Home of KR

Computer
- Siesta-GUI
- Logsplitter
- NAT-Tutorial
- HTML-Tutorial
- PHP-Tutorial
- Gästebuch-Spam bekämpfen
- Linux Tipps

Elektronik
- Lenkrad für PC
- Schaltungen
- USB
- Mikrocontroller
- LED spots

Uni
- Matlab-Einführung
- Java-Einführung
- Bakk.-Arbeiten
- Brunel-Blog

Fun
- Seltsame Suchbegriffe
- Witziger Gästebuch-Spam

Links
- Fun
- Allgemeine

Impressum
Kontakt

NAT - Network Address Translation

Einleitung

Network Address Translation beschreibt ganz allgemein *"ein Verfahren, um eine IP-Adresse in einem Datenpaket durch eine andere zu ersetzen. Häufig wird dies benutzt, um private IP-Adressen auf öffentliche IP-Adressen abzubilden."*(aus http://de.wikipedia.org/wiki/Network_Address_Translation)

In diesem Tutorial möchte ich erläutern, was die Network Address Translation (NAT) ist, was man mit ihr anfangen kann und wie man sie (unter Linux bzw. Unix-Derivaten) konfiguriert. Diese Einführung erhebt aber keinen Anspruch auf Vollständigkeit, vielmehr soll sie eine Idee bzw. ein Gefühl vermitteln, was in unseren modernen Computernetzen möglich und sinnvoll ist und was nicht.

Zuerst soll der Aufbau eines IP-Pakets eingegangen werden. Nach einer kurzen Übersicht über die Möglichkeiten im Linux-Kernel möchte ich auf das primären Anwendungsgebiet eingehen, nämlich die Anbindung eines privaten Netzes über einen Router an das Internet. Schließlich möchte ich noch auf weitere Möglichkeiten eingehen, wie man zum Beispiel NAT einsetzen kann, um alle Ports im Internet zu erreichen, auch wenn man selbst hinter einer restriktiven Firewall sitzt. Auch wenn ich zu den Beispielszenarios Lösungen präsentiere, heißt das nicht automatisch, dass diese die einfachsten/besten/genialsten sind.

Pakete im Netzwerk

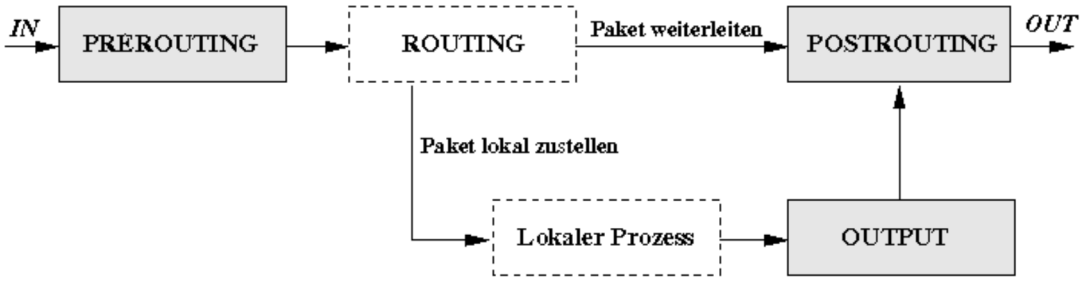
Bevor wir Pakete durch das Netz schicken bzw. umleiten, sei an dieser Stelle noch kurz erwähnt, welche Attribute eines Pakets für uns von Interesse sind. Dazu beschränke ich mich auf IP-Pakete und die zugehörigen Protokolle des Transport Layers (Transportschicht): UDP und TCP.

Der genaue Aufbau eines IP-Pakets ist bereits an anderen Stellen im Netz gut beschrieben, ich verweise daher an diese, wie zum Beispiel IP-Paket bei Wikipedia. Was für uns wichtig sein wird, sind die Felder **Source Address** und **Destination Address**. Dort stehen - nomen est omen - die IP-Adressen der Empfänger und Absender des Pakets drinnen.

Damit der Empfänger nun weiters weiß, für welche Anwendung das Paket zu verwenden ist, gibt es die Transportschicht, im Speziellen TCP und UDP. Wiederum verweise ich für weitere Details auf vorhandene Quellen, zum Beispiel TCP bei Elektronikkompendium und UDP-Paket bei Elektronikkompendium. Für unsere Zwecke ist es ausreichend zu wissen, dass die Transportschicht den Prozessen Ports zuweist. So hat z.B. der http-Server standardmäßig Port 80, SSH verwendet Port 22, usw. Die Kombination aus Port und IP-Adresse wird Socket genannt und ist eindeutig. Verbindungen laufen also von einem Socket zu einem anderen Socket, zum Beispiel vom Socket mit der IP 123.123.123.123, Port 65432 zum Socket mit der IP 112.112.112.112, Port 80, wie es zum Beispiel bei einem Browser, der auf 123.123.123.123 läuft und einem http-Server auf 112.112.112.112 der Fall sein könnte. Serverprozesse haben meist fest zugewiesene Ports ("well known ports", siehe z.B. Well known ports bei IANA oder die wichtigsten zusammengefasst unter Port bei Wikipedia), während ein Client bei Bedarf einen (mehr oder weniger beliebigen) Port aus dem oberen Adressbereich verwendet. Der vermutlich am meisten benutze Server-Port ist der HTTP-Port 80, der auch beim Laden dieser Seite verwendet wurde.

Linux und Netfilter

Der Linux-Kernel hat standardmäßig ein Paketfilter-Framework mit Namen **netfilter** (Siehe netfilter.org) eingebaut. Damit kann man aus jeder Linux-Maschine mit entsprechender Anzahl an Netzwerkkarten bzw. -interfaces einen vollwertigen Router basteln. Wir werden das Kommandozeilenprogramm "iptables" verwenden, mit dem sich komplexe Regeln für das Filtern und Modifizieren von Paketen festlegen lassen. Die für uns relevanten Regeln werden (wenig überraschend) in der "nat"-Tabelle eingetragen. Diese Tabelle kennt drei vordefinierte Ketten: **PREROUTING**, **OUTPUT** und **POSTROUTING**.



Die Ketten **PREROUTING** und **POSTROUTING** sind dabei die beiden wichtigsten. Wie die Namen schon sagen, ist die PREROUTING-Kette für Pakete zuständig, die gerade frisch am Netzwerkinterface angekommen sind. Dabei ist noch keine Routing-Entscheidung gefallen, es ist also noch nicht entschieden, ob das Paket am lokalen Rechner interpretiert wird oder ob es nur an einen anderen Rechner weitergeleitet wird. Nach der PREROUTING-Kette findet der eigentliche Routing-Prozess statt. Ist der lokale Rechner der Empfänger, so wird das Paket an die entsprechende Anwendung weitergereicht und wir NAT ist nicht weiter notwendig. Stellt sich jedoch heraus, dass der Empfänger ein anderer ist, so wird das Paket wieder auf die Reise geschickt ("forwarding", vorausgesetzt natürlich, dass diese Weiterleitung überhaupt aktiviert ist). Bevor das Paket aber auf die Reise geschickt wird, kommt es vorher noch in die POSTROUTING-Kette. Sobald es diese passiert hat, verlässt das Paket das Netzwerkinterface. Sofern das Paket lokal erstellt wurde, ist der Weg ein etwas anderer: Das Paket kommt durch die OUTPUT-Kette und passiert dann die POSTROUTING-Kette.

