsaufbau über Sockets verwendet ein normaler Client die entsprechenden Funktionen aus der C-Library, wie etwa `connect` und `bind`. Mit der SOCKS-Library werden die für die Netzwerkkommunikation benötigten Routinen ausgetauscht. Damit ist zwar eine Anpassung des Clients nötig, diese ist aber sehr einfach durchführbar, da nicht tiefer in die Anwendungslogik eingegriffen werden muß.
Die erste frei verfügbare Implementation kam von einer Tochter von NEC USA Inc., die heute unter Networking Systems Laboratory (NWSL) firmiert. Der erste offene Standard, das **SOCKS V4 protocol**, wurde über die Implementierung von NEC definiert. Die Weiterentwicklung Version 5 wurde über RFCs standardisiert:

- RFC 1928: Basis für SOCKS V5 [LGL+ 96]
- RFC 1929: Username/Password Authentication für SOCKS V5 [Leec 96]
- RFC 1961: GSS-API, Generic Security Service Application Programming Interface [McMa 96]

SOCKS V4 im Vergleich zu SOCKS V5

SOCKS V4 als erstes verfügbares SOCKS-Protokoll arbeitet ausschließlich auf TCP Basis. Entscheidungen werden über die IP- und TCP-Header getroffen (Source-IP und Source-Port, Destination-IP und Destination-Port). Die einzige mögliche Userauthentisierung ist über `ident`[49] gegeben.
In der ursprünglichen Version mußte der Client selber für die Namensauflösung per DNS sorgen. Die Internetgemeinde hat später eine Erweiterung für SOCKS V4 definiert, die es dem Client ermöglicht, dem SOCKS-Server einen nicht aufgelösten Namen mitzuteilen und

[49] RFC 1413 [John 93]

den DNS-Lookup vom SOCKS-Server durchführen zu lassen.
Mit **SOCKS V5** wurden neben der Standardisierung auch Erweiterungen zu SOCKS V4 definiert, die eine Reihe von Unzulänglichkeiten von SOCKS V4 beheben sollten:

- **Authentication Methode Negotation**: Der Server teilt dem Client mit, welche Authentisierungsmethode der Server wünscht. Kann der Client keine der möglichen Methoden verwenden und sich nicht authentifizieren, wird die Verbindung unterbrochen. Gegenüber einem einfachen, generischen Proxy kann hier zusätzlich eine Userauthentisierung stattfinden.
- **Address Resolution Proxy**: Mit SOCKS V5 ist die DNS-Namensauflösung im Protokoll definiert. D.h. der SOCKS-Server muß gleichzeitig als DNS-Proxy agieren können.
- **UDP Proxy**: UDP ist nun im Protokoll enthalten, allerdings gibt es Unterschiede zu TCP. Da UDP ein verbindungsloses Protokoll ist, beschreibt der Proxy-Circuit nur Endpunkte einer Verbindung für das Senden und Empfangen von Daten. Die Anwendungsdaten einschließlich der Destinationadresse werden eingebettet.
- **Generic Security Service Application Programming Interface**: Das GSS-API ermöglicht starke Authentifikation und die Möglichkeit, Virtuel Privat Networks (VPN) über SOCKS zu implementieren.

Implementierung von SOCKS

Bis jetzt haben wir uns nur über das SOCKS-Protokoll unterhalten. Um damit arbeiten zu können, ist die konkrete Implementierung nötig.
Neben dem SOCKS-Package von NEC [soc 02] gibt es auch eine freie Implementierung von SOCKS V5 von Inferno Nettverk A/S mit dem Namen **Dante** [dan 02].
Dante unterstützt derzeitig folgende Standards:

- SOCKS-Protokoll V4
- RFC 1928 ohne GSS-API
- RFC 1929: User-ID und Passwort Authentisierung für SOCKS V5
- msproxy: Diese Erweiterung macht es möglich, Unix Clients über einen MS-Proxy-Server zu verwenden. Dabei ist allerdings keine Userauthentisierung möglich und es wird nur TCP unterstützt.

Dante-Server

Die Konfigurationsdatei ist `/etc/sockd.conf`. Die Konfiguration kann in folgende Abschnitte unterteilt werden: **Servereinstellungen**, **Regeln für die Verbindungen zwischen Client und Server** und **Regeln für die Verbindungen zwischen Client und**

Zielserver.
Hier ist ein Konfigurationsbeispiel aus [swo 02] und [Bart 01] für den Server in der `/etc/sockd.conf` dargestellt, wobei `0.0.0.0/0` für eine beliebige IP-Adresse steht:

```
logoutput: syslog /var/log/dante

internal: 192.168.2.2 port = 1080
external: 192.168.2.2

method: username none #rfc931

user.privileged: sockd
user.notprivileged: sockd

compatibility: sameport

# -----------------------------------------------------------
# welche Clients

# (1) erlaubte Clients
client pass {
        from: 192.168.1.0/24 port 1-65535
        to: 0.0.0.0/0
        method: none
}

# (2) keine sonstigen Clients erlaubt
client block {
        from: 0.0.0.0/0
        to: 0.0.0.0/0
        log: connect error
}

# -----------------------------------------------------------
# welche Dienste

# (3) loopback am Server
block {
        from: 0.0.0.0/0 to: 127.0.0.0/8
        log: connect error
}

# (4) alle Bind Requests
```

```
block {
        from: 0.0.0.0/0 to: 0.0.0.0/0
        command: bind
        log: connect error
}

# (5) Rueckkanal fuer bestimmte Antwortpakete
pass {
        from: 0.0.0.0/0 to: 192.168.1.0/24
        command: bindreply udpreply
        log: connect error
}

# (6) Namensaufloesung fuer alle internen Clients
pass {
        from: 192.168.1.0/24 to: 0.0.0.0/0 port = domain
        log: connect error
        method: none
}

# (7) alle internen Clients duerfen surfen
pass {
        from: 192.168.1.0/24 to: 0.0.0.0/0 port = http
        log: connect error
        method: none
}

# (8) nur ein Client darf mehr ...
pass {
        from: 192.168.1.9/32 to: 0.0.0.0/0
        protocol: tcp udp
}

# (9) Ausputzer: alles sonstige sperren
block {
        from: 0.0.0.0/0 to: 0.0.0.0/0
        log: connect error
}
```

Hier eine Aufschlüsselung der Einstellungsmöglichkeiten:

- `logoutput`: Hier wird angegeben, wohin die Logeinträge geschrieben werden sollen. Möglich ist `stdout`, `stderr` oder `syslog`. Der Wert `syslog` kann mit einer Erweiterung ergänzt werden, unter der die Protokollierung erzeugt werden soll. Dies kann z.B. `syslog/daemon` sein. Kombinationen sind möglich und werden durch ein Leerzeichen getrennt. Es kann auch eine genaue Spezifikation des Logfiles erfolgen: `syslog /var/log/dante`.

