Linux netfilter Hacking HOWTO

Rusty Russell, mailing list `netfilter@lists.samba.org`, ins Deutsche uebersetzt von Melanie Berg `mel@sekurity.de`
v1.0 Tue May 2 14:07:03 CST 2000

Dieses Dokument beschreibt die netfilter-Architektur fuer Linux, wie man sie hacken kann, und einiger darauf aufbauende groessere Systeme, wie Paketfilter, connection tracking und Network Address Translation (NAT). Diese deutsche Uebersetzung steht unter den Bedingungen der GNU General Public License (http://www.gnu.org/copyleft/gpl.html).

Contents

1 Einleitung

Hi Leute.

Dieses Dokument ist eine Reise; manche Teile sind gut besucht, und in anderen Bereichen wirst Du Dich fast alleine fuehlen. Der beste Rat, den ich Dir geben kann, ist Dir eine grosse, heisse Tasse Kaffee oder Schokolade zu besorgen, Dich in einen bequemen Stuhl zu setzen, und den Inhalt in Dich aufzunehmen, bevor Du Dich in die gefaehrliche Welt des Netzwerk-Hackens begibst.

Um die Verwendung der Infrastruktur auf dem Netfilter-Rahmenwerk besser zu verstehen, empfehle ich, das Paketfiltering-HOWTO und das NAT-HOWTO zu lesen. Fuer Informationen ueber Kernel-Programmierung empfehle ich Rusty's Unreliable Guide to Kernle Hacking und Rusty's Unreliable Guide to Kernel Locking.

Dieses Dokument enthaelt Flueche. Das ist ein natuerliches Gewuerz in meiner Sprache, aber Du kannst die Originalversion dieses HOWTOs in American Broadcast uebersetzen, wenn Du den Filter benutzt:

```
sed 's/[^]aeio]ck/reak/'
```

1.1 Was ist Netfilter?

Netfilter ist eine Basis der Paketbehandlung, gehoert aber nicht zu dem normalen Berkeley Socket Interface. Es besteht aus vier Teilen. Zuerst definiert jedes Protokoll 'Hooks' (IPv4 definiert 5), welche wohldefinierte Punkte auf der Reise eines Pakets durch den Protokoll-Stack sind. An jedem dieser Punkte wird das Protokoll Netfilter mit dem Paket und der Hook-Nummer aufrufen.

Zweitens koennen Teile des Kernels sich fuer das Aufpassen auf verschiedene Hooks fuer jedes Protokoll registrieren. Wenn ein Paket also an Netfilter gereicht wird, wird ueberprueft, ob irgendjemand fuer dieses Protokoll oder fuer diesen Hook registriert ist; Wenn ja, bekommt jeder von ihnen der Reihe nach die Chance, das Paket zu untersuchen (es moeglicherweise zu veraendern), das Paket zu verwerfen, es durchzulassen oder Netfilter zu beauftragen, das Paket fuer Userspace einzureihen.

Der dritte Teil besteht darin, dass eingereihte Pakete gesammelt werden (von ip_queue-Treiber), um an den Userspace geschickt zu werden; diese Pakete werden asynchron behandelt.

Der letzte Teil besteht aus coolen Kommentaren im Code und Dokumentation. Das ist Bestandteil eines jeden experimentiellen Projekts. Das Netfilter- Motto ist (schamlos von Cort Dougan gestohlen):

```
'' Also... wo ist das hier besser als KDE? ''
```

(Dieses Motto hat sich ein wenig ausgeweitet: "Peitsch mich aus, schlag mich, lass mich ipchains benutzen").

Zusaetzlich zu diesem einfach Rahmenwerk wurden verschiedene Module geschrieben, welche Funktionalitaeten aehnlich zu frueheren (pre- netfilter) Kernel bieten, im Besonderen ein erweiterbares NAT-System, und ein erweiterbares Paketfilter-System (iptables).

1.2 Was stimmte nicht mit dem, was wir im 2.0er und 2.2er Kernel hatten?

1. Keine ausgebaute Infrastruktur, um Pakete an Userspace weiterzugeben:
   - Kernelprogrammierung ist schwer
   - Kernelprogrammierung muss in C/C++ gemacht werden
   - Dynamische Filter-Policies gehoeren nicht in den Kernel
   - 2.2 fuehrte ein, Pakete via netlink zum Userspace zu kopieren, aber das ist langsam, und 'sanity checks' (wenn ein Paket z.B. nur behauptet, von einer existierenden Schnittstelle zu kommen) sind nicht moeglich.
2. Transparente Proxies sind eine Qual:
   - Wir untersuchen **jedes** Paket, um zu sehen, ob ein Socket an diese Adresse gebunden ist

- Root darf an fremde Adressen binden
- Lokal-generierte Pakete koennen nicht redirected werden
- REDIRECT behandelt keine UDP-Antworten: Ein UDP-Named-Paket an 1153 weiterleiten funktioniert nicht, weil manche Clients Antworten von anderen Ports als 53 nicht moegen.
- REDIRECT koordiniert sich nicht mit der TCP/UDP Portzuordnung: Ein Benutzer kann einen Port durch eine REDIRECT-Regeln shadowen.
- Es ist waehrend der 2.1er-Serie mindestens zweimal zusammengebrochen.
- Code ist extrem aufdringlich. Denk an die Statistiken von #ifdef CONFIG_IP_TRANSPARENT_PROXY in 2.2.1: es tritt in 11 Dateien insgesamt 34 mal auf. Im Vergleich dazu taucht CONFIG_IP_FIREWALL in 5 Dateien nur insgesamt 10 mal auf.

3. Es ist nicht moegliche, Paketfilter-Regeln unabhaengig von der Adresse der Schnittstelle zu erstellen:
   - Um lokal-generierte oder lokal-bestimmte Pakete von durchgehenden Paketen unterscheiden zu koennen, muss die lokale Adresse der Schnittstelle bekannt sein.
   - Sogar das reicht bei Redirection oder Masquerading nicht aus.
   - Die FORWARD-Kette besitzt nur Informationen ueber die ausgehende Schnittstelle, was bedeutet, dass Du herausfinden musst, woher ein Paket kam, indem Du Dein Wissen ueber die Netzwerk-Topologie anwendest.
4. Masquerading ist abhaengig von Paketfiltern:Interaktionen zwischen Paketfiltern und Masquerading machen eine Firewall komplex:
   - Beim Eingangsfilter scheinen Antwortpakete fuer den Rechner selbst bestimmt zu sein
   - Beim Forward-Filter werden demaskierte Pakete ueberhaupt nicht gesehen
   - Beim Ausgangsfilter scheinen Pakete vom lokalen Rechner zu kommen
5. TOS-Manipulation, Redirect, ICMP unreachable und mark (welche Effekte auf Portforwarding, Routing und QoS haben koennen) haengen ebenfalls vom Paketfilter-Code ab.
6. ipchains-Code ist weder modular noch erweiterbar (zum Beispiel MAC- Adressen Filter, Filtern nach Optionen, etc).
7. Mangel an ausreichender Infrastruktur hat zu einem Ueberfluss von von verschiedenen Techniken gefuehrt:
   - Masquerading plus Per-Protokoll Module
   - Schnelles, statisches NAT durch Routing Code (hat keine Per- Protokoll Routinen)
   - Portforwarding, Redirect, Auto-Forwarding
   - Das Linux NAT und virtuelle Server Projekte.
8. Inkompatibilitaet zwischen CONFIG_NET_FASTROUTE und Paketfiltern:
   - Weitergeleitete Pakete gehen trotzdem durch drei Ketten
   - Keine Moeglichkeit, zu sagen, ob diese Ketten umgangen werden koennen
9. Wegen Routing-Protection (z.B. Verifikation der Quelladresse) keine Moeglichkeit, verworfene Pakete zu untersuchen.