Bevor wir Pakete manipulieren, müssen wir eben dieses Feature im Kernel überhaupt aktivieren. Um dort jegliche Routing-Funktionalität freizuschalten, sollten die Kommandos (ohne führendes "$>", Zeilen beginnend mit "#" sind Kommentare)

```
# WICHTIG: IP-Forwarding im Kernel aktivieren.
# Default ist "disabled"!
$> echo "1" > /proc/sys/net/ipv4/ip_forward

# Laden diverser Module. Oft sind die Module bereits geladen
# (gerade bei neueren Kernels oft der Fall), dann sind die
# folgenden Kommandos nicht notwendig.

# iptables-Modul im Kernel laden:
$> modprobe ip_tables

# Connection-Tracking aktivieren
# (Status der Verbindungen wird beruecksichtigt)
$> modprobe ip_conntrack

# Zusaetzliche Funktionen fuer IRC:
$> modprobe ip_conntrack_irc

# Zusaetzliche Infos fuer FTP:
$> modprobe ip_conntrack_ftp
```

ausreichen. Bei Fehlermeldungen nach obigen Befehlen sind vermutlich die Routing-Funktionen nicht in den Kernel hineinkompiliert, in diesem Fall sei wiederum an weitere Quellen im Netz verwiesen (z.B. http://www.linuxfocus.org/Deutsch/May2000/article151.shtml).

Beispiel: Ein privates Netz über NAT an das Internet anbinden

Wir wissen nun einerseits, wie IP-Pakete aussehen, und andererseits, wie man unter Linux (bzw. Unix-Derivaten) Pakete verändern kann. Wir können daher den Schritt in die Anwendung wagen. Die populärste Frage betreffend NAT ist vermutlich jene nach dem Freigeben einer Internetverbindung für Rechner in einem privaten Netz, weshalb ich auch gleich damit beginnen möchte.

Ein Analogon: Mehrere Untermieter ohne eigene Postadressen

Zuvor aber ein weitaus plakativeres und hoffentlich wesentlich einfacher verständlicheres Analogon: Nehmen wir an, wir hätten einen Hausbesitzer und ein paar Untermieterfamilien. Nehmen wir an, der Postbote wüsste von den Untermietern nichts und würde jeden Brief, der nicht an den Hausbesitzer adressiert ist, als unzustellbar zurückschicken. Wir nehmen zusätzlich an, dass er mehrere Postfächer hat, die auch auf der Adresse aufscheinen. Für die Untermieter besteht nur die Möglichkeit, ihre Briefe in einen Briefkasten vor dem Büro des Hausmeisters zu werfen, der diese dann zur Post bringt. Wie können dennoch alle Untermieter am Postverkehr teilnehmen?

Eine Lösung für das Untermieter-Problem wäre, dass der Hauseigentümer die Briefe nimmt,

jedem versendenden Untermieter einen Postkasten zuweist und dann seine (=die des Hauseigentümers) Adresse inkl. Postfach des zugehörigen Untermieters als Absender auf den Brief schreibt bevor er ihn auf das Postamt bringt. A bekommt den Brief und schickt die Antwort an den Hauseigentümer (plus Postfach) zurück. Der Brief landet dann im Postfach des Untermieters, der den Brief dann entgegen nehmen wird. Schließlich ersetzt der Hausmeister noch seine Adresse auf dem Antwort-Brief durch die des jeweiligen Untermieters. Damit wäre soweit alles perfekt, für den Untermieter wäre diese Lösung vollkommen transparent! :-)

Vom Untermieter-Problem zur Computerwelt

NAT funktioniert für das lokale Netz im Prinzip genauso wie das Untermieter-Beispiel. Jede Untermieterfamilie entspricht einer IP im lokalen Netz, jedes Familienmitglied entspricht einer Port-Nummer zur Untermieter-Familien-IP, der Hauseigentümer wäre der Router und der Adressat A ein Server im Internet. Ebenso entspricht ein Socket einer Kombination aus Adresse und Postfach bzw. Untermieterfamilie und Untermieterfamilienmitglied. Als Aufgaben im Untermieter-Beispiel haben wir die folgenden:

- Die Untermieter haben ihre Briefe in den Postkasten beim Hausbesitzer zu werfen
- Der Hausbesitzer ersetzt die Versender-Adressen durch seine eigene inkl. Postfach
- Bei den Antworten ersetzt der Hausbesitzer seine Adresse durch die Adresse des entsprechenden Untermieters.

Ganz analog ist die Situation im lokalen Netz:

- Die Rechner im lokalen Netz ("Clients") senden ihre Pakete mit dem gewünschten Empfänger-Socket an den Router (durch Setzen des Routers als Standard-Gateway wird das über Ethernet ermöglicht)
- Der Router ersetzt in jedem Paket das Absender-Socket des Clients durch ein eigenes, freies Socket.
- Bei Antworten auf eines der Router-Sockets ersetzt der Router den Empfänger durch den entsprechenden Client und schickt das Paket ins lokale Netz.

Dass die Clients alle den Standard-Gateway korrekt eingetragen haben, setzen wir einmal voraus. Damit bleibt noch der Router zu konfigurieren. Netfilter ist glücklicherweise so ausgelegt, dass es zu jeder Regel automatisch auch die dazu inverse Regel gibt, wir brauchen also nur eine Regel explizit vorgeben. Welche der beiden zueinander inversen Regeln nun tatsächlich dem Kernel mitgeteilt wird, ergibt sich meist von selbst aus dem Grad der Unbestimmtheit der Regel. Hier zum Beispiel ist die Regel "Ersetze bei allen Paketen aus dem lokalen Netz den Absender" einfacher als "wenn ein Client etwas an einen Server geschickt hat, dann ersetze bei der Antwort des Servers die Empfänger durch blablabla". Im Allgemeinen kann man sich auch an die Daumenregel halten, dass jene Regel, die im Laufe einer Verbindung zuerst angewandt wird, auch jene ist, die man dem Kernel mitteilt.

Dem Kernel das Weiterleiten beibringen

Wir möchten also dem Kernel das folgende mitteilen: Bei Paketen aus dem lokalen Netz, dessen IP-Adresse nicht mit seiner übereinstimmt, soll er die Absender-Adresse auf sich selbst ändern. Dazu machen wir noch die Annahme, dass das erste Netzwerk-Interface des Linux-Routers "eth0" mit dem lokalen Netz verbunden ist und die Internet-Verbindung am zweiten Interface "eth1" verfügbar ist. Der Befehl zur Freigabe der Internet-Verbindung lautet dann:

```
# Anbinden eines LAN an das Internet
$> iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
```

Der Befehl erklärt sich wie folgt:

iptables:	das Kommandozeilenprogramm, mit dem wir den Kernel konfigurieren
-t nat	Wähle die Tabelle "nat", um NAT zu betreiben.
-A POSTROUTING	Füge eine Regel in der POSTROUTING-Kette ein (-A steht für "append").
-o eth1	Wir wollen Pakete, die den Router an der zweiten Netzwerkschnittstelle "eth1" verlassen (-o für "output")...
-j MASQUERADE	... maskieren, der Router soll also seine eigene Adresse als Quelladresse setzen.