- `internal`: gibt die IP-Adresse an, auf die der Proxy hört. Gehen Anfragen an eine andere IP-Adresse, werden diese ignoriert.
- `external`: ist die IP-Adresse, mit der der Server seine Anfragen nach außen stellt.
- `method`: definiert global die erlaubten Authentifikationsmethoden für den Client. Standard hierfür sind `username`, `rfc931`[50] und `none`.
- `user.privileged`: In diesem Fall wird der User definiert, mit dessen Berechtigungen privilegierte Operationen ausgeführt werden. Hier ist das die ID des unprivilegierten Users `sockd`. Das bietet zwar eine höhere Sicherheit, führt aber auch dazu, daß keine Operationen ausgeführt werden dürfen, die einen privilegierten User erfordern. Dieser User kann `root` sein, es kann aber auch ein spezieller User mit den benötigten Rechten angelegt werden.
- `user.notprivileged`: legt den Usernamen für die Operationen fest, die keine besonderen Rechte erfordern.
- `compatibility`:
  - `sameport`: dadurch verwendet der Proxy für die ausgehende Verbindung den gleichen Port, den der Client für seine Anfrage an den SOCKS-Server verwendet hat. Das kann bei Sourceports aus dem priviligierten Bereich Probleme verursachen. Diese Einstellung ist Standard.
  - `reuseaddr`: Bei Problemen kann diese Einstellung verwendet werden. Da aber der genaue Grund, warum die Probleme dann nicht mehr auftreten, nicht genau bekannt ist, sollte diese Einstellung nur im Notfall Anwendung finden [Bart 01].
- `connecttimeout`: definiert die Zeitspanne, die zwischen Authentifikation und Verbindungsaufbau erlaubt ist. Ist dieser Wert 0, so wird nach der Authentifikation ohne Begrenzung auf einen Request gewartet. Ist hier ein positiver Wert eingetragen, so wird nach diesem Timeout die Verbindung geschloßen, wenn kein Request erfolgt ist. Dabei muß aber die Authentifikationsmethode berücksichtigt werden. Wird als Wert `none` eingetragen, so muß sich der Client nicht authentifizieren und der erste Request kann schnell erfolgen. Setzt man aber `rfc931`, muß bedacht werden, daß nicht alle Clients `ident` unterstützen und zuerst der Timeout für die `ident` Anfrage miteingerechnet werden muß.

[50] ident [John 85]

- `iotimeout`: gibt die Zeitspanne in Sekunden an, die zwischen dem Ende der Datenübertragung und dem endgültigen Verbindungsabbau liegen soll. Beim Wert 0 erfolgt kein Verbindungsabbruch.

Diese Einstellungsmöglichkeiten sind global für die Konfiguration des Dante-Servers nötig. Nun muß man sich Gedanken machen, wie die Zugriffe von Clients zu behandeln sind. Dazu gibt es zwei sogenannte Regelklassen: Die Regeln, die den Verbindungsaufbau vom Client zum SOCKS-Server betreffen und die Regeln, die die weiteren Verbindungen zu den Zielservern festlegen.
Die Punkte (1) und (2) im obigen Beispiel definieren die **Clientregeln**, die festlegen, welche Clients sich zum Server verbinden dürfen und was geblockt werden soll. Die Clientregeln beginnen mit dem Schlüsselwort `client`. Das angefügte `pass` bedeutet, daß die so definierten Pakete passieren dürfen. Bei `block` werden die so angegebenen Pakete geblockt, der Verbindungsaufbau wird untersagt. `from` gibt die Client-IP-Adresse an und mit `to` wird die IP-Adresse des Proxyservers angegeben. `port` legt den Portbereich fest. Das kann auf unterschiedliche Arten geschehen: neben

```
>, >= , =, !=, <, <=
```

versteht der Parser auch `neq`, `eq`, `ge` und so weiter.
Mit `log` wird festgelegt, welche Informationen ins Logfile geschrieben werden. Weitere Möglichkeiten können mit `man 5 sockd.conf` herausgefunden werden.
Die Punkte (3) bis (9) stellen nun die eigentlichen **Verbindungsregeln** dar. Mit der Regel (3) werden alle Verbindungsversuche zu den Loopback-Adressen unterbunden. Das ist eine reine Vorsichtsmaßnahme, da normalerweise von außen nicht auf diese Adressen zugegriffen werden kann.
Mit (4) werden alle Bind-Versuche unterbunden. In diesem Zusammenhang ist nicht der DNS-Server BIND gemeint, sondern eine Routine des Betriebssystems, mit dem ein Name an einen Socket gebunden werden kann (`man 2 bind`). Mit der Routine `socket` (`man 2 socket`) wird ein neuer Socket erzeugt, über den die Kommunikation im Netz abgewickelt wird. Um diesen Socket zu benutzen, muß die Verbindung aufgebaut werden. Der Client benutzt die Routine `connect` (`man 2 connect`). Wenn es sich um einen Client handelt, ist damit alles erledigt. Ist die Maschine aber ein Server, der auf eingehende Verbindungen wartet, so muß der neue Socket einem Namensraum zugeordnet, also an ihn gebunden werden, bevor die Routinen `listen` und `accept` auf eingehende Verbindungen angewendet werden können. Dazu wird die Routine `bind` benötigt. In diesem Zusammenhang steht auch die Zeile `command: bind`. Mit der Regel (4) wird unterbunden, daß ein Client ein `bind` durchführt und damit einen Socket für eingehende Verbindungen öffnet. Das kann zum Beispiel bei aktivem FTP interessant werden, da hier der Client einen eigenen Socket öffnet, die Portnummer dem Server mitteilt und auf diesem Socket die eingehende Datenverbindung erwartet. `bind` ist verbunden mit dem selbstständigen Öffnen von Sockets für eingehende Verbindungen durch den Client. Eine Alternative stellt in diesem Fall der passive FTP dar, bei dem sowohl die FTP- wie auch die FTP-Data-Verbindung vom Client ausgehen. Hier ein kurze Liste der möglichen Werte für `command`:

- `bind`: eigenmächtiges Öffnen eines Sockets durch einen Client für eingehende Verbindungen.
- `bindreply`: Ist ein ausgehender `bind` erlaubt, so muß auch ein eingehender `bindreply` erlaubt sein.
- `connect`: normaler Verbindungsaufbau des Clients.
- `udpassociate` und `udpreply`: Da es bei UDP keinen Verbindungsauf- bzw. -abbau gibt wie bei TCP, muß hier mit `udpassociate` ein Paket erfaßt werden, das eine UDP Verbindung startet. Mit `udpreply` werden alle Pakete behandelt, die Antworten auf UDP Anfragen darstellen könnten.