10. Keine Moeglichkeit, automatisch die Zaehler auf Paketfilter-Regeln zu lesen.
11. CONFIG_IP_ALWAYS_DEFRAG ist eine Compilezeit-Option, die das Leben fuer Distributionen, die einen general-purpose Kernel wollen, schwer macht.

1.3 Wer bist Du?

Ich bin der einzige, der dumm genug ist, das zu tun. Als ipchains Co-Autor und jetziger Linux Kernel IP Firwall Maintainer sehe ich viele der Probleme, die die Leute mit dem jetzigen System haben, da ich erkenne, was sie alles machen wollen.

1.4 Wieso stuerzt es ab?

Woah! Du haettest es **letzte** Woche sehen sollen!

Weil ich kein so guter Programmierer bin, wir wir alle es uns wuenschen wuerden, und ich natuerlich nicht alle Szenarien getestet habe, wegen Mangel an Zeit, Ausstattung und/oder Inspiration. Ich habe eine Testumgebung, und ich will Dich ermutigen, daran teilzunehmen.

2 Wo kann ich die neueste Version herbekommen?

Es gibt einen CVS Server auf samba.org, der die neuesten HOWTOs, Anwender- tools und Testseiten enthaelt. Wenn Du browsen moechtest, kannst Du das Webinterface benutzen:

Web Interface http://www.samba.org/cgi-bin/cvsweb/netfilter/ .

Um die neuesten Quellcodes zu bekommen, kannst Du folgendes tun:

1. Log Dich anonym auf dem SAMBA CVS Server ein:

   ```
   cvs -d :pserver:cvs@cvs.samba.org:/cvsroot login
   ```

2. Wenn Du nach einem Passwort gefragt wirst, tippe 'cvs'.
3. Ueberpruefe den verwendeten Code:

   ```
   cvs -d :pserver:cvs@cvs.samba.org:/cvsroot co netfilter
   ```

4. Um ein Update auf die neueste Version zu bekommen, verwende

   ```
   cvs update -d -P
   ```

3 Netfilter Architektur

Netfilter ist mehr eine Serie von Hooks auf bestimmten Positionen in einem Protokoll-Stack (zur Zeit IPv4, IPv6 und DECnet). Das (idealisierte) IPv4 Reise-Diagramm sieht folgendermassen aus:

```
Ein Paket, das das Netfilter System durchreist:

   --->[1]--->[ROUTE]--->[3]--->[4]--->
                 |            ^
                 |            |
                 |         [ROUTE]
                 v            |
                [2]          [5]
                 |            ^
                 |            |
                 v            |
```

Links geht das Paket ein: Den einfachen sanity-check ueberstanden (ich meine, nicht verstuemmelt, Checksumme OK, kein promiscuous receive), werden sie an den NF_IP_PRE_ROUTING [1] Hook des Netfilter Rahmenwerks weitergereicht.

Als naechstes kommen sie in den Routing-Code, welcher entscheidet, ob das Paket fuer eine andere Schnittstelle oder fuer einen lokalen Prozess bestimmt ist. Der Routing-Code kann unroutebare Pakete verwerfen.

Wenn es fuer den Rechner selbst bestimmt ist, wird der NF_IP_LOCAL_IN [2] Hook des Netfilter Rahmenwerks wieder aufgerufen, bevor das Paket an den Prozess (wenn es einen gibt) geschickt wird.

Wenn der Zielort stattdessen eine andere Schnittstelle ist, wird der NF_IP_FORWARD [3] Hook des Netfilter Rahmenwerks stattdessen aufgerufen.

Das Paket durchlaeuft dann den letzten Netfilter Hook, den NF_IP_POST_ROUTING [4] Hook, bevor es wieder in die Leitung geschickt wird.

Fuer Pakete, die lokal generiert wurden, wird der NF_IP_LOCAL_OUT [5] Hook aufgerufen. Hier kannst Du sehen, dass Routing erst dann einsetzt, wenn dieser Hook aufgerufen wurde: Tatsaechlich wird zuerst der Routing-Code aufgerufen (um Angaben ueber Quelladresse und IP-Optionen zu bestimmen) und wird erneut aufgerufen, wenn das Paket geaendert werden sollte.

3.1 Netfilter Grundlagen

Jetzt haben wir ein Beispiel von netfilter fuer IPv4, Du kannst sehen, wann jeder Hook aktiviert wird. Das ist die Essenz von Netfilter.

Kernelmodule koennen sich registrieren, um an irgendeinem dieser Hooks lauschen. Wenn dieser Netfilter Hook dann vom Networking-Code aufgerufen wird, hat an diesem Punkt jedes registrierte Modul die Moeglichkeit, das Paket zu veraendern. Das Modul kann Netfilter sagen, eine von drei Sachen zu tun:

1. NF_ACCEPT: Die Reise wie gewoehnlich fortsetzen.
2. NF_DROP: das Paket verwerfen, die Reise nicht fortsetzen.
3. NF_STOLEN: Ich habe das Paket uebernommen, setz die Reise nicht fort.
4. NF_QUEUE: Das Paket einreihen (gewoehnlich fuer Userspace).
5. NF_REPEAT: Diesen Hook erneut aufrufen.

Die anderen Teile von Netfilter (eingereihte Pakete behandeln, coole Kommentare) werden spaeter in der Kernel-Sektion erklaert.

Auf diesem Fundament koennen wir ziemliche komplexe Paketfilter- Manipulationen aufbauen, wie in den naechsten zwei Sektionen gezeigt werden wird.

3.2 Paketauswahl: iptables

Ein System zur Paketauswahl mit dem Namen IP-Tables wurde ueber das Netfilter Rahmenwerk gebaut. Es ist ein direkter Abkoemmling von ipchains (welches von ipfwadm abstammt, welches, wenn ich mich richtig erinnere, von BSD's ipfw abstammt), mit Erweiterungen. Kernelmodule koennen eine neue Tabelle registrieren und von einem Paket verlangen, eine vorgegebene Tabelle zu durchwandern. Diese Methode zur Paketauswahl wird fuer Paketfilter (die 'Filter'-Tabelle), Network Address Translation (die 'NAT'-Tabelle) und allgemeine Behandlung von pre-routing Paketen (die 'mangle'-Tabelle) verwendet.

3.2.1 Paketfiltern

Diese Tabelle, 'filter', sollte die Pakete niemals veraendern: sie soll sie nur filtern.