Noch ein paar Anmerkungen zu obigem Befehl: Pakete, die vom Router selbst ins Internet geschickt werden, werden ebenfalls maskiert, da diese - siehe Grafik weiter oben - nach der OUTPUT-Kette ebenfalls in die POSTROUTING-Kette kommen. Da der Kernel aber nach Möglichkeit die Quellports unverändert lässt und Prozesse am Router ohnehin nur freie Ports binden können, werden lokale Pakete in der Regel nicht verändert, obwohl ja "eigentlich" die Regel darauf angewandt wird. Weiters ist es noch wichtig, dass das Output-Interface ein ganz beliebiges sein kann, also auch ISDN-oder DSL-Interfaces (oft "ppp0" oder "ippp0"). Eine

Übersicht über die verfügbaren Interfaces gibt

```
# Anzeige der Netzwerkinterfaces
$> ifconfig
```

Nachteil der Anbingung mittels NAT

Die Rechner im lokalen Netz haben zwar Zugriff aus das Internet, jedoch ist dieser mit gewissen Einschränkungen verbunden. Will zum Beispiel ein Rechner aus dem Internet mit einem lokalen Rechner eine Verbindung aufbauen, so kann dieser nur den Router (genauer: Einen Port am Router) adressieren und auf das Beste hoffen. Im Normalfall wird der Port gerade nicht in Verwendung sein und das Paket wird verworfen. Und selbst wenn der Port zufällig gerade vom beabsichtigten lokalen Rechner verwendet wird, so hat dieser ja über jenen Port eine Verbindung zu einem anderen Rechner laufen und wird daher den unerwarteten Verbinungsaufbau ablehnen. Für Verbindungsaufbauten von außerhalb des lokalen Netzes sieht es also nicht gut. Für regelmäßige Services nach außen kann man Ports am Router auf bestimmte Ports im lokalen Netz abbilden, um beispielsweise einen HTTP-Server abgeschottet hinter dem Router laufen zu lassen oder um das eine oder andere Online-Spiel spielen zu können.

Ein genauerer Blick auf iptables

Nun, da das erste Beispiel hinter uns liegt, ist es an der Zeit, einen genaueren Blick auf die Möglichkeiten und Schalter von *iptables* zu werfen. Ein iptables-Aufruf sieht abstrakt gesehen so aus:

```
# Abstrakter Aufbau einer iptables-Anweisung
iptables [-t tabelle] Anweisung [Passmuster] [Aktion]
```

Für NAT ist, wie der Name schon sagt, die *nat*-Tabelle auszuwählen. Eine Anweisung kann auch weitere Optionen verlangen: Ein Passmuster für die Pakete und eine Aktion, die im Falle einer Übereinstimmung ausgeführt wird.

Tabelle auswählen

Alle unsere Befehle betreffend NAT werden mit

```
# Auswaehlen der NAT-Tabelle
# (weitere Argumente mit [...] abgekuerzt)
iptables -t nat [...]
```

begonnen. Damit wird die nat-Tabelle ausgewählt. Die andere beiden Tabellen wären *mangle* und *filter*, diese werden aber nicht für NAT verwendet und werden daher nur der Vollständigkeit halber erwähnt. Da die Default-Tabelle *filter* ist, muss für NAT die Tabelle immer ausgewählt werden.

Kommandos

Die wichtigsten Kommandos sind die folgenden (eventuelle weitere Passmuster sind mit [...] ausgeklammert):

```
# Im folgenden ist "Kette" stellvertretend
# fuer eine der Ketten PREROUTING, OUTPUT oder POSTROUTING

# Regel hinzufuegen:
$> iptables -t nat -A Kette [...]

# Regeln anzeigen lassen:
$> iptables -t nat -L

# Regel in Kette 'Kette' mit Nummer 'index' loeschen:
$> iptables -t nat -D Kette index

# Alle Regeln in 'Kette' loeschen:
$> iptables -t nat -F Kette
```

Für eine vollständige Auflistung der verfügbaren Kommandos ist die Manual-Page zu iptables zu empfehlen, abzurufen mittels

```
# Manual-page von iptables
$> man iptables
```

und mittels "q" wieder zu beenden.

Auswahl des Passmusters

Um gewisse Pakete zu manipulieren, möchten diese natürlich auch ausgewählt werden, weshalb es eine Vielzahl an Optionen gibt, diese zu spezifizieren. Ich werde hier ein paar ausgewählte Beispiele anführen, um die Verwendung zu verdeutlichen. Als Referenz aller verfügbaren Passmuster ("matches") sind wieder die Manual-Pages von *iptables* zu empfehlen.

```
# Aktionen, die auf ein passendes Paket zutreffen sollen,
# werden mit 'aktion' abgekuerzt.
# Je nach Passmuster wird eine passende Kette verwendet.

# TCP-Pakete von 192.168.1.2:
$> iptables -t nat -A POSTROUTING -p tcp -s 192.168.1.2 aktion

# UDP-Pakete an 192.168.1.2:
$> iptables -t nat -A POSTROUTING -p udp -d 192.168.1.2 aktion

# Alle Pakete von 192.168.x.x, die bei eth0 hereinkommen:
$> iptables -t nat -A PREROUTING -s 192.168.0.0/16 -i eth0 aktion

# Alle Pakete ausser TCP-Pakete von jedem ausser 192.168.1.2:
$> iptables -t nat -A PREROUTING -p ! tcp -s ! 192.168.1.2 aktion

# Pakete, die den Rechner ueber eth1 verlassen:
$> iptables -t nat -A POSTROUTING -o eth1 aktion

# TCP-Pakete von 192.168.1.2, Port 12345 bis 12356
# an 123.123.123.123, Port 22
# (Backslash bedeutet, dass es in der naechsten Zeile weitergeht)
$> iptables -t nat -A POSTROUTING -p tcp -s 192.168.1.2 \
   --sport 12345:12356 -d 123.123.123.123 --dport 22 aktion
```

Zu den Schaltern gibt es auch meistens eine Langform, wie beispielsweise *--source* zu *-s*, dadurch werden die Zeilen aber länger, während aber auch die Kürzel für gute Lesbarkeit sorgen.