In dem obigen Beispiel ist bislang keine Userauthentisierung vorgenommen worden. Möchte man z.B. die HTTP Anfragen von `192.168.1.0/24` an die Domain `.test.de` mit Userauthentisierung einrichten, so würde das so aussehen:

```
pass {
       from: 192.168.1.0/24 to: .test.de port = http
       log: connect error
       method: username
}

block {
       from: 0.0.0.0/0 to: .test.de port = http
       log: connect error
}
```

Kann sich nun der Anwender nicht korrekt authentisieren oder kommt die HTTP Anfrage aus einem anderen Netz, so wird der Request geblockt. Das dabei verwendete Authentisierungsfile ist die `/etc/passwd`.
Möchte man seinen internen Anwendern alles nach außen erlauben, so wird das durch:

```
pass {
       from: 192.168.1.0/4 to: 0.0.0.0/0
       protocol: tcp udp
}
```

ermöglicht.
Das Dante-Startscript heißt `/etc/init.d/sockd` und hat die entsprechenden Verlinkungen in die Runlevelverzeichnisse. Startet der Dante-Server nicht korrekt, so kann mit dem Aufruf des Binaries im Debugmodus der Fehler festgestellt werden:

```
/usr/sbin/sockd -d
```

SOCKS-Client

In einigen Browsern, z.B. Netscape, ist bereits der SOCKS-Support eingetragen, so daß sie ohne Anpassungen mit einem SOCKS-Server zusammenarbeiten können. Viele andere Programme sind aber nicht von vornherein SOCKS-fähig. Es gibt zwei Möglichkeiten, einen Client SOCKS-fähig zu machen. Die Erste ist, das Programm neu zu übersetzen. Da diese Methode sehr umständlich ist, wird sie nur verwendet, wenn der benötigte Client nicht mit der zweiten Methode socksifiziert werden kann. Die zweite Möglichkeit ist, einen dynamisch gelinkten Client davon zu überzeugen, vor allen anderen Libraries eine SOCKS-Library zu laden, die die Netzwerkaufrufe abfängt und socksifiziert. Beim Dante-Client wird hierzu das bereits fertige Programm `/usr/bin/socksify` mitgeliefert, das beim Clientstart die `libdsocks.so` lädt und über die Environment-Variable `LD_PRELOAD` die SOCKS-Libraries permanent vor die eigentlichen Netzwerkcalls setzt.
Der FTP-Client wird dann z.B. so aufgerufen:

```
socksify ftp
```

Wurde die SOCKS-Library nicht in der Environment-Variable `LD_PRELOAD` vor die eigentlichen Aufrufe gestellt, so muß jede Verbindung gesondert socksifiziert werden.
Möchte man nun den Dante-Client konfigurieren, so geschieht das über das Konfigurationsfile `/etc/socks.conf`.
Hier ein Konfigurationsbeispiel für den Dante-Client in `socks.conf` [Bart 01]:

```
logoutput:       syslog
resolveprotocol: udp    # default

# (1) direkte Verbindung zu den Nameservern
# route {
#        from: 192.168.1.0/24
#        to: 193.101.111.0/24 port = domain
#        via: direct
# }

# (2) loopback = hostinterne Verbindungen
route {
        from: 0.0.0.0/0
        to: 127.0.0.0/8
        via: direct
        command: connect udpassociate
        # everything but bind, bind confuses us.
}

# (3) Internes Netzwerk
route {
```

```
        from: 0.0.0.0/0
        to: 192.168.1.0/24
        via: direct
}

# (4) Sonstiges --> SOCKS-Server
route {
        from: 0.0.0.0/0
        to: 0.0.0.0/0
        via: 192.168.2.2
        port = 1080
        protocol: tcp udp
        proxyprotocol: socks_v4 socks_v5
        method: none
}

# (5) dito, nur mit Namensangabe
route {
        from: 0.0.0.0/0
        to: .
        via: 192.168.2.2
        port = 1080
        protocol: tcp udp
        proxyprotocol: socks_v4 socks_v5
        method: none
}
```

Das Konfigurationsfile besteht ebenso wie das Serverkonfigurationsfile aus zwei Abschnitten: zuerst die **allgemeinen Einstellungen des Clients** und dann die einzelnen **Routingregeln** mit den Klassen der zum Proxy geschickten Pakete. Die Option `logoutput` ist analog zur Serverkonfiguration. `resolveprotocol` gibt an, welches Protokoll zur Namensauflösung benutzt werden soll. Standard ist natürlich UDP. Die Regel (1) würde Verwendung finden, wenn die Namensserver im Netz `193.101.111.0/24` direkt anzusprechen wären. Welche Umsetzung hier Sinn macht, ist von Fall zu Fall zu unterscheiden. Mit der Regel (2) wird festgelegt, daß Anfragen von der lokalen Maschine an das Loopback Interface nicht über den Proxy zu schicken sind. Die dritte Regel legt fest, daß interne Server direkt anzusprechen sind und mit Regel (4) wird alles andere zum SOCKS-Proxy auf Port 1080 geschickt. Die Regel (5) ist analog zur Regel (4), nur daß hier Namen anstatt IP-Adresse übergeben werden und die Namensauflösung auf dem SOCKS-Proxy erfolgt.
Wird nun vom Dante-Server als Authentisierungsmethode `username` verlangt, so muß beim Dante-Client nicht nur `username` als zu übergebende Methode festgelegt sein, sondern es müssen noch folgende Umgebungsvariablen für den User festgelegt werden:

```
SOCKS_USERNAME=oberhait
SOCKS_PASSWORD=hallo
```

wobei die User-ID und das Passwort mit den Einträgen in der `/etc/passwd` auf dem Dante-Server übereinstimmen müssen. Damit hat man dann erreicht, daß nur bestimmte User auf einem Clientsystem den Dante-Server verwenden dürfen. Wie bei Telnet und FTP geht auch in diesem Fall das Passwort unverschlüsselt über das Netz.

9.2 Firewallarchitekturen

Firewalls lassen sich in fast beliebiger Weise aus den beiden Funktionen Paketfilter und Porixies zusammenbauen, so daß fast jede Sicherheitsanforderung erfüllt werden kann.

- **Position des Firewalls**: Ein Firewall sollte den Übergang zum unsicheren Netz darstellen, d.h. es sollten so viele Rechner wie möglich im Schutzbereich des Firewalls sein. Müssen interne Bereiche ebenfalls unterschiedlich geschützt werden, so empfiehlt es sich, einen weiteren Firwall für diese Belange intern zu positionieren. Es sollte keine weiteren ungeschützen Übergänge zum Internet geben.
- **Redundanz**: In einer Firewallanordnung sollte eine Staffelung unterschiedlicher Systeme erfolgen. Dadurch können eventuelle Schwachstellen in Hard- und Software nicht so leicht ausgenutzt werden.