Einer der Vorteile von iptables Filtern gegenueber ipchains ist, dass es klein und schnell ist, und es die Netfilter-Hooks NF_IP_LOCAL_IN, NF_IP_FORWARD und NF_IP_LOCAL_OUT verwendet. Das bedeutet, dass es fuer jedes gegebene Paket einen (und nur einen) moeglichen Ort gibt, um es zu filtern. Dies vereinfacht die Dinge ungemein. Ausserdem bedeutet der Fakt, dass das Netfilter Rahmenwerk beides, eingehende und ausgehende Schnittstellen fuer den NF_IP_FORWARD Hook bietet, dass viele Arten des Filterns weitaus einfacher werden.

Beachte: Ich habe die Kernelteile sowohl von ipchains als auch von ipfwadm zu Modulen auf Netfilter portiert, was es ermoeglicht, die alten ipfwadm und ipchains Anwendungstools ohne ein Upgrade weiterzuverwenden.

3.2.2 NAT

Dies ist das Koenigreich der NAT-Tabelle, welche mit Paketen von drei Netfilter-Hooks gefuettert wird: fuer nicht-lokale Pakete sind fuer Quell- und Zielveraenderungen jeweils der NF_IP_PRE_ROUTING und der NF_IP_POST_ROUTING Hook perfekt. Um das Ziel von lokalen Paketen zu veraendern, wird der NF_IP_LOCAL_OUT Hook verwendet.

Diese Tabelle unterscheidet sich insoweit leicht von der 'Filter'-Tabelle, als dass nur das erste Paket einer neuen Verbindung die Tabelle durchwandern wird: Das Resultat dieser Untersuchung wird dann auf alle weiteren Pakete derselben Verbindung angewandt.

Masquerading, Portforwarding, transparente Proxies Ich unterteile NAT in Source NAT (wo die Quelle des ersten Pakets veraendert wird) und in Destination NAT (wo das Ziel der ersten Pakets veraendert wird).

Masquerading ist eine spezielle Form von Source NAT: Port-Forwarding und transparente Proxies sind eine spezielle Form von Destination NAT. Dies wird nun alles mit dem NAT-Rahmenwerk erledigt, anstatt unabhaengige Einheiten zu sein.

3.3 Connection Tracking

Connection Tracking ist ein Fundament von NAT, wurde aber als separates Modul implementiert; dies erlaubt es, durch eine Erweiterung der Paketfilter einfach und sauber Connection Tracking zu verwenden (das 'state' Modul).

3.4 Sonstige Moeglichkeiten

Die neue Flexibilitaet bietet sowohl die Moeglichkeit, wirklich abgefahrene Dinge zu tun, als auch Erweiterungen oder einen kompletten Ersatz zu schreiben, die auch vermischt werden koennen.

4 Informationen fuer Programmierer

Ich weihe Dich in ein Geheimnis ein: mein Hamster hat alles programmiert. Ich war nur das Medium, wenn Du es so willst das 'Front-End', im grossen Plan meines Hamsters. Also mach mich nicht fuer irgendwelche Bugs verantwortlich. Mach den schlauen mit dem Fell verantwortlich.

4.1 ip_tables verstehen

iptables besteht einfach aus einem benannten Array von Regeln im Speicher (daher der Name 'iptables') und Informationen darueber, wo Pakete von jedem Hook die Reise beginnen sollen. Nachdem eine Tabelle registriert wurde, kann ihr Inhalt von Anwenderseite her gelesen oder ersetzt werden, indem getsockopt() und setsockopt() benutzt werden.

iptables registriert nicht mit irgendwelchen Netfilter Hooks: es verlaesst sich auf andere Module, die das tun und es angemessen mit Paketen fuettern sollen.

4.1.1 ip_tables Datenstruktur

Der Bequemlichkeit halber wird dieselbe Datenstruktur verwendet, um eine Regel von Anwenderseite her oder im Kernel selbst zu repraesentieren, obwohl einige Felder nur im Kernel selbst benutzt werden.

Jede Regel besteht aus folgenden Teilen:

1. Einem 'struct ip_entry'.
2. Null oder mehr 'struct ipt_entry_match' Strukturen, an jede Platz fuer Variablen (0 oder mehr Bytes) von Daten angehaengt.
3. Einer 'struct ipt_entry_target' Struktur, an jede Platz fuer Variablen (0 oder mehr Bytes) von Daten angehaengt.

Die Variablen-Natur der Regeln gibt eine grosse Flexibilitaet fuer Erweiterungen, wie wir sehen werden, besonders, da jeder Treffer oder jedes Ziel eine willkuerliche Menge von Daten tragen koennen. Dies birgt zwar auch ein paar Fallen, aber wie auch immer: Wir muessen aufpassen. Wir tun dies, indem wir uns darueber versichern, dass die 'ipt_entry', die 'ipt_entry_match' und die 'ipt_entry_target' Strukturen eine angenehme Groesse haben, und dass alle Daten auf die maximale Auslastung des Rechners aufgerundet werden, indem wir den IPT_ALIGN() Makro verwenden.

'struct ipt_entry' hat die folgenden Felder:

1. Einen 'struct ipt_ip' Teil, der die Bestimmungen fuer den IP-Header enthaelt, der zutreffen soll.
2. Ein 'nf_cache' Bitfeld, das zeigt, welche Teile des Pakets diese Regel untersucht hat.
3. Ein 'target_offset' Feld, das den Absatz vom Anfang dieser Regel angibt, wo die ipt_entry_target Struktur beginnt.
4. Ein 'next_offset' Feld, das die absolute Groesse dieser Regel angibt, einschliesslich Treffer und Ziele.
5. Ein 'comefrom' Feld, das vom Kernel benutzt wird, um die Reise des Pakets zurueckzuverfolgen.
6. Ein 'struct ipt_counters' Feld, was die Paket- und Bytezaehler fuer Pakete, die auf diese Regel gepasst haben, enthaelt.

ipt_entry_match' und 'struct ipt_entry_target' sind sehr aehnlich darin, dass sie ein Feld fuer die Gesamtlaenge (jeweils 'match_size' und 'target_size') enthalten, einen Union, der den Namen des Treffers oder des Ziels (auf Anwenderseite) enthaelt, und einen Pointer (fuer den Kernel).

Wegen der trickreichen Natur der Datenstruktur einer Regel werden einige Hilfsroutinen angeboten:

ipt_get_target()
: Diese Funktion liefert einen Pointer auf das Target einer Regel.

IPT_MATCH_ITERATE()
: Dieser Makro ruft die gegebene Funktion fuer jeden Treffer einer Regel auf. Das erste Argument der Funktion ist 'ipt_match_entry' und andere Argumente (wenn es welche gibt) sind die, die im IPT_MATCH_ITERATE() Makro angeboten werden.

IPT_ENTRY_ITERATE()
: Diese Funktion macht einen Pointer zu einem Eintrag, der Gesamtgroesse der Tabelle der Eintraege und einer Funktion, die aufzurufen ist. Das erste Argument der Funktion ist 'struct ipt_entry', und andere Argumente (wenn es welche gibt) sind die, die im IPT_ENTRY_ITERATE() Makro angeboten werden.

4.1.2 ip_tables aus Sicht der Anwender

Userspace hat vier Operationen: Es kann die aktuelle Tabelle lesen, die Info lesen (Hook-Positionen und Groesse der Tabelle), die Tabelle ersetzen (und die alten Zaehler nehmen), und neue Zaehler einbauen.

So kann Userspace eine kleine Operation simulieren: Dies wird von der libiptc Library getan, welche die einfachen "add/delete/replace" semantics fuer Programme bietet.