Aktionen für ein Paket

Pakete können bereits ausgewählt werden, jetzt fehlen nur noch die jeweiligen Aktionen. In der NAT-Tabelle sind nur die vier Aktionen *SNAT*, *MASQUERADE*, *DNAT* und *REDIRECT* mit vorgestelltem "*-j*" sinnvoll. Auf die genauen Bedeutungen und Funktionen wird in den folgenden Abschnitten eingegangen.

```
# Im folgenden werden Optionen (Tabellenauswahl, Kommando, Passmuster),
# die nicht zur Aktion gehoeren, durch ein [...] abgekuerzt.

# Source-NAT: Absender zu 123.123.123.213 aendern
$> iptables [...] -j SNAT --to-source 123.123.123.123

# Maskieren: Absender auf ausgehendes Netzwerkinterface aendern
$> iptables [...] -j MASQUERADE

# Destination-NAT: Empfaenger zu 123.123.123.123, Port 22 aendern
$> iptables [...] -j DNAT --to-destination 123.123.123.123:22

# Umleitung: Auf lokalen Port 8080 umleiten
$> iptables [...] -j REDIRECT --to-ports 8080
```

Beschreibung möglicher Aktionen

Nun, da die iptables-Optionen erläutert sind, ist es an der Zeit, die vier Aktionsziele etwas genauer zu beschreiben:

Source-NAT (SNAT) - Statisches Verändern des Absenders

Im Beispiel auf der vorigen Seite, wo ein lokales Netz an das Internet angebunden wird, haben wir bereits Source-NAT (kurz: SNAT) betrieben. Dabei wird - wie der Name schon sagt - die Adresse des Absenders geändert. Weshalb wir als Aktion dennoch MASQUERADE ausgewählt haben, hat den folgenden Grund: Bei SNAT gibt man die Quell-IP (und ggf. den Quellport) explizit an. Hat ein Server eine statische IP-Adresse, ist SNAT die bessere Wahl, weil es schneller als MASQUERADE ist, welches ja bei jedem Paket die aktuelle IP überprüfen muss. SNAT macht nur Sinn, wenn ein Paket den Rechner verlässt, daher ist die zugehörige Kette die POSTROUTING-Kette.

```
# Optionen für SNAT (Auszug aus manual page)
--to-source <ipaddr>[-<ipaddr>][:port-port]
```

MASQUERADE - Ändern des Absenders auf die lokale IP-Adresse

Mit dem MASQUERADE-Ziel erhält jedes ausgehende Paket als Absender die IP des ausgehenden Interfaces am Router. Der Vorteil gegenüber SNAT liegt darin, dass bei dynamisch zugewiesenen IPs vom Provider jedes Paket automatisch mit der richtigen IP versehen wird, während man bei gewöhnlichem SNAT die Adresse von Hand anpassen müsste. Wie schon bei SNAT ist die zu verwendende Kette die POSTROUTING-Kette. MASQUERADE bietet keine weiteren Optionen.

Destination-NAT (DNAT) - Ändern des Empfängers

Möchte man den Empfänger eines Paketes ändern, so ist DNAT die richtige Wahl. Dies macht beispielsweise Sinn, wenn man hinter einer Firewall Server laufen lassen möchte, dazu später mehr. Aus naheliegenden Gründen muss der Empfänger geändert werden, bevor die Routing-Entscheidung stattfindet, die korrekte Kette ist daher die PREROUTING Kette bzw. für lokal erzeugte Pakete die OUTPUT-Kette.

```
# Optionen für DNAT (Auszug aus manual page)
--to-destination <ipaddr>[-<ipaddr>][:port-port]
```

REDIRECT - Auf den lokalen Recher umleiten

Als spezielle Art des DNAT gibt es das Ziel REDIRECT. Es leitet ein Paket auf einen Port am Router um, was beispielsweise für transparentes Proxying nützlich ist. Wie schon bei DNAT sind die passenden Ketten die PREROUTING-Kette bzw. die OUTPUT-Kette.

```
# Optionen für REDIRECT (Auszug aus manual page)
--to-ports <port>[-<port>]
```

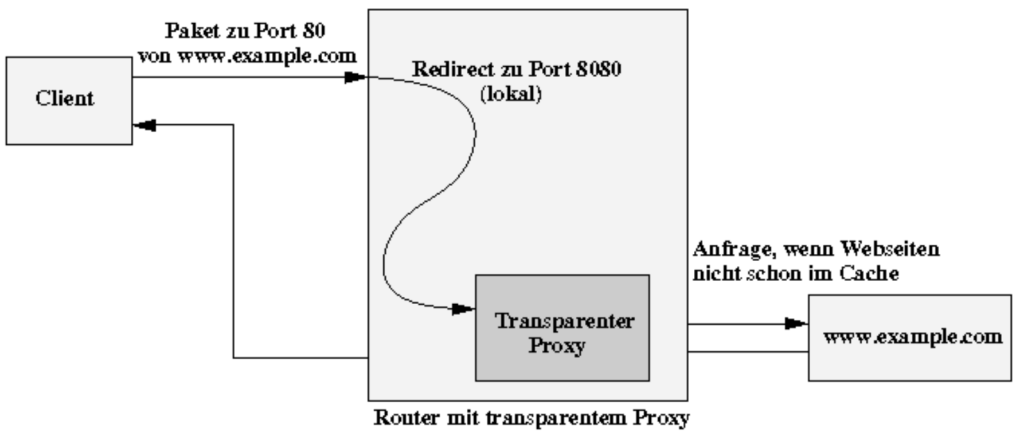
Anwendungen

Beim ersten Beispiel, dem Anschließen eines lokalen Netzes an das Internet, hat sich vielleicht der eine oder andere Leser gedacht: "Wie kommt man auf so kryptische Befehle?". Nun, nach auf den vorangegangenen Erklärungen mag diese Frage sich zu "Was mache ich jetzt nur mit so kryptischen Befehlen?" geändert haben. Diese Frage soll in diesem Abschnitt beantwortet werden. Natürlich ist mir bewusst, dass die Anwendungsmöglichkeiten nahezu unbegrenzt sind, ich werde also versuchen, möglichst viel abzudecken.

Transparentes Proxying

Angenommen, wir haben ein lokales Netz über NAT an das Internet angeschlossen. Um den Traffic etwas geringer zu halten, möchten wir auf der Routing-Maschine einen Proxy auf Port 8080 laufen lassen, über den jeglicher HTTP-Traffic geht.

Die erste (und vermutlich naheliegendste) Möglichkeit wäre, jeden Benutzer dazu "bewegen", an seinem Rechner den Proxy einzutragen und dafür Port 80 (=HTTP-Standard-Port) ganz zu sperren. Ob das eine zufriedenstellende Lösung ist, muss ein Admin für sich selber entscheiden, auf jeden Fall erspart er sich manche Nachteile des transparenten Proxyings.



Mit NAT haben wir allerdings eine zweite Möglichkeit, nämlich alle Pakete, die an einen Port 80 gehen sollen, auf Port 8080 am lokalen Rechner umzuleiten. Der entsprechende lautet dann:

```
# Transparentes Proxying:
# (lokales Netz an eth0, Proxy auf Port 8080)
$> iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 \
   -j REDIRECT --to-ports 8080
```

Damit das funktioniert, muss natürlich auch ein HTTP-Proxy auf Port 8080 laufen und entsprechend für transparentes Proxying konfiguriert (oder gar kompiliert) sein. Probleme mit transparentem Proxying sind zum einen die höhere CPU-Last (ausschlaggebend in wirklich großen Netzen) und zum anderen eventuelle kleine Probleme mit alten bzw. simplen Browsern.