Nachfolgend sollen einige gängige Firewall-Architekturen vorgestellt werden [Plat 02].

9.2.1 Einstufige Firewallarchitektur

Paketfilter

Die einfachste Art eines Firewallaufbaus ist der Einsatz eines einzelnen Paketfilters. Das ist ausreichend, wenn keine Proxyfunktionalitäten benötigt werden. Der ideale Einsatzort für diese Art der Firewallarchitektur ist in internen Netzen, wenn einzelne Netzbereiche von einander getrennt werden sollen. In Richtung Internet ist diese Anordnung zu unsicher, da nur die Einschränkung auf IP-Adressen und Ports möglich ist. Eine Überprüfung des Protokolls und der Paketinhalte ist hier nicht möglich. Außerdem ist der Weg ins interne Netz komplett offen, sobald der Paketfilter durch einen Angreifer kompromitiert wird.

Application Level Gateway

Ein Application Level Gateway, der als erster oder einziger Rechner aus dem Internet erreichbar ist, wird auch als **Bastion-Host**, kurz **Bastion**, bezeichnet. Dadurch müssen diese Rechner besonders gut gegen Angriffe geschützt sein. In der Regel handelt es sich bei Application Level Gatesways um Rechner mit Betriebssystem und zusätzlicher Firewallsoftware, weshalb hier das Gefahrenpotential durch Programmierfehler deutlich erhöht ist.

Eine Architektur mit **Dualhomed-Host** wird um einen Host herum aufgebaut, der über mindestens zwei Netzwerkschnittstellen verfügt (siehe auch Abbildung 72). Ein solcher Host ist als Router zwischen den Netzen einsetzbar, die an die Schnittstellen angeschlossen sind. Er kann dann IP-Pakete von Netz zu Netz routen. Für diese Firewall-Architektur muß diese Routingfunktion jedoch deaktiviert werden. IP-Pakete werden somit nicht direkt von dem einen Netz (dem Internet) in das andere Netz (das interne, geschützte Netz) geroutet. Systeme innerhalb des Firewalls und Systeme außerhalb (im Internet) können jeweils mit dem Dualhomed-Host, aber nicht direkt miteinander kommunizieren. Der IP-Verkehr zwischen ihnen wird vollständig blockiert. Die Netzarchitektur für einen Firewall mit Dualhomed-Host ist denkbar einfach: der Dualhomed-Host sitzt in der Mitte, wobei er mit dem Internet und dem internen Netz verbunden ist.

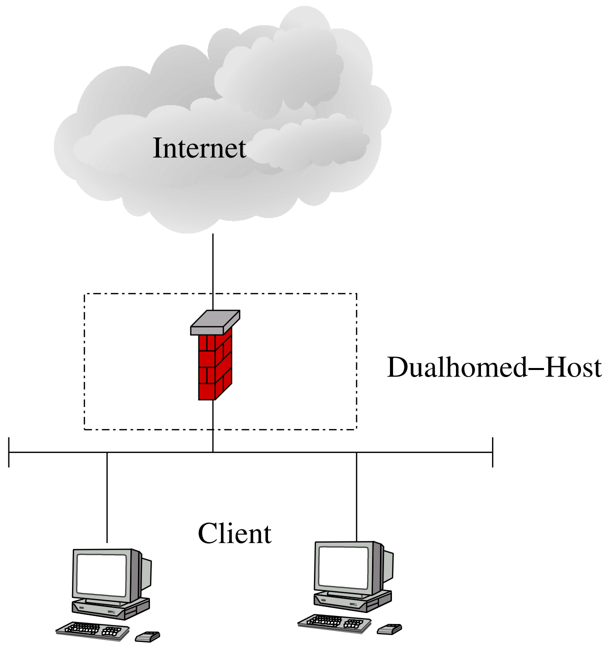

Abbildung 72: Dualhomed Architektur

Ein Dualhomed-Host kann Dienste nur anbieten, indem er entsprechende Proxies (Stellvertreter) oder Gateways einsetzt. Es ist jedoch auch möglich, direkte Nutzerzugriffe zu gestatten (Sicherheitsrisiko).

9.2.2 Architektur mit überwachtem Host

Die **screened host architecture** bietet Dienste auf einem Rechner an, der nur an das interne Netz direkt angeschlossen ist, wobei ein trennender Router verwendet wird. Der Bastion-Host befindet sich im inneren Netz. Auf dem Router verhindern Paketfilterregeln das Umgehen des Bastion-Host (Abbildung 73).

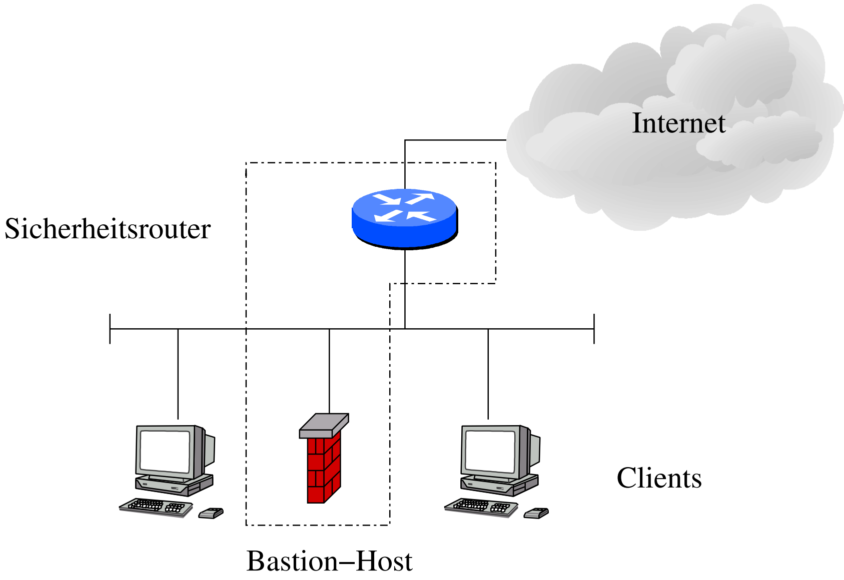

Abbildung 73: Bastion Host Architektur mit überwachtem Host

Die Paketfilterung auf dem Sicherheitsrouter muß so konfiguriert werden, daß der Bastion-Host das einzige System im internen Netz darstellt, zu dem Rechner aus dem Internet Verbindungen aufbauen können[51]. Zusätzlich sind nur gewisse Dienste zugelassen. Alle externen Systeme, die auf interne Systeme zugreifen wollen, und auch alle internen Systeme, die externe Dienste wahrnehmen wollen, müssen sich mit diesem Rechner verbinden. Daraus ergibt sich ein besonderes Schutzbedürfnis für diesen Bastion-Host. Der Vorteil bei dieser Konstruktion ist die Tatsache, daß ein Router leichter zu verteidigen ist. Dies liegt u.a. daran, daß auf ihm keine Dienste angeboten werden. Nachteilig wirkt sich aus, daß bei einer eventuellen Erstürmung des Bastion-Host das interne Netz vollkommen schutzlos ist.