Da diese Tabellen in den Kernel transferiert werden, stellt sich die Frage der Auslastung der Maschinen, die unterschiedliche Userspace- und Kernel- space Typ-Regeln haben (z.B. Sparc mit 32-Bit Userland). Diese Faelle werden mit dem Ueberschreiben der Definition von IPT_ALIGN fuer diese Platformen in 'libiptc.h' behandelt.

4.1.3 ip_tables verwenden und erforschen

Der Kernel beginnt die Untersuchung an der Stelle, die durch einen bestimmten Hook definiert ist. Diese Regel wird ueberprueft; wenn das 'struct ipt_ip' Element zutrifft, wird jedes 'struct ip_entry_match' der Reihe nach untersucht (es wird die match-Funktion entsprechend dem Treffer aufgerufen). Wenn die match-Funktion 0 liefert, endet die Iteration auf dieser Regel. Wenn sie den 'hotdrop' Parameter auf 1 setzt, wird das Paket ebenfalls sofort verworfen werden (Dies wird fuer einige verdaechtige Pakete benutzt, so wie in der TCP-match Funktion).

Wenn diese Iteration bis zum Ende durchlaeuft, werden die Counter erhoeht, 'struct ipt_entry_target' wird untersucht: Wenn es ein Standardtarget ist, wird das 'verdict' Feld gelesen (negativ bedeutet ein Urteil ueber das Paket, positiv steht fuer ein Offset, zu dem man springen kann). Wenn die Antwort positiv ist und das Offset nicht das der naechsten Regel ist, wird die 'back' Variable gesetzt, und der vorige 'back' Wert wird in das 'comefrom' Feld der Regel geschrieben.

Fuer nicht-standard Targets wird die target Funktion aufgerufen: Sie liefert ein Urteil (nicht-standard Ziele koennen zu nichts springen, das wuerde den statischen Loop-Detection-Code brechen). Das Urteil kann IPT_CONTINUE sein, um bei der naechsten Regel weiterzumachen.

4.2 iptables erweitern

Weil ich so faul bin, ist `iptables` leicht erweiterbar. Dies ist hauptsaechlich der Versuch, die Arbeit auf die anderen zu schieben, worum es bei Open Source eigentlich geht (Bei Freier Software, wie RMS sagen wuerde, geht es um Freiheit, und ich sass bei einem seiner Gespraeche dabei, als ich das hier schrieb).

`iptables` erweitern besteht aus zwei Teilen: den Kernel erweitern, indem man ein neues Modul schreibt, und moeglicherweise das Anwendungsprogramm `iptables` erweitern, indem man eine neue Shared Library schreibt.

4.2.1 Der Kernel

Ein Kernelmodul selbst zu schreiben ist ziemlich einfach, wie Du an den Beispielen sehen kannst. Eine Sache, auf die Du achten musst, ist, dass Dein Code integriert werden muss: Es kann ein Paket von Anwenderseite her ankommen, waehrend ein anderes an einem Interrupt eintrifft. Tatsaechlich kann es ab 2.3.4 bei SMP vorkommen, dass ein Paket an einem Interrupt pro CPU eintrifft.

Folgende Funktionen musst Du kennen:

init_module()
: Das ist der Eintrittspunkt des Moduls. Es liefert eine negative Zahl oder 0, wenn Netfilter es erfolgreich registriert.

cleanup_module()
: Das ist der Austrittspunkt des Moduls; es soll die Registration beim Kernel wieder aufheben.

ipt_register_match()
: Dies wird verwendet, um einen neuen, zutreffenden Typ zu registrieren. Du uebergibst ihm ein 'struct ip_match', was gewoehnlich als statische Variable deklariert wird (file-scope).

ipt_register_target()

Dies wird verwendet, um einen neuen Typ zu registrieren. Du uebergibst ihm ein 'struct_ip_target', was gewoehnlich als statische Variable deklariert wird (file-scope).

ipt_unregister_target()

Um die Registration des Ziel aufzuheben.

ipt_unregister_match()

Um die Registration des Treffers aufzuheben.

Eine Warnung zu trickreichen Dingen (wie das Verwenden von Countern) fuer Deine neuen Treffer oder Ziele. Auf SMP-Maschinen wird die vollstaendige Tabelle mit Hilfe von memcpy fuer jede CPU dupliziert: Wenn Du die Information wirklich zentral halten willst, solltest Du Dir die Treffermethode zu 'limit' ansehen.

Neue Match-Funktionen Neue match-Funktionen werden gewoehnlich als alleinstehende Module geschrieben. Es ist moeglich, diese Module der Reihe nach zu erweitern, obwohl es normalerweise nicht noetig ist. Eine Moeglichkeit, um den Benutzern zu erlauben, direkt mit Deinem Modul zu sprechen, ist es, die 'nf_register_sockopt' Funktion des Netfilter Rahmenwerks zu nutzen. Eine andere Moeglichkeit ist es, Symbole fuer andere Module zu exportieren, damit sie sich registrieren koennen; auf diese Weise machen es Netfilter und iptables.

Das Wesentliche Deiner neuen Match-Funktion ist 'struct ipt_match', was an 'ipt_register_match()' uebergeben wird. Diese Struktur hat die folgenden Felder:

list

Dieses Feld ist auf irgendeinen Bloedsinn gesetzt, z.B. '{NULL, NULL}'.

name

Dieses Feld ist der Name der Match-Funktion, auf die sich der Anwender bezieht. Der Name sollte dem Modul entsprechend gewaehlt werden (wenn der Name "mac" ist, muss das Modul "ipt_mac.o" heissen), damit es automatisch geladen werden kann.

match

Diese Feld ist ein Pointer auf eine Match-Funktion, welche aus einem skb, dem Pointer auf ein Eingangs- und ein Ausgangsgeraet (eins davon kann NULL sein, das haengt von dem Hook ab), einem Pointer auf die Match-Daten der zutreffenden Regel, der Groesse dieser Regel, dem IP-Offset (nicht Null steht fuer ein non-head Fragment), einem Pointer auf den Protokoll-Header (ich meine nur den IP-Header), der Gesamtlaenge der Daten (ich meine die Paketlaenge minus der IP-Header Laenge) und schliesslich einem Pointer auf die 'hotdrop' Variable besteht. Es soll nicht Null liefern, wenn die Regel auf das Paket zutrifft, und kann 'hotdrop' auf 1 setzen, wenn es Null liefert, um anzuzeigen, dass das Paket sofort verworfen werden soll.

checkentry

Dieses Feld ist ein Pointer auf eine Funktion, die die Bestimmungen fuer eine Regel untersucht; Wenn es 0 liefert, wird die Regel des Benutzers nicht akzeptiert werden. Zum Beispiel wird der "tcp" Match Typ nur TCP-Pakete akzeptieren, wenn also der 'struct ipt_ip' Teil der Regel nicht spezifiziert, dass das Protokoll TCP sein muss, wird Null geliefert. Das Tablename Argument erlaubt Deinem Match, zu kontrollieren, in welchen Tabellen es eingesetzt werden kann, und ‘hook_mask’ ist eine Bitmaske von Hooks, von denen diese Regel moeglicherweise aufgerufen wurde: Wenn Dein Match von einem der Netfilter-Hooks keinen Sinn macht, kannst Du das hier umgehen.

destroy

Dieses Feld ist ein Pointer auf eine Funktion, die aufgerufen wird, wenn ein Eintrag, der dieses Match verwendet, geloescht wird. Dies erlaubt Dir auch, Resourcen in checkentry dynamisch zuzuordnen und sie hier wieder aufzuraeumen.

me

Dieses Feld ist auf ’&__this_module’ gesetzt, welches Deinem Modul einen Pointer gibt. Es bewirkt, dass der usage-count steigt und faellt, je nachdem, ob Regeln dieses Typs erstellt oder geloescht werden. Das bewahrt den Benutzer davon, das Modul zu entfernen (und somit das cleanup_module aufzurufen), wenn sich eine Regel darauf bezieht.