Hilfe, ich bin hinter einer Firewall!

Bevor wir loslegen, noch eine kleine Warnung:

Jede(r) muss VORHER abklären, ob die folgenden Schritte/Befehle für sich geltende Nutzungsbedingungen verletzen! Verwenden der nachfolgenden Befehle erfolgt auf eigene Gefahr, ich übernehme keine Verantwortung für Schäden jeglicher Art bei unsachgemäßer Verwendung der folgenden Befehle bzw. Techniken!

So paradox das klingen mag, auch hier kann NAT abhelfen. Angenommen, der (freiwillig oder unfreiwillig gewählte) Provider bietet - aus welchen Gründen auch immer - nur wenige offene Ports an. Der erste Schritt ist, die offenen Ports herauszufinden. Dazu kann man beispielsweise nmap verwenden:

```
# Einen entfernten Rechner mit nmap scannen:
# Statt www.example.com ist ein passender Rechner zu waehlen
$> nmap www.example.com
```

Die Ausgabe sollte dann ein paar Ports ausgeben, die meisten vermutlich im Zustand "closed". Angenommen, alle Ports unter 5000 sind gesperrt mit Ausnahme von Port 80 (http), dafür sind die Ports ab 5000 in ausreichender Menge geöffnet. Um dennoch eine Verbindung zu einem entfernten Rechner an den Ports kleiner 5000 aber ungleich 80 zu ermöglichen, benötigt man einen (Linux-)Rechner außerhalb der Firewall (ganz egal wo, er darf nur nicht selbst hinter einer Firewall oder einem NAT-Router sitzen), den man mittels iptables konfigurieren kann.

Zuerst müssen wir uns SSH-Zugriff auf besagten Rechner (zukünftig mit IP 111.111.111.111 identifiziert) außerhalb der Firewall verschaffen. Dazu suchen wir (irgend-)einen Rechner mit Verbindungsmöglichkeit zu Port 22 (ssh) und loggen uns ein. Der Befehl

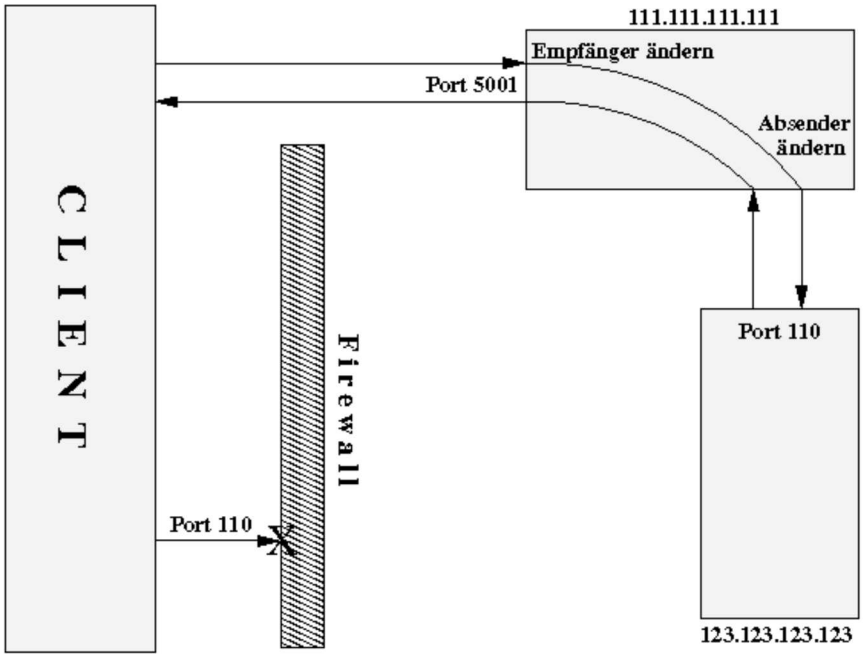
```
# SSH von Port 5000 auf Port 22 umleiten:
$> iptables -t nat -A PREROUTING -p tcp --dport 5000 -j REDIRECT --to-ports 22
```

ist ausreichend, um uns von hinter der Firewall Zugriff über SSH auf den Umleite-Rechner 111.111.111.111 zu verschaffen. Alternativ kann man überhaupt SSH auf Port 5000 laufen lassen, man muss das nur allen Nutzern mitteilen und hätte gleichzeitig die Probleme mit Zombie-PCs, die sich über SSH einloggen wollen, vom Hals.

Nun können wir uns der Reihe nach andere Ports auf anderen Rechnern freischalten, exemplarisch sei das für Port 110 (pop3) gezeigt:

```
# Port 5001 auf Port 110 (pop3) auf 123.123.123.123 weiterleiten:
$> iptables -t nat -A PREROUTING -p tcp --dport 5001 \
   -j DNAT --to-destination 123.123.123.123:110

# Absender auf umleitenden Rechner ändern:
$> iptables -t nat -A POSTROUTING -p tcp --dport 110 \
   -j MASQUERADE
```



Statt letzterem MASQUERADE-Befehl wäre auch SNAT (*-j SNAT --to-source 111.111.111.111*) denkbar, da man aber vermutlich der einzige Nutzer der Umleitung ist, gibt es keine stichhaltigen Gründe für das eine oder das andere, außer dass eben der MASQUERADE-Befehl kürzer ist.

Ebenso können alle anderen Ports freigeschalten werden, solange man offene Ports in der Firewall findet bzw. hat. Selbst sichere Verbindungen wie IMAPS können auf diese Weise weitergeleitet werden, gegebenenfalls sind Warnung bzgl. Zertifikate in Kauf zu nehmen. In den Anwendungen muss dann der jeweilige umleitende Rechner angegeben werden, wenn man also in obigem Beispiel POP3-Mails abrufen möchte, dann muss man zum umleitenden Rechner auf Port 5001 verbinden.

Nehmen wir nun zusätzlich an, der "Provider" hätte Port 80 überwacht, also leitet die Verbindung über einen Proxy und prüft dort den Content. Wenn wir mit dieser Überprüfung nicht einverstanden sind und eine Umgehung nicht verboten ist, dann können wir auf unserem Umleite-Rechner 111.111.111.111 selbst einen transparenten Proxy z.B. auf Port 5002 laufen lassen. Am lokalen (!!) Rechner hinter der Firewall sollte dann

```
# http-Traffic an Port 80 auf 111.111.111.111:5002 umleiten:
$> iptables -t nat -A OUTPUT -p tcp --dport 80 \
   -j DNAT --to-destination 111.111.111.111:5002
```

genügen, um frei von Überwachung zu surfen. (*Ich persönlich finde, einen transparenten Proxy mit einem transparenten Proxy auszutricksen hat Stil! :-)*) Alternativ kann man auch den Proxy von Hand im Browser eintragen, dass könnte aber zu Problemen bei Programmen, die keinen Proxy unterstützen, führen. Dafür erspart man sich Nachteile, die ein transparenter Proxy mit sich bringen kann.