9.2.3 Architektur mit überwachtem Teilnetz und Single-Homed Gateway

Die **screened subnet architecture** erweitert die Architektur mit überwachtem Host um eine Art Pufferzone, die als Grenznetz das interne Netz vom Internet isoliert. Diese Isolierzone wird auch **Demilitarisierte Zone** (**DMZ**) genannt.

Bastion-Hosts sind von ihrer Art her die gefährdetsten Rechner in einer Firewallkonstruktion. Auch wenn sie in der Regel mit allen Mitteln geschützt sind, werden sie doch am häufigsten angegriffen. Die Ursache liegt darin, daß ein Bastion-Host als einziges System Kontakt zur Außenwelt unterhält.
Die Eigenschaften eines überwachten Teilnetzes sind:

- Das Teilnetz dient ausschließlich zum Transport von Daten ins Internet oder aus dem

[51]das einzige nach außen sichtbare System

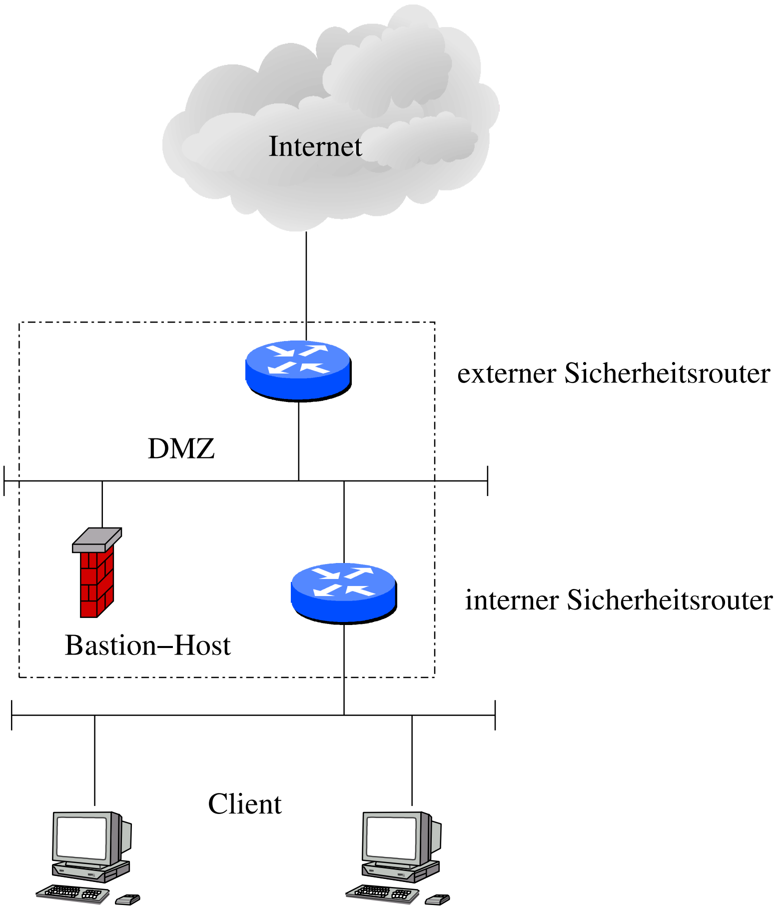

Abbildung 74: Bastion Host Architektur mit überwachtem Teilnetz

Internet.

- Das Teilnetz wird an beiden Seiten durch Paketfilter geschützt, so daß alle Rechner, die an das Teilnetz angeschlossen sind, in einer geschützen Position stehen.
- Um das zu schützende Netz zu erreichen, müssen die Daten aus dem Internet zuerst das Teilnetz erreichen, zum Bastion geleitet werden und dort dann an einen Client im internen Netz gegeben werden.
- An das überwachte Teilnetz können neben dem Bastion-Host weitere Rechner angeschlossen werden, die Dienste zur Verfügung stellen, die aus dem Internet erreicht werden dürfen (z.B. Mail, WWW, FTP, etc.).

Ein überwachtes Teilnetz mit **Single-Homed Application Level Gateway** ist in Abbildung 74 dargestellt. Der äußere Paketfilter stellt die Verbindung zum Internet her und hat die Aufgabe, das Teilnetz und das dahinterliegende, zu schützende Netz

vor Angriffen aus dem Internet zu sichern. Die Regeln müssen so eingestellt sein, daß nur Pakete hindurchgelassen werden, die für den Application Level Gateway bestimmt sind. Durch die Proxys auf dem Application Level Gateway erfolgt die Überprüfung des gesprochenen Protokolls auf Anwendungsebene.
Eine weitere Aufgabe des äußeren Paketfilters ist die Abwehr von IP-Spoofing (vgl. Seite 89), indem an dem äußeren Netzinterface alle einkommenden Pakete verworfen werden, die als Absenderadresse eine IP-Adresse aus dem zu schützenden Bereich haben.
Der zweite Paketfilter ist intern mit dem zu schützenden Netz verbunden. Er sichert das interne Netz vor Angriffen aus dem Internet und dem überwachten Teilnetz. Außerdem sorgt er dafür, das alle proxyfähigen Dienste vom internen Netz aus nur über den Application Level Gateway abgewickelt werden können.
Somit müssen Angreifer aus dem Internet zuerst den externen Paketfilter, evtl. den Application Level Gateway und natürlich auch noch den internen Paketfilter überwinden, um in das interne Netz eindringen zu können.
Ein Nachteil dieser Architktur besteht darin, daß ein Angreifer die Server angreifen kann, die ihre Dienste für das Internet in der DMZ zur Verfügung stellen. Von diesen Rechnern aus können dann weitere Angriffe gegen das interne Netz gefahren werden. Ebenso ist zu bedenken, wenn der äußere Paketfilter fehlerhaft im Verhalten ist, so ist der direkte Zugriff auf den interne Filter möglich. Deshalb sollten die beiden Filter idealerweise unterschiedliche Produkte von verschiedenen Herstellern sein.