Neue Targets Neue Targets werden auch gewoehnlich als alleinstehende Module geschrieben. Die Diskussionen der oberen Sektion ’Neue Match Funktionen’ koennen hier aehnlich angewandt werden.

Das Wesentliche Deines neuen Targets ist ’struct ipt_target’, das an ‘ipt_register_target()’ uebergeben wird. Diese Struktur hat die folgenden Felder:

list

Dieses Feld ist auf irgendeinen Bloedsinn gesetzt, z.B. ’{NULL, NULL}’.

name

Dieses Feld ist der Name der Ziel-Funktion, auf die sich der Anwender bezieht. Der Name sollte dem Modul entsprechend gewaehlt werden (wenn der Name "REJECT" ist, muss das Modul "ipt_REJECT.o" heissen), damit es automatisch geladen werden kann.

target

Diese Feld ist ein Pointer auf eine Ziel-Funktion, welche aus einem skbuff, dem Pointer auf ein Eingangs- und ein Ausgangsgeraet (eins davon kann NULL sein) einem Pointer auf die Ziel-Daten, der Groesse der Ziel-Daten, und der Position der Regel in der Tabelle besteht. Die Target-Funktion liefert eine nicht-negative, absolute Position, zu der zu springen ist, oder ein negatives Urteil (was das negierte Urteil minus 1 ist).

checkentry

Dieses Feld ist ein Pointer auf eine Funktion, die die Bestimmungen fuer eine Regel ueberprueft; wenn das 0 liefert, wird die Regel des Benutzers nicht akzeptiert werden.

destroy

Dieses Feld ist ein Pointer auf eine Funktion, die aufgerufen wird, wenn ein Eintrag, der dieses Match verwendet, geloescht wird. Dies erlaubt Dir auch, Resourcen in checkentry dynamisch zuzuordnen und sie hier wieder aufzuraeumen.

me

Dieses Feld ist auf ’&__this_module’ gesetzt, welches Deinem Modul einen Pointer gibt. Es bewirkt, dass der usage-count steigt und faellt, je nachdem, ob Regeln dieses Typs erstellt oder geloescht werden.

Das bewahrt den Benutzer davon, das Modul zu entfernen (und somit das cleanup_module aufzurufen), wenn sich eine Regel darauf bezieht.

Neue Tabellen Wenn Du willst, kannst Du fuer einen speziellen Zweck eine neue Tabelle erstellen. Um das zu tun, rufst Du 'ipt_register_table()' mit 'struct ipt_table' auf, was die folgenden Felder hat:

list

Dieses Feld ist auf irgendeinen Bloedsinn gesetzt, z.B. '{NULL, NULL}'.

name

Dieses Feld ist der Name der Ziel-Funktion, auf die sich der Anwender bezieht. Der Name sollte dem Modul entsprechend gewaehlt werden (wenn der Name "nat" ist, muss das Modul "ipt_nat.o" heissen), damit es automatisch geladen werden kann.

table

Dies ist ein voll ausgenutztes 'stuct ipt_replace', wie es vom Anwender genutzt werden kann, um eine Tabelle zu ersetzen. Der 'counters' Pointer sollte auf NULL gesetzt sein. Diese Datenstruktur kann als '__initdata' deklariert werden, damit es nach dem Booten verworfen werden kann.

valid_hooks

Dies ist eine Bitmaske der IPv4 Netfilter Hooks, mit der Du die Tabellen betreten wirst: Sie wird verwendet, um zu ueberpruefen, dass diese Eintrittspunkte gueltig sind und um die moeglichen Hooks fuer die ipt_match und die ipt_target 'checkentry() Funktionen zu berechnen.

lock

Dies ist das read-write Lock fuer die vollstaendige Tabelle, initiali- siere es auf RW_LOCK_UNLOCKED.

private

Dies wird vom ip_tables Code intern verwendet.

4.2.2 Anwendertools

Jetzt, wo Du Dein nettes cooles Kernelmodul geschrieben hast, moechtest Du vielleicht seine Optionen von Anwenderseite her kontrollieren. Lieber, als fuer jede Erweiterung von `iptables` eine abgespaltene Version zu haben, benutze ich eine Technologie der spaeten 90er: Furbies. Sorry, ich meine shared Libraries.

Gewoehnlich benoetigen neue Tabellen keine Erweiterungen von `iptables`: Der Benutzer verwendet einfach die '-t' Option, um auf eine neue Tabelle zugreifen zu koennen.

Die shared Libraries sollten eine '_init()' Funktion haben, die beim Laden automatisch aufgerufen wird: Das moralische Gegenstueck zu der 'init_module()' Funktion der Kernelmodule. Dies sollte 'register_match()' oder 'register_target()' aufrufen, abhaengig davon, ob Deine shared Library ein neues Match oder ein neues Ziel ist.

Du brauchst nur dann eine shared Library, wenn Du einen Teil der Struktur initialisieren oder eine zusaetzliche Option bieten willst. Das Ziel 'REJECT' zum Beispiel benoetigt nichts dergleichen, hat also auch keine shared Library.

Es gibt nuetzliche Funktionen, die in 'iptables.h.' beschrieben werden, besonders:

check_inverse()

Ueberprueft, ob eins der Argumente ein '!' ist, und wenn ja, setzt es das 'invert' Flag, wenn das noch nicht geschehen sein sollte. Wenn sie 'true' liefert, solltest Du optind wie im Beispiel erhoehen.

string_to_number()

Konvertiert einen String zu einer Zahl in der gegebenen Groessenordnung. Sie liefert -1, wenn der String ungueltig ist oder ausserhalb der Groessenordnung liegt.

exit_error()

Diese Funktion sollte aufgerufen werden, wenn ein Fehler gefunden wird. Das erste Argument ist gewoehnlich 'PARAMETER_PROBLEM', was bedeutet, dass der User die Kommandozeile nicht richtig benutzt hat.