Zum Abschluss dieses Unterabschnitts vielleicht noch einmal eine kurze Zusammenfassung des selbigen: Ein offener Port in der Firewall wird für eine SSH-Verbindung verwendet, die anderen offenen Ports können auf passende Rechner umgebogen werden. Wenn man das Umleiten noch dynamisch hinbekommt, dann reicht ein offener TCP-Port und je ein offener TCP- und UDP-Port in der Firewall, um jeden öffentlichen Serverprozess im Internet per TCP und UDP zu erreichen! Nachteil dieser Zwei-Port-Variante: Es ist immer nur eine Verbindung gleichzeitig möglich.

Server hinter einer Firewall

Hat man einen Server in einem lokalen Netz stehen, das über NAT an das Internet angebunden ist, so hat man von außerhalb vorerst keine Möglichkeit, auf den Server zuzugreifen. Angenommen, wir haben einen HTTP-Server auf 192.168.1.2 laufen, unser Router hat die IP 192.168.1.1 und ist mit dem zweiten Interface (eth1) mit der IP 123.123.123.123 an das Internet angeschlossen. Um von außerhalb auf den HTTP-Server zugreifen zu können, ist am Router der Befehl

```
# http-Traffic auf 192.168.1.2 weiterleiten:
$> iptables -t nat -A PREROUTING -p tcp -i eth1 --dport 80 -j DNAT --to 192.168.1.2
```

notwendig und hinreichend. Nun kann auch von außen auf den Server zugegriffen werden, indem die IP 123.123.123.123 verwendet wird.

Verwandte Artikel

Ähnliche Darstellungen bzw. weiteres Hintergrundwissen ist unter den folgen Adressen zu finden:

- http://iptables-tutorial.frozentux.net/iptables-tutorial.html (english): Sehr umfangreiche Quelle zu Informationen rund um iptables.
- http://www.faqs.org/docs/Linux-mini/TransparentProxy.html (english): Ausführliche Behandlung transparentes Proxyings.
- http://www.barryodonovan.com/publications/lg/108/ (english): Weitere Möglichkeiten des Netfilter-Frameworks durch Verwendung der Erweiterungen.
- http://www.different-thinking.de/linux-nat-iptables.php: Anbindung eines lokalen Netzes an das Internet per NAT, geht detaillierter auf eine iptables-Installation ein und verwendet auch eine rudimentäre Firewall.
- http://users.tkk.fi/~tkarvine/nat-iptables.html (english): Kurze Darstellung der Anbindung eines lokalen Netzes in Verbindung mit einer Firewall
- http://trojaner-und-sicherheit.de/TschiTschi/ip_masquerading.htm: Kurze Erklärung verwendeter Begriffe.

Schlusswort, Danksagung und so weiter

Das netfilter-Framework ist trotz seiner eigentlich simplen Handhabbarkeit sehr mächtig, um den täglichen Tücken des Internets und dessen Routings begegnen zu können. Die anfangs kryptischen Befehle lösen sich nach kurzer Einarbeitung in klare Muster auf, die eine schnelle Administration ermöglichen.

Danken möchte ich in erster Linie jedem Leser, der es bis hierher durchgehalten hat. :-) Ansonsten noch vielen Dank an meinen Laptop mit SUSE 10.1, der die ganze lange Zeit kein einziges Mal abgestürzt ist.

Abschließend möchte ich nocheinmal die Bitte kundtun, bei Wünschen, Anregungen, Beschwerden, etc. die Kommentarmöglichkeiten zu benutzen oder gleich direkt mit mir unter nat@karlrupp.net in Kontakt zu treten.

Kommentare:

robert, am 18. 1. 2014 um 01:18

So, jetzt habe ich das untere Problem nicht nur für einen einzelnen Server sondern für alles was via DNAT in das entsprechende Netz forwarded wird gelöst:
iptables -t nat -I POSTROUTING -p tcp -d MY_NETWORK/NETMASK -j SNAT --to-source MY_GATEWAY_IP

Wobei das MY_NETWORK das Netzwerk ist im welchem sich der zu erreichender Server befindet. MY_NETWORK und MY_GATEWAY_IP sind im selben Netz.
Beispiel:
iptables -t nat -I POSTROUTING -p tcp -d 192.168.1.0/24 -j SNAT --to-source 192.168.1.254

Wenn eine DNAT Regel auf den Server (zb. 192.168.1.20) zeigt, dann kommen die Pakete sauber zu meinen GateWay zurück und alles ist schick.
Vielen Dank für dieses schöne Tut.

robert, am 17. 1. 2014 um 09:40

Dank dieses Tutorials habe ich wenigstens mein Problem verstanden (Ob ich das jetzt richtig formulieren kann ist wieder eine andere Sache).
Ich habe mehrere Netzwerke und würde gerne einen Server extern erreichbar machen. Leider hat der Server ein anderes Gateway und ich kann nichts daran ändern. Mit DNAT auf dem Router kommen die Pakete auf den Server an, aber dann kommen sie natürlich nicht wieder zurück zum Ursprungs-GW sondern suchen sich den schnellsten Weg durch das default GW.
Ich finde einfach nichts, anscheinend kann ich das Problem für die Suchmaschinen in

passende Stichwörter eingrenzen. Hat jemand so ein Problem schon mal gelöst?

Thomas, am 30. 11. 2013 um 14:21

Danke!!!
hilft weiter :)

kvmuser, am 18. 9. 2013 um 20:21

So, hoffentlicher letzer Post zu diesem Thema. Es scheint jetzt tatsächlich zu klappen. Szenario: Host mit einer öffentlichen IP Adresse, darauf ein KVM Gast im privaten Netz. Ziel: beliebige Ports vom Host auf den Gast umleiten, der aber auch Internet-Zugang haben soll. Folgende Regeln scheinen zu funktionieren (hier am Beispiel Port 80):

iptables -t nat -I PREROUTING -p tcp -d öffentliche_IP_Host --dport 80 -j DNAT --to privateIP_Gast:80

iptables -I FORWARD -d privateIP_Gast/32 -p tcp -m state --state NEW,RELATED,ESTABLISHED -m tcp --dport 80 -j ACCEPT

iptables -t nat -I POSTROUTING -p tcp -s privateIP_Gast --sport 80 -j SNAT --to-source öffentliche_IP_Host

kvmuser, am 18. 9. 2013 um 19:08

Sorry für das Gespamme, aber meine Lösung war doch keine Lösung. Es lag nicht am POSTROUTING, es funktioniert also immer noch nicht. Webseiten werden korrekt ausgeliefert, aber aus dem Gast komme ich nicht auf Ressourcen auf Port 80 (also zB wget auf eine http Ressource). Jemand eine Idee????

kvmuser, am 18. 9. 2013 um 10:12

Ich beantworte meine Frage mal eben selbst: Das Problem war scheinbar nicht das Ziel, sondern die fehlende bzw. falsche POSTROUTING chain...