9.2.4 Architektur mit Screened Subnet und Multi-Homed Gateway

Diese Firewallarchitektur ist analog zur Single-Homed Screened Subnet. Der wesentliche Unterschied besteht nur darin, daß der Application Level Gateway zwei oder mehr Interfaces mit angeschlossenen Teilnetzen hat. Bei nur zwei Interfaces wird der Gateway als Dualhomed bezeichnet. Somit entsteht zusammen mit dem externen Paketfilter das äußere Screened Subnet, in dem allgemeine Dienste zur Verfügung gestellt werden, die eine sehr niedrige Vertrauensstellung aufweisen. .
Der Application Level Gateway zusammen mit dem internen Packetfilter bildet das interne Screened Subnet, das als sehr vertrauenswürdig einzustufen ist.
An den weiteren Interfaces des Application Level Gateways werden Dienste zur Verfügung gestellt, die weniger vertrauenswürdig sind. Hier wird durch den Administrator das Sicherheitsniveau vorgegeben.
Die Abbildung 75 stellt diese Art der Architektur dar.
Bei dieser Konstellation müssen nun bei einem Angriff drei verschiedene Sicherheitsprodukte überwunden werden.

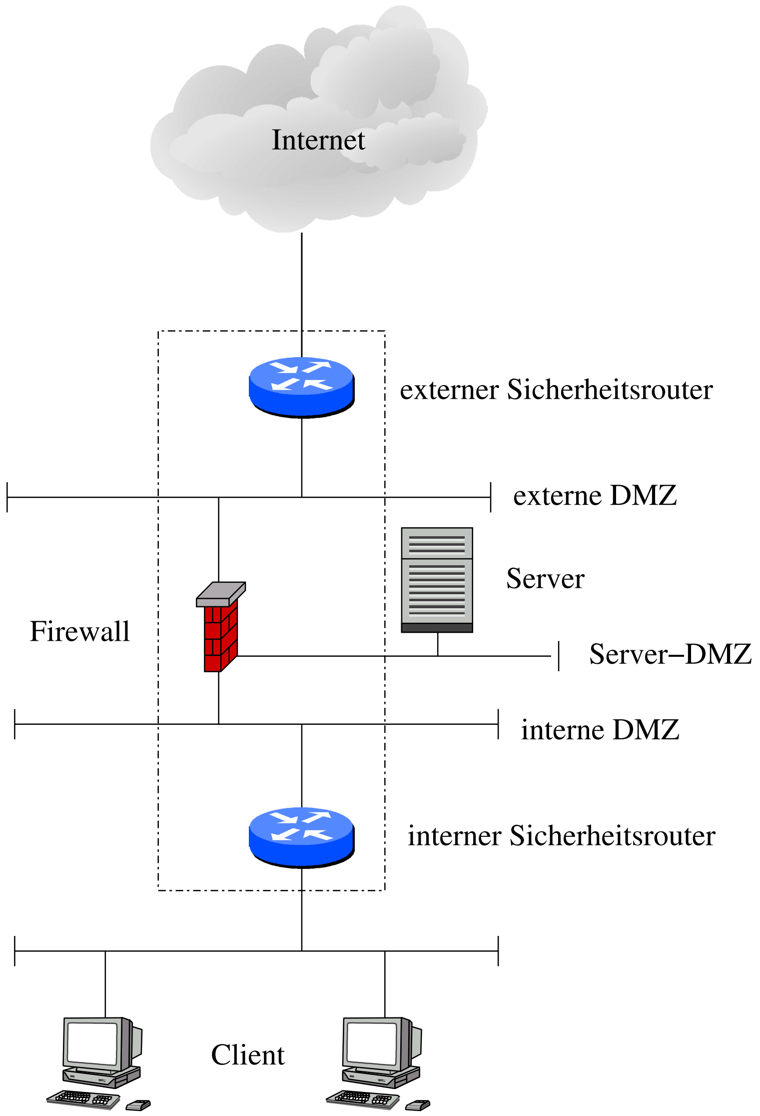

Abbildung 75: Multihomed Bastion Host Architektur mit überwachtem Teilnetz

- Jede Verbindung vom und zum Internet wird durch den Application Level Gateway abgesichert, so daß alle Vorteile der Überprüfung auf Anwendungsschicht genutzt werden.
- Durch den symmetrischen Aufbau ist im Prinzip die gleiche Sicherheit gegen interne wie externe Angriffe gegeben. Das ist ein deutlicher Pluspunkt, wenn diese Umgebung durch Dritte betrieben wird.
- Werden zwei baugleiche Paketfilter verwendet, so muß zumindest der andersartige Application Level Gateway noch überwunden werden.

- Es gibt keine Wege an den Paketfiltern und dem Application Level Gateway vorbei ins Internet.

Diese Lösung ist weniger flexibel, da alle Verbindungen durch den Application Level Gateway kontrolliert werden. Protokolle, die nicht über Proxies abgebildet werden können, können hier nicht einfach so durchgeschalten werden. Die Handhabung dieser Problematik ist von Firewallsoftware zu Firewallsoftware unterschiedlich.

9.3 Management

Werden in einem Unternehmen mehrere Firewalls betrieben, so ergeben sich zusätzliche Komplikationen beim Management der Systeme. Mehrere Firewalls sind notwendig bei räumlicher Trennung mehrerer Niederlassungen mit eigenen Internetzugängen oder bei einer geschäftsprozeßbedingten Kopplung zu den Netzen verschiedener Partnerfirmen (**Extranet**). Die Konfiguration eines Firewallsystems ist immer einmalig auf den jeweiligen Standort und die Netzwerktopologie zugeschnitten. Es gibt keine Firewallkonfiguration von der Stange.
In den wenigsten Fällen ist es möglich, einen Firewall direkt an der Konsole des Rechners zu managen. Selten wird der Firewallrechner am Schreibtisch des Administrators stehen. Manche Firewalls erlauben ein Remote-Management nur über dedizierte Leitungen (seriell, eigenes LAN). Die Kenntnis der internen Konfiguration oder das Mithören der Passwörter beim Remote Login über das interne LAN erleichtern einen Einbruchsversuch. Daher muß beim direkten Login in den Firewall-Rechner immer eine verschlüsselte Verbindungen benutzt werden. Liefert der Hersteller keine ausreichenden Tools zum Remote-Management von Firewallsystemen, so kann man durch den Einsatz von frei verfügbaren Tools die Situation wesentlich verbessern. Programme wie Secure Shell (ssh, für remote login), PGP (zum Signieren von Konfigurationsdateien) und rsync (zum Datentransfer und Datei-Synchronisation über ssh) bilden schon eine sehr gute Basis. Hat man viele Systeme zu betreuen, mag die Installation eines eigenen Firewall-Management-Rechners vorteilhaft sein. Dieser zentrale Management-Rechner kann dann die Meldungen aller Firewalls auswerten und archivieren, und kann seinerseits einen abgesicherten WWW-Server bereitstellen (Apache-SSL), über den Management-Funktionen zentral für mehrere Firewallsysteme und Statusabfragen abgewickelt werden können.
Firewallsysteme sind für einen vollautomatischen Betrieb ausgelegt. Allerdings mag dies täuschen, da je nach Firewall-Typ unter Umständen Statistiken, Logdaten, Alarme usw. generiert werden. Diese Botschaften benötigen einen Adressaten, einen kompetenten und verantwortlichen Menschen, der die Nachrichten des Firewalls bewertet und entsprechend handelt. Zur Definition dieser Aktionen ist zusammen mit, oder besser noch vor der Installation des Firewalls festzulegen, was der Firewall wie schützen soll. Es ist festzulegen, was in bestimmten Situationen zu tun ist und wer dies tut. Der Betrieb eines Firewallsystems kann mehr Personalresourcen fordern als der Betrieb eines LAN-Servers.
Die sichere und einfache Verwaltung der Firewallkomponenten ist ein wichtiger Bestand-