Neue Match-Funktionen Die _init() Funktion Deiner shared Library uebergibt einen Pointer auf ein statisches 'struct iptables_match' an 'register_match()', was die folgenden Felder hat:

next

Dieser Pointer wird verwendet, um eine verlinkte Liste von Matches zu erstellen (wie sie fuer die listing-Regeln verwendet wird). Am Anfang sollte er auf NULL gesetzt sein.

name

Der Name der Match-Funktion. Er sollte entsprechend dem Namen der Library gewaehlt sein (z.B. "tcp" fuer 'libipt_tcp.so').

version

Dies wird gewoehnlich auf den NETFILTER_VERION Makro gesetzt: Es stellt sicher, dass das `iptables` Binary nicht aus Versehen die falsche Library auswaehlt.

size

Die Groesse der Match-Daten fuer dieses Match; damit Du Dir keine Sorgen machen musst, wird sie von IPT_ALIGN() automatisch aufgerundet werden.

userspacesize

Fuer einige Matches aendert der Kernel intern einige Felder (z.B. im Fall von 'limit'). Das bedeutet, dass ein einfaches 'memcmp' nicht mehr ausreicht, um zwei Regeln zu vergleichen (benoetigt fuer die delete-matching-rule Funktionalitaet). Wenn dies der Fall ist, plaziere all die Felder, die sich nicht aendern, an den Anfang der Struktur, und lege die Groesse der sich nicht aendernden Felder hier ab.

help

Eine Funktion, die die Synopsis der Option ausdruckt.

init

Dies kann verwendet werden, um etwas mehr Platz (wenn vorhanden) in der ipt_entry_match Stuktur zu schaffen, und um jegliche nfcache Bits zu setzen; Wenn Du etwas untersuchst, das Du nicht ausdruecken kannst, indem Du den Inhalt von '/linux/include/ netfilter_ipv4.h' verwendest, wende im NFC_UNKNOWN Bit einfach OR an. Dies wird vor 'parse()' aufgerufen werden.

parse

Dies wird aufgerufen, wenn eine nicht erkannte Option auf der Kommandozeile gesehen wird: Es sollte nicht NULL liefern, wenn diese Option tatsaechlich fuer Deine Library bestimmt war. 'invert' liefert 'true', wenn bereits ein '!' gesehen wurde. Der 'flags' Pointer ist fuer die exklusive Verwendung Deiner Match-Library und wird gewoehnlich dazu verwendet, eine Bitmaske von bestimmten Optionen zu speichern. Vergiss nicht, das nfcache Feld auszurichten. Wenn noetig, kannst Du die Groesse der 'ipt_entry_match' Struktur erhoehen, indem Du Sie erneut zuordnest, dann musst Du aber auch sicherstellen, dass die Groesse durch den IPT_ALIGN Makro gehen muss.

final_check

Dies wird nach ueberstandener Kommandozeileneingabe aufgerufen. Ihm werden die fuer Deine Library reservierten Integer-'flags' uebergeben. Dies gibt Dir die Gelegenheit, zu ueberpruefen, ob alle benoetigten Optionen angegeben wurden, zum Beispiel 'exit_error()' aufzurufen, wenn dies der Fall sein sollte.

print

Dies wird von Chain-Listing-Code verwendet, um jegliche zusaetzliche Match-Information einer Regel (wenn vorhanden) ausdrucken (zum Standard Output). Das numerische Flag wird gesetzt, wenn der User es mit der '-n' Option angegeben hat.

save

Dies ist das Gegenteil von parse: Es wird von 'iptables-save' verwendet, um die Optionen, die die bestimmte Regel erstellt haben, zu reproduzieren.

extra_opts

Dies ist eine NULL-terminierte Liste von Extra-Optionen, die Deine Library anbietet. Dies wird vermischt mit den jetzigen Optionen und so an getopt_long uebergeben; Fuer Details siehe Manpage.

Es gibt noch extra Elemente am Ende dieser Struktur, die `iptables` intern verwendet: Du brauchst sie hier nicht zu setzen.

Neue Targets Die _init() Funktionen Deiner shared Libraries uebergeben 'register_target()' einen Pointer auf ein statisches 'struct iptables_target', welches aehnliche Felder hat wie die iptables_match Strukturen, die oben detailliert beschrieben wurden.

Manchmal braucht ein Target keine Library fuer den Userspace; Eine triviale musst Du sowieso erstellen: Es gab frueher zu viele Probleme mit schlecht plazierten Libraries.

4.2.3 'libiptc' verwenden

`libiptc` ist die Kontroll Library fuer iptables, entworfen, um Regeln im iptables Kernelmodul aufzulisten und zu manipulieren. Waehrend seine jetzige Verwendung sich auf das iptables Tool beschraenkt, macht es das Schreiben neuer Tools recht einfach. Um diese Funktionen nutzen zu koennen, musst Du root sein.

Die Kerneltabellen selbst bestehen nur aus ein paar Tabellen mit Regeln und einer Anzahl von Nummern, die die Eintrittspunkte repraesentieren. Die Namen der Ketten ("INPUT") werden als Abstraktion von der Library unterstuetzt. Benutzerdefinierte Ketten werden benannt, indem ein Fehlerverweis vor dem Beginn der benutzerdefinierten Kette eingefuegt wird, der den Kettennamen in einer speziellen Datensektion des

Targets enthaelt (Die Positionen der eingebauten Ketten werden durch die drei Eintrittspunkte der Tabelle definiert).

Wenn 'iptc_init()' aufgerufen wird, wird die Tabelle einschliesslich der Counter gelesen. Diese Tabelle wird von der 'iptc_insert_entry()', 'iptc_replace_entry()', 'iptc_append_entry()', 'iptc_delete_entry()', 'iptc_delete_num_entry()', 'iptc_flush_entries()', 'iptc_zero_entries()', 'iptc_create_chain()' 'iptc_delete_chain()', und 'iptc_set_policy()' Funktion manipuliert.

Die Aenderungen an der Tabelle werden nicht eher zurueckgeschrieben, bis die 'iptc_commit()' Funktion aufgerufen wird. Das bedeutet, dass es fuer zwei Benutzer der Library gleichzeitig moeglich ist, auf derselben Kette zu operieren, um jeweils der Schnellere zu sein; Um das zu verhindern, ist Locking noetig, was aber zur Zeit nicht eingesetzt wird.

Es gibt kein Wettrennen mit bei Countern, wie auch immer; Counter werden so in den Kernel zurueckaddiert, dass eine Erhoehung der Zaehler zwischen dem Lesen und dem Schreiben der Tabelle immernoch in der neuen Tabelle auftauchen wird.

Es gibt verschiedene Hilfsfunktionen:

iptc_first_chain()

Diese Funktion liefert den Namen der ersten Kette in dieser Tabelle.

iptc_next_chain()

Diese Funktion liefert den Namen der naechsten Kette in dieser Tabelle: NULL bedeutet, es gibt keine weiteren Ketten.

iptc_builtin()

Liefert true, wenn der Name der gegebenen Kette der Name einer eingebauten Kette ist.

iptc_first_rule()

This returns a pointer to the first rule in the given chain name: NULL for an empty chain.

iptc_next_rule()

Dies liefert einen Pointer auf die erste Regel im Namen der gegebenen Kette: NULL fuer eine leere Kette.

iptc_get_target()

Dies gibt das Target der gegebenen Regel. Wenn es kein erweitertes Target ist, wird der Name dieses Targets geliefert. Wenn es ein Sprung auf eine andere Kette ist, wird der Name dieser Kette geliefert. Wenn es ein Urteil (z.B. DROP) ist, wird dieser Name geliefert. Wenn die Regel kein Ziel hat (und eine accountig-style Regel), wird ein leerer String geliefert.