kvmuser, am 18. 9. 2013 um 09:27

Danke für die Antwort. Ich habe die Regeln jetzt geändert auf:
$iptables -t nat -I PREROUTING -p tcp -d 192.168.122.1 --dport 80 -j DNAT --to 192.168.122.43:80

wobei 192.168.122.1 die Bridge auf dem Host ist und 43 die VM. Leider funktioniert das gar nicht, sodaß nun nicht mal mehr die Webseiten ausgeliefert werden. Wie müßte das Ziel in die Regel integriert werden?

kvmuser, am 17. 9. 2013 um 10:58

Endlich mal eine Anleitung für iptables, die ich auch ansatzweise verstehe ;-)

Ich habe ein Problem mit port forwarding in KVM. Ich habe einen Host einen eine VM. Ich möchte port 80 vom Host auf Port 80 des Gastes weiterleiten. Das klappt mit den beiden Regeln ganz gut:
iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to ip.des.gastes:80
iptables -I FORWARD -d ip.des.gastes/32 -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT
Das Problem ist aber: Ich kann aus dem Gast nur teilweise verbindungen nach außen aufbauen. wget zu einer http Ressource klappt nicht (keine Aktivität nach Namensauflösung mehr), zu einer ftp Ressource schon. Das Schlimmste: ein apt-get update funktioniert auch nicht (bleibt nach der Namensauflösung auch bei 0% stehen). Woran kann das liegen? Eingehende Verbindungen auf Port 80 werden wie gewünscht umgeleitet.

Antwort: Schaut so aus, als ob die erste Regel auch alle ausgehenden Pakete auf den Gast umleitet. Hier sollte man also die IP-Adresse des Ziels in die Regel aufnehmen.

Heiko, am 18. 7. 2013 um 12:22

Tolle Anleitung!

Nic, am 28. 3. 2013 um 12:43

Hab die Lösung gefunden: nicht der tatsächliche Port der Quelle wird im header gesetzt sondern aus diesem und aus der IP Adresse der Quelle (der privaten ip adresse) wird ein Schlüssel gebildet. Dieser Schlüssel wird ins Feld Quellport eingetragen. Somit ist bei

Antwortpaketen die umgekehrte Operation möglich und das Paket zuordenbar. Auf alle Fälle super Beitrag auf dieser Seite. Danke schon mal gel!

Bitte, gern geschehen! ;-) Karli

Nic, am 27. 3. 2013 um 21:06

Wenn der Hausbesitzer beim Versenden eines Pakets die Adresse des Untermieters (private IP) mit der eigenen ersetzt (öff IP) plus Postfach des Untermieters (Port), was passiert beim Empfang eines Antwort-Pakets? Der Absender des Pakets hat dieses an die Adresse des Hausbesitzers und Postfach des Untermieters (Port) adressiert. Der Hausbesitzer selbst kennt in diesem Fall lediglich an welches Port (Postfach) er das Paket zusenden muss, aber nicht die Adresse des Untermieters. Zitat: "Ebenso entspricht ein Socket einer Kombination aus Adresse und Postfach" heißt Postfach=Port. Der Hausbesitzer hat im IP Header nur für eine IP Adresse für den Absender Platz (für seine) wird also die IP Adresse wie gesagt gestrichen.. wie wird also ein ankommendes Paket dem Untermieter zugeordnet? Oder wird auch MAC in die Tabelle eingetragen? Sollte nicht MAC das Postfach sein?

Martin, am 29. 12. 2011 um 15:32

Das wars :D Cooler Typ besten Dank und einen Guten Rutsch. Wünsche ich dir ;)

*nat
-A POSTROUTING -o eth0 -j MASQUERADE

Vielen Dank

Karl Rupp, am 29. 12. 2011 um 15:16

Wie schaut die Sache aus, wenn du zusaetzlich ein MASQUERADE auf eth0 legst? Ansonsten trage in deinem lokalen DNS eine Aufloesung der Domain domain.de auf 192.168.12.26 ein, damit kannst du zumindest per http drauf zugreifen.

Matin, am 29. 12. 2011 um 14:06

ich komme da einfach nicht weiter in meinem DNS ist eine Weiterleitung zum NS der domain.de eingerichtet.

Ein Bekannter hat mir gesagt das es eventuell daran liegen könnte das die Dynamischen Ports des Webservers nicht wieder zurück geordnet werden können und ich sollte das mit dem MARK Flag versuchen aber dazu findet man nicht viel hat jemand eine Idee?

Martin, am 29. 12. 2011 um 13:11

Ja die Domain.de ist immer auf die IP vom CompanyConnect gestellt.

Wenn die Regeln so einstelle wird der komplette HTTP Traffic an den internen Server gesendet! Richtig? Dann kann ich nicht mehr aufs WWW zugreifen. Wie und wo kann ich den als zuständig Eintragen?

Karl Rupp, am 29. 12. 2011 um 13:02

Hallo Martin!
Ich kenne dein genaues DNS-Setup nicht, vermute aber, dass zusaetzlich noch PREROUTING fuer -i eth0 notwendig ist, da die Pakete der internen Clients ja auf eth0 eintreffen. Versuche daher einmal folgende zusaetzlichen Befehle, wobei 123.123.123.123 die exemplarische oeffentliche IP deines Gateway ist:
-A PREROUTING -i eth0 -p tcp -m tcp -d 123.123.123.123 --dport 25 -j DNAT --to-destination 192.168.12.21
-A PREROUTING -i eth0 -p tcp -m tcp -d 123.123.123.123 --dport 80 -j DNAT --to-destination 192.168.12.26
-A PREROUTING -i eth0 -p tcp -m tcp -d 123.123.123.123 --dport 443 -j DNAT --to-destination 192.168.12.26
Wie wird deine domain.de aufgeloest? Wird die immer auf die lokale IP des Gateway gemappt? Falls ja, koenntest du ja lokal direkt die 192.168.12.xxx als zustaendige Rechner eintragen (lassen).

Nachtrag: Habe noch die Destination -d nachgetragen, sonst wird der gesamte Traffic umgeleitet, was ja auch nicht erwuenscht ist ;-)

Martin, am 29. 12. 2011 um 10:29

Hallo erstmal vielen Dank für dieses gelungene Tutorial. Ich bin seid mehreren Tagen dabei eine Lösung für mein system zu finden.

Ich habe einen Gateway (eth0 = LAN, eth1 = WAN) worüber der komplette Traffic drüber läuft. Hinter dem Gateway steht eine Domäne mit ca. 50 Usern, DNS, MAIL und Webservern. Jetzt möchte ich das der Webserver von außen erreichbar ist dies mache ich mit:

```
*nat
:PREROUTING ACCEPT [251:24054]
:POSTROUTING ACCEPT [36:4623]
:OUTPUT ACCEPT [3:559]
-A PREROUTING -i eth1 -p tcp -m tcp --dport 25 -j DNAT --to-destination 192.168.12.21
-A PREROUTING -i eth1 -p tcp -m tcp --dport 80 -j DNAT --to-destination 192.168.12.26
-A PREROUTING -i eth1 -p tcp -m tcp --dport 443 -j DNAT --to-destination 192.168.12.26
-A POSTROUTING -o eth1 -j MASQUERADE
```

funktioniert von Außen auch ganz Ordentlich. Wen ich die Domain.de aber vom LAN aufrufe wird die Verbindung nicht aufgebaut. DNS kann ich ausschließen.