teil eines guten Firewalls. Die hierbei eingesetzten Programme oder Rechner werden als **Sicherheitsmanagementkomponenten** eines Firewalls bezeichnet und sollten die folgenden Aufgaben erfüllen:

- Zugangskontrolle zur Konfigurationssoftware und den dazugehörigen Dateien unter Umständen mit Rollentrennung für Administrator und Revisor.
- Eingabe und Kontrolle von Filterregeln oder zur Umsetzung der Sicherheitspolitk in anderer Form, also z.B. durch benutzerspezifische Daten bei reinen Application Level Gateways.
- Eingabe und Kontrolle von Daten, die für den Betrieb des Firewalls notwendig sind, z.B. DNS-Informationen, Aliasnamen für Mails, Routingeinträge u.a.
- Einstellung und Überprüfung der Protokolldaten unter Umständen mit automatischen Hilfsmitteln (Intrusion Detection System).
- Verwaltung der Schlüssel und anderer relevanter Daten, die z.B. beim Einsatz von virtuellen, privaten Netzen benötigt werden.
- Erstellung und Wiedereinspielen von Backups.

Weitere Informationen zur Sicherheitspolitik wurden bereits in Abschnitt 2.5 erläutert.
Es existieren sehr viele verschiedene Ansätze, wie die doch recht komplizierte Umsetzung der Sicherheitspolitik, beziehungsweise die Eingabe der Filterregeln bei einem Firewall mit mehreren Interfaces und mehreren zu schützenden Netzen mit unterschiedlichen Anforderungen realisiert werden kann. Neben zumeist älteren Produkten, deren Konfigurationsdateien nur mit Hilfe eines Editors erstellt und verändert werden können, haben sich heute vor allem graphische Benutzeroberflächen durchgesetzt.
Viele dieser Programme bieten aber nur eine einfache Eingabemöglichkeit für die erstellten Regeln, die anschließend automatisch in eine Form übersetzt werden, die die Filterprogramme verstehen und in die Konfigurationsdateien geschrieben werden. Dieses Vorgehen hat den großen Vorteil, daß eine Kontrolle dieser Konfigurationsfiles durch den Administrator möglich ist. Man muß aber auch bedenken, daß es durch die automatische Umsetzung zu Regelinkonsitenzen kommen kann, evtl. durch einen Fehler im Programm.
Moderne Firewalls bieten durch die graphische Oberflächen eine Möglichkeit an, die definierte Sicherheitspolicy in einer leicht verständlichen Form darzustellen.
Neben einer leichten und übersichtlichen Administrierbarkeit sollte aber auch der Aspekt der Sicherheit nicht vergessen werden. Ein Firewall, dessen Konfiguration auch von unbefugten Personen modifiziert werden kann, ist gefährlicher als gar kein Firewall. Es kommt daher wesentlich darauf an, daß insbesondere bei mehreren zu verwaltenden Firewallkomponenten eine gegen Mitlesen und Veränderungen geschützte Übertragung der Daten gewährleistet ist. Hierfür kommen mehrere Möglichkeiten in Frage:

- Kein administrativer Zugang zu den Firewallkomponenten außer über eine Konsole und ein Diskettenlaufwerk. Die Erzeugung der Konfigurationdateien findet auf einem isolierten Rechner (**Security Management Server**, **SMS**) statt, der unter Umständen auch mit einer Zusatzsoftware zur einfacheren Erstellung der Daten versehen ist. Die Konfigurationsdateien werden dann per Diskette auf den eigentlichen Firewall-Rechner übertragen und dort von der Konsole aus installiert. Alle anderen Zugänge zu den Filterkomponenten sind abgeschalten. Dieses Vorgehen hat den Vorteil, daß nur Personen, die den Zugang zu den Aufstellungsräumen der Geräte haben, auch in der Lage sind, die Konfiguration zu ändern. Nachteil dieser Methode ist es, daß Änderungen nicht sofort durchgeführt werden können und auch die Verwaltung von vielen Komponenten relativ umständlich ist.

- Verwaltung des Firewalls durch Administrationssoftware, die sich auf den Komponenten befindet. Viele Filterkomponenten werden mit einer sehr komfortablen graphischen Bedienoberfläche ausgeliefert, die direkt zur Verwaltung eines oder mehrerer Geräte desselben Typs eingesetzt werden kann und auch dort installiert wird. Ein spezieller Administrationsrechner ist nicht nötig. Vorteil dieser Lösung ist, daß die komplette Funktionalität des Firewalls in einem Gerät gebündelt ist und auch die Wirkung von Änderungen direkt geprüft werden kann. Aber auch hier ist ein Zugang zu den Aufstellungsräumen des Firewalls nötig. Außerdem haben die Administratorsoftware und die dafür notwendigen Bibliotheken oftmals einen größeren Umfang als die eigentliche Filtersoftware, so daß es sich nicht mehr um ein Minimalsystem handelt. Dementsprechend muß mit zusätzlichen Fehlern gerechnet werden, die die Sicherheit des Firewalls verringern.

- Die gängigste Lösung zur Administration von Firewallkomponenten ist die über ein abgetrenntes Netz. Auch bei dieser Alternative werden die eigentlichen Konfigurationen auf einem speziellen Administrationsrechner ausgeführt, die Dateien aber anschließend über ein, nur für diese Zwecke vorgesehenes Netz an die Filterkomponenten übertragen. Bei diesem Netz kann es sich sowohl um ein physikalisch getrenntes Netz handeln, an das alle Filterkomponenten über dedizierte Netzwerkinterfaces angeschlossen sind, als auch um ein logisches Netz, welches mit Hilfe von verschlüsselten Verbindungen aufgebaut wird.