Beachte, dass Du diese Funktion verwenden solltest, anstatt den Wert des 'verdict' Felds der ipt_entry Struktur direkt zu benutzen, da sie die oben genannten weiteren Interpretationen des Standard Urteils bietet.

iptc_get_policy()

Dies liefert die Policy einer eingebauten Kette und fuellt das Counter- argument mit den Treff-Statistiken dieser Policy.

iptc_strerror()

Diese Funktion liefert eine aussagekraeftigere Erklaerung von gescheitertem Code in der iptc Library. Wenn eine Funktion scheitert, wird es errno setzen: Dieser Wert kann an iptc_strerror() uebergeben werden, um eine Fehlermeldung zu erhalten.

4.3 NAT verstehen

Willkommen zu Network Address Translation im Kernel. Beachte, dass die angebotene Infrastruktur mehr fuer Vollstaendigkeit als fuer Effizienz entworfen wurde und dass weitere Entwicklungen die Effiziens erhoehen koennen. Im Moment bin ich froh, dass es ueberhaupt funktioniert.

NAT wird aufgeteilt in Connection Tracking (was die Pakete ueberhaupt nicht veraendert) und in den NAT Code selbst. Connection Tracking kann auch als iptables Modul verwendet werden, um subtile Unterscheidungen von Zustaenden zu machen, die NAT egal sind.

4.3.1 Connection Tracking

Connection Tracking benutzt als Hooks die von der Prioritaet hoehergestellten NF_IP_LOCAL_OUT und NF_IP_PRE_ROUTING, um Pakete zu sehen, bevor sie das System betreten.

Das ncft Feld in skb ist ein Pointer auf einen der infos[] Arrays im Inneren von struct ip_conntrack. So koennen wir den Zustand von skb durch das Element bestimmen, auf welches dieser Array zeigt: Dieser Pointer erkennt beides, die State Struktur und die Beziehung des skb zu diesem Zustand.

Der beste Weg, das 'nfct' Feld zu lesen, ist es, die 'ip_conntrack_get()' Funktion aufzurufen, die NULL liefert, wenn es nicht gesetzt ist, oder den Connection Pointer. Ausserdem fuellt sie ctinfo, was die Beziehung des Paket zu der Verbindung beschreibt. Dieser numerierte Typ hat folgende Werte:

IP_CT_ESTABLISHED

Das Paket ist Teil einer in Originalrichtung aufgebauten Verbindung.

IP_CT_RELATED

Das Paket steht in Zusammenhang mit der Verbindung und geht in die Originalrichtung.

IP_CT_NEW

Das Paket versucht, eine neue Verbindung aufzubauen (offensichtlich geht es in die Originalrichtung).

IP_CT_ESTABLISHED + IP_CT_IS_REPLY

Das Paket ist Teil einer aufgebauten Verbindung, und zwar in Antwort- richtung.

IP_CT_RELATED + IP_CT_IS_REPLY

Das Paket steht in Zusammenhang mit der Verbindung und geht in die Antwortrichtung.

Ein Antwortpaket kann also als solches indentifiziert werden, indem man auf >= IP_CT_IS_REPLY testet.

4.4 NAT/Connection Tracking erweitern

Diese Rahmenwerke wurden entworfen, um jegliche Art von Protokollen und verschiedenen Mapping-Arten unterzubringen. Manche dieser Mapping-Arten koennen sehr spezifisch sein, so wie load-balancing oder fail-over Mappings.

Intern konvertiert Connection Tracking ein Paket zu einem 'Tupel', der den interessanten Teil des Pakets repraesentiert, bevor nach darauf zutreffenden Regeln gesucht wird. Dieser Tupel hat einen manipulierbaren Teil und einen nicht-manipulierbaren Teil; "src" und "dst" genannt, da dies die Ansicht des ersten Pakets in der Source NAT Welt ist (in der Destination NAT Welt wuerde es ein Antwortpaket sein). Die Tupel aller Pakete in demselben Paket-Stream in dieser Richtung sind gleich.

Das Tupel eine TCP-Pakets enthaelt z.B. den manipulierbaren Teil (Quell- IP und Quellport) und den nicht-manipulierbaren Teil (Ziel-IP und Zielport) Der manipulierbare und der nicht-manipulierbare Teil muessen trotzdem nicht vom selben Typ sein; Das Tupel eines ICMP-Pakets enthaelt den manipulierbaren Teil (Quell-IP und ICMP-ID) und den nicht-manipulierbaren Teil (Ziel-IP und ICMP Typ und Code).

Jedes Tupel hat ein Inverses, naemlich das Tupel des Antwortpakets in dem Stream. Das Inverse eines ICMP Ping-Pakets (icmp id 1234, von 192.168.1.1 an 1.2.3.4) ist zum Beispiel ein Ping Antwort Paket (icmp id 1234, von 1.2.3.4 an 192.168.1.1).

Diese Tupel, die von 'struct ip_conntrack_tuple' repraesentiert werden, werden haeufig verwendet. Tatsaechlich ist das, zusammen mit dem Hook, an dem das Paket einging (was einen Effekt auf die zu erwartende Art der Manipulation hat) und dem beteiligten Device, die komplette Information ueber das Paket.

Die meisten Tupel sind Teil von 'struct ip_conntrack_tuple_hash', was einen doppelt verlinkten Listeneintrag und einen Pointer auf die Verbindung, zu der das Paket gehoert, hinzufuegt.

Eine Verbindung wird von 'struct ip_conntrack' repraesentiert: Es hat zwei 'struct ip_conntrack_tuple_hash' Felder: Eins, was sich auf die Richtung des Originalpakets bezieht (tuplehash[IP_CT_DIR_ORIGINAL]), und eins, was sich auf die Antwortrichtung des Pakets bezieht (tuplehash[IP_CT_DIR_REPLY]).

Wie auch immer, das erste, was der NAT Code tut, ist nachzusehen, ob der Connection Tracking Code es geschafft hat, ein Tupel zu extrahieren und ein bestehende Verbindung zu finden, indem er in das nfct Feld von skbuff sieht; Das sagt uns, ob es ein Versuch ist, eine neue Verbindung aufzubauen, oder, wenn nicht, in welche Richtung das ganze geht; im letzteren Fall werden die Manipulationen ausgefuehrt, die vorher fuer diese Verbindung bestimmt wurden.

Wenn es der Anfang einer neuen Verbindung war, suchen wir nach einer Regel fuer dieses Tupel, indem wir den Standard-Mechanismus von iptables verwenden. Trifft eine Regel zu, wir sie verwendet, um Manipulationen sowohl fuer die Original-, als auch fuer die Antwort- richtung auszufuehren; dem Connection Tracking Code wird mitgeteilt, dass die zu erwartenden Antwort sich geaendert hat. Dann wird das Paket wie oben beschrieben veraendert.

Wenn es keine Regel gibt, wird eine 'Null' Bindung erstellt: Das mappt das Paket gewoehnlich nicht, stellt aber sicher, dass wir keinen anderen Stream ueber den existierenden mappen. Manchmal kann die Null-Bindung nicht erstellt werden, weil wir bereits einen existierenden Stream daruebergemappt haben. In diesem Fall koennte die Pre-Protocol Manipulation versuchen, ihn zu remappen, obwohl es vom Namen her keine Null-Bindung ist.