Ne Idee? Vielen Dank

wanne, am 26. 5. 2011 um 20:30

http://users.tkk.fi/~tkarvine/nat-iptables.html ist ein toter Link

Fabian, am 13. 5. 2011 um 15:15

Hi,

Anleitung hat mir sehr geholfen. Benötige für einen Staging-Server nämlich eine Regel, welche sämtliche Ausgehenden SSH-Verbindungen umleitet. So kann ein 1:1 Testsystem zum Live-Server zur Verfügung gestellt werden, ohne daß die Kunden die Test-Daten erhalten.

Folgendes funktioniert:
iptables -t nat -A OUTPUT -p tcp --dport 22 -j REDIRECT --to-port 2222
iptables -L -t nat gibt die Regel aus.

Beim Löschen der Regel ebenfalls -t nat mitgeben.

Beste Grüße
Fabian

hugo, am 13. 5. 2011 um 08:59

Hallo Karl,

Respekt und Anerkennung. Das ist eine sehr kompetente und kompakte Erklärung.

Gruß

Hugo

Ralf, am 31. 10. 2010 um 16:17

Hallo,

sehr schöne Anleitung. Habe aber noch eine Frage, damit ich sicher bin, daß ich es auch wirklich kapiert habe... ;-)

Nehmen wir mal an, ich habe einen Server (z.B. Ubuntu Server 10.04 LTS), der eine öffentliche (hier natürlich eine fiktive) IP-Adresse 10.10.10.10 auf eth0 hat. Auf diesem Server würde z.B. ssh auf Port 22 laufen.

Auf diesem Server würde ich z.B. via KVM eine virtuelle Maschine installieren, diese hätte dann z.B. die interne IP-Adresse 19.168.1.10, auf dieser virtuellen Maschine würde ebenfalls ssh auf dem Port 22 laufen. Diese Maschine wäre erstens von außen nicht erreichbar, zweitens kann der Port 22 natürlich sowieso nur einmal vergeben werden, daher würde ich für diesen Zugriff z.B. Port 10022 wählen.

Die diversen angesprichenen Module müßten bei Ubuntu eigentlich standardmäßig geladen werden, das IP-Forwarding allerdings ist deaktiviert. Dann müßte ich also folgende Konfiguration dürchführen:

IP-Forwarding im Kernel aktivieren.
$> echo "1" > /proc/sys/net/ipv4/ip_forward

Port 10022 des Hosts auf Port 22 des Gasts weiterleiten:
$> iptables -t nat -A PREROUTING -p tcp --dport 10022 -j DNAT --to-destination 192.168.1.100:22

Absender auf umleitenden Rechner ändern:
$> iptables -t nat -A POSTROUTING -p tcp --dport 10022 -j MASQUERADE

Gibt es eigentlich auch eine Möglichkeit, das Masquerading für einen Bereich zu aktivieren? Dies wäre nützlich, wenn man mehrere Ports für den eingehenden Traffic auf interne Maschinen weiterleiten will, und nicht jeden einzelnen Port beim ausgehenden Traffic maskieren will.

Ansonsten nochmals vielen Dank für die Anleitung und ein schönes Wochenende,

Ralf

Antwort:
Deine Gedanken sind soweit richtig. Du solltest dich jedenfalls auch noch vergewissern, dass das Netzwerkinterfaces des Hosts auch mit der virtuellen Maschine sprechen kann. Es ist an sich möglich, auch port ranges anzugeben:
$> iptables -t nat -A PREROUTING -p tcp --dport 10022 -j DNAT --to-destination 192.168.1.100:22
wird dann etwa zu
$> iptables -t nat -A PREROUTING -p tcp --dport 10022:10030 -j DNAT --to-destination 192.168.1.100:22-30
Allerdings kann ich nicht sagen, ob das Kommando auch so funktioniert, wie man es erwartet. Beim MASQUERADE kann man genauso verfahren, dort sollte es gehen. Alternativ schreibt man sich ein kleines Shellskript, wo die jeweiligen Ports einzeln abgearbeitet werden.

Karl Nabb, am 28. 8. 2010 um 02:46

Gedankt, wurde dir ja schon recht viel,
aber in diesem Fall spiele ich gerne Papagei:

DANKE fuer diese kompakte und hervorragende Zusammenfassung.

Mir hat sie z.B. im konkreten Fall geholfen die Loesung fuer ein IPSec routing problem zu finden
(OpenSWAN GW hinter einer Firewall, mit 2 NICs - LAN und TUNNEL)

iptables -t nat -A PREROUTING -j SNAT --to-source <tunnel-endpunkt-ip>

Karl Rupp, am 20. 8. 2010 um 23:24

Hallo!
Ich habe 'iptrafficvolume' (auf Sourceforge.net zu finden) im Einsatz, da wird Traffic mit iptables geloggt. Kannst ja abkupfern ;-)
Allerdings hat das nichts mit NAT zu tun, ich kann dir daher nicht direkt weiterhelfen.

Schmitt, am 17. 8. 2010 um 12:35

hi, sehr interessant, leider habe ich es noch nicht geschafft, das Volumen (MB..) je IP mit iptables zu listen, wäre schön und wichtig, gibt es da eine Tip??
WRT54GL mit DD-WRT v24 SP2 13064 mini
vielen herzlichen Dank

Markus, am 2. 8. 2009 um 21:54

Danke,

für mich als anfänger in sowas ist das hier genau richtig gut erklärt und schlüssig

1000 Dank.

Daniel, am 6. 10. 2008 um 14:52

Hallo,

finde die Erklärungen zur iptables mit dem Filter NAT sehr gut und verständlich erklärt.

Was mir aber nicht ganz klar ist, ist die beschriebene Netzwerktopologie ab Punkt

Hilfe, ich bin hinter einer Firewall!

Eventuell kann man dazu eine Grafik mit einbinden, die die gegebenheiten aufzeigt.

Gruß,
Daniel

Andre (http://typo3-agentur.typo3-fanworld.de), am 24. 8. 2008 um 00:14

Dieses Tutorial hatte mir sehr geholfen ... Vielen Dank dafür.

sichelmichel, am 15. 7. 2008 um 10:21

vielen dank! du hast mir meinen arsch fuer die morgige klausur mit deiner detaillierten ausarbeitung gerettet :)

neuling, am 7. 7. 2008 um 17:08

wirklich gut - für mich als einsteiger mal schön eine so gute anleitung zu finden

Tabakhase (http://www.Tabakhase.com), am 7. 1. 2008 um 16:21

SEHR SCHÖN!

Ist echt wunderbar, genau was ich gesucht habe =) Danke!

bookmark - Wilkommen in meiner "ServerKram" Favorieten Gruppe!

Letztes Update dieser Seite am 25. 11. 2006