9.4 Praktische Aufgaben

9.4.1 Plug Gateways FWTK

1. Aktivieren Sie ein Plug-Gateway auf Port 9999.
2. Sorgen Sie dafür, daß Verbindungen von der `127.0.0.1` über diesem Plug auf den `secserver`, `192.168.216.254`, Port 23 weitergegeben werden (Test mit `telnet localhost 9999` und Logfiles). Verbindungen von Ihrem `eth0` Interface aus sollen zum `test4all`, `192.168.216.253`, Port 23 gehen.

9.4.2 SOCKS

1. Installieren Sie den Dante-Server und Client über `YaST2` auf Ihrer Maschine und lesen Sie die Manpages für die Konfigurationsfiles (`man sockd.conf` und `man socks.conf`).
2. Sie arbeiten immer mit Ihrem Partnerrechner zusammen, d.h. Sie installieren und konfigurieren den Dante-Server auf Ihrem Rechner und den Dante-Client auf Ihrem Partnerrechner, um darüber zu testen. Das schließt ein, daß Sie dafür Sorge zu tragen haben, daß die 'Besatzung' Ihres Partnerrechners die Möglichkeit hat, das Clientkonfigurationsfile `/etc/socks.conf` zu editieren.
3. Erstellen Sie als Grundkonfiguration des Dante-Servers Folgendes:
   - Die Logfileeinträge sollen nach `/var/log/messages` geschrieben werden.
   - Das Logging jeder Regel soll maximal sein.
   - Legen Sie die interne und externe IP-Adresse Ihres Dante-Servers fest.
   - Legen Sie fest, daß für die Clientrules keine Authentisierung nötig ist.
   - Editieren Sie Clientregeln so, daß grundsätzlich Ihr Partnerrechner erlaubt ist, Sie als SOCKS-Server zu verwenden.
   - Blocken Sie alle anderen Verbindungsaufbauversuche.
   - Erlauben Sie folgendes bei den Serverregeln:
     - Niemand soll Ihr Loopback Interface erreichen dürfen.
     - Telnet und SSH soll von Ihrem Partnerrechner aus ins Netz `192.168.216.0/24` erlaubt sein, andere Destinations werden geblockt.
     - Telnet und SSH soll von Ihrem Partnerrechner aus in die Domain `.secp.nm.informatik.uni-muenchen.de` erlaubt sein, andere Domains sollen geblockt werden.
     - Alle anderen Verbindungen werden unterbunden.
4. Konfigurieren Sie den Dante-Client Ihres Partnerrechners so, daß

- DNS-Auflösung über UDP geht.
- der auf Ihrem Partnerrechner eingetragene Nameserver in der `/etc/resolv.conf` direkt und nicht über den SOCKS-Server angesprochen wird.
- Anfragen an das Loopback Interface Ihres Clients direkt gehen.
- Kommunikation zwischen Ihrem Rechner und Ihrem Partnerrechner direkt geht.
- alle andere Kommunikation (auf IP-Basis oder über Namen) an den Dante-Server geschickt wird.

5. Prüfen Sie die Client- und Serverkonfiguration, indem Sie versuchen, Verbindungen über Telnet, SSH und z.B. HTTP zu den unterschiedlichsten Zielen aufzubauen. Dies geschieht z.B. bei Telnet mit `socksify telnet 192.168.216.254`. Treten Probleme auf, überprüfen Sie das Logfile Ihres Dante-Servers. Es kann auch der Debug-Level des Clients erhöht und die Logausgaben nach STDOUT gelenkt werden.

Für alle, die sich noch etwas mit dem Dante-Server spielen wollen, hier noch ein paar Zusatzaufgaben. Die Bearbeitung ist auf freiwilliger Basis:

1. Wie ist die Konfiguration Ihres Servers und des Clients abzuändern, wenn der Client keine DNS-Auflösung besitzt und der Server diese übernimmt?
2. Sie haben mehrere User auf dem Client und wollen aber nur einem bestimmten User den Zugriff auf eine spezielle Einstellung Ihres Servers erlauben (Authentisierungsmethode `username`)?

10 Intrusion Detection und Intrusion Response

Hinter den Begriffen **Intrusion Detection** (**ID**) [Amor 99] und **Intrusion Response** (**IR**) [RRW 01] verbergen sich zwei Ansätze, bei denen mit mehr oder weniger intelligenten Programmen Angriffe automatisch erkannt und abgewehrt werden sollen. Ziel eines ID-Systems muß sein, einen durchschnittlichen Administrator soweit zu unterstützen, daß er ohne tiefgreifende Kenntnisse im Bereich Internetsicherheit in der Lage ist, aus einer großen Menge von Protokolldaten einen Angriff zu erkennen. IR-Systeme sollen dagegen dazu dienen, automatische Gegenmaßnahmen einzuleiten, sobald ein Angriff erkannt wurde.
Im Idealfall verfügen diese Programme über ebensoviel Kenntnisse wie ein guter Administrator und sind daher in der Lage, in beliebigen Protokolldaten nicht nur einen Angriff zu erkennen, sondern auch noch Aussagen über die Stärke der Bedrohung und die notwendigen Gegenmaßnahmen zu machen. Zur Zeit ist das allerdings noch ein Gebiet, das intensiv erforscht wird, so daß wesentliche Verbesserungen an den vorhandenen Programmen jederzeit möglich sind.
Genauere Ausführungen zu den folgenden Abschnitten finden Sie unter [Nort 99] und [EDZ 00].

10.1 Zeitliche Abfolge

Betrachtet man Intrusion Detection, so muß man sich Gedanken machen, zu welchen Zeitpunkten eine Auswertung der gesammelten Daten erfolgt. Dazu gibt es zwei Möglichkeiten.

10.1.1 Batch oder Intervall orientierte Auswertung

Bei der Batch orientierten, auch Intervall orientiert genannten, Variante schreiben Auditing Mechanismen des Betriebssystems oder andere Host basierende Agenten Informationen zu erkannten Ereignissen in Dateien, die dann von einen ID System periodisch nach Anzeichen von Eindringen oder Mißbrauch durchsucht werden. Dieser Vorgang ist somit nicht ständig in Aktion, sondern wird zu fest definierten Zeiten gestartet.

Vorteile:

- Batch orientierte Analyseverfahren erzeugen wenig Prozessorlast, besonders wenn die Intervalle kurz sind und deshalb das Datenvolumen gering bleibt.
- Sie eignen sich gut in Umgebungen, in denen der Bedrohungsgrad niedrig, das Verlustpotential durch Einzelangriffe aber sehr hoch ist. In diesen Umgebungen wird oft großer Wert darauf gelegt, verdächtige Vorfälle zu dokumentieren, um genügend Beweismateriel für Anzeigen zu sammeln. Der Fokus liegt daher weniger auf der sofortigen Behandlung und Abwehr von Vorfällen.