4.4.1 Standard NAT-Targets

NAT Targets sind wie alle anderen iptables Target-Erweiterungen, ausser, dass sie darauf bestehen, nur in der 'nat' Tabelle verwendet zu werden. Sowohl SNAT als auch DNAT Targets verwenden 'struct ip_nat_multi_range' als Extradaten; dies wird benutzt, um einen Bereich von Adressen zu bestimmen, auf denen ein Mapping gueltig ist. Ein Element dieses Bereichs, 'struct ip_nat_range', besteht aus einer inklusiven minimalen und maximalen IP-Adresse und aus einem inklusiven minimalen und maximalen Wert zur Protokollbestimmung (zum Beispiel TCP Ports). Es gibt auch Platz fuer Flags, die uns sagen, ob die IP-Adresse gemappt werden kann (manchmal wollen wir nur den protokoll-spezifischen Teil eines Tupels mappen, nicht die IP), oder ein anderes, das uns sagt, dass der protokoll-spezifische Teil des Bereichs 'valid' ist.

Ein Multi-Range ist ein Array aus diesen 'struct ip_nat_range' Elementen; das bedeutet, dass ein Bereich "1.1.1.1-1.1.1.2 Ports 50-55 AND 1.1.1.3 Port 80" sein kann. Jedes Element wird zu dem Bereich dazuaddiert (Union, fuer die, die diese Theorie moegen).

4.4.2 Neue Protokolle

Der Kernel von innen Ein neues Protokoll zu implementieren bedeutet zuerst, zu entscheiden, was der manipulierbare und was der nicht-manipulierbare Teil des Tupels sein sollen. Jedes Teil des Tupels hat die Eigenschaft, dass es den Stream eindeutig identifiziert. Der manipulierbare Teil des Tupel ist der Teil, mit dem Du NAT machen kannst: Fuer TCP ist das der Quellport, fuer ICMP ist das die ICMP id; etwas, das das "Stream Identifier" benutzt werden kann. Der nicht-manipulierbare Teil ist der Rest des Pakets, der den Stream zwar auch eindeutig identifiziert, mit dem wir aber nicht spielen koennen (z.B. TCP Zielport, ICMP Typ).

Sobald Du Dich dazu entschieden hast, kannst Du eine Erweiterung fuer den Connection Tracking Code schreiben und die 'ip_conntrack_protocol' Struktur, die Du an 'ip_conntrack_register_protocol()' uebergeben musst, ausbauen.

Die Felder von 'struct ip_conntrack_protocol' sind:

list

Setz es auf '{NULL, NULL}'; verwendet, um sich der Liste zu naehern.

proto

Die Nummer Deines Protokolls, siehe '/etc/protocols'.

name

Der Name Deines Protokolls. Das ist der Name, den der Anwender sehen wird; meistens ist es das beste, den entsprechenden Namen aus '/etc/protocols' zu verwenden.

pkt_to_tuple

Die Funktion, die die protokollspezifischen Teile des Tupels ausfuellt, sobald ihr das Paket uebergeben wird. Der 'datah' Pointer zeigt auf den Anfang des Headers (direkt hinter der IP-Adresse), und datalen ist die Laenge des Pakets. Es liefert NULL, wenn das Paket nicht lang genug ist, um Header-Informationen zu enthalten; datalen wird (gezwungenermassen) immer mindestens 8 Byte lang sein.

invert_tuple

Diese Funktion wird einfach dazu verwendet, den protokollspezifischen Teil des Tupels so zu aendern, wie eine Antwort auf dieses Paket aussehen wuerde.

print_tuple

Diese Funktion wird verwendet, um den protokollspezifischen Teil eines Tupels auszudrucken; gewoehnlich wird es mit sprintf() in den mitgegebenen Buffer geschrieben. Geliefert wird die Anzahl von verwendeten Buffer-Zeichen. Dies wird verwendet, um den Status des /proc Eintrags zu drucken.

print_conntrack

Diese Funktion wird benutzt, um den privaten Teil der Conntrack Struktur (wenn vorhanden) zu drucken. Sie wird auch verwendet, um den Status unter /proc zu drucken.

packet

Diese Funktion wird aufgerufen, wenn ein Paket erkannt wird, das Teil einer aufgebauten Verbindung ist. Du bekommst einen Pointer auf die Conntrack Struktur, den IP-Header, die Laenge und ctinfo. Du gibt ein Urteil ueber das Paket zurueck (gewoehnlich NF_ACCEPT), oder -1 wenn das Paket kein gueltiger Teil der Verbindung ist. Innerhalb dieser Funktion kannst Du, wenn Du willst, diese Verbindung loeschen, Du solltest aber folgendes Idiom verwenden, um races zu vermeiden:

```
if (del_timer(&ct->timeout))
        ct->timeout.function((unsigned long)ct);
```

new

Diese Funktion wird aufgerufen, wenn ein Paket zum ersten Mal eine Verbindung aufbaut; es gibt kein ctinfo arg, da das erste Paket der Definition nach ctinfo IP_CT_NEW ist. Wenn es beim Verbindungsaufbau scheitert, liefert die Funktion 0, oder im selben Moment ein Connection Timeout.

Sobald Dein fertig bist und getestet hast, dass Du Dein neues Protokoll einsetzen kannst, ist es an der Zeit, NAT beizubringen, wie es zu uebersetzen ist. Dies bedeutet, ein neues Modul zu schreiben; eine Erweiterung zum NAT-Code und die 'ip_conntrack_protocol' Struktur, die Du an 'ip_conntrack_register_protocol()' uebergeben musst.

list

Setz es auf '{NULL, NULL}'; verwendet, um sich der Liste zu naehern.

name

Der Name Deines Protokolls. Das ist der Name, den der Anwender sehen wird; meistens ist es das beste, den entsprechenden Namen aus '/etc/protocols' zu verwenden, um auto-loading zu ermoeglichen, wie wir spaeter noch sehen werden.

protonum

Die Nummer Deines Protokolls, siehe '/etc/protocols'.

manip_pkt

Das ist die andere Haelfte der von Conntection Tracking benutzten 'ptk_to_tuple' Funktion: Stell sie Dir als "tuple_to_ptk" vor. Trotzdem gibt es ein paar Unterschiede: Du bekommst einen Pointer auf den Anfang des IP-Headers, und die Gesamtlaenge des Pakets. Das liegt daran, dass manche Protokolle (UDP, TCP) den IP-Header kennen muessen. Dir wird das 'ip_nat_tuple_manip' Feld des Tupels gegeben (ich meine das "src" Feld) statt des ganzen Tupels und der Art der Manipulation, die auszufuehren ist.

in_range

Diese Funktion wird verwendet, um zu sagen, ob manipulierbarer Teil des gegebenen Tupels im gegebenen Bereich liegt. Diese Funktion ist ein bisschen trickreich: Wir erhalten die Art der Manipulation, die auf das Tupel angewandt wurde, die uns sagt, wie wir den Bereich interpretieren koennen (ist es ein Quell- oder ein Zielbereich, auf den wir abzielen?).

Diese Funktion wird benutzt, um zu ueberpruefen, ob ein existierendes Mapping uns in den richtigen Bereich bringt, ausserdem noch, um zu ueberpruefen, ob vielleicht ueberhaupt keine Manipulation noetig ist.

unique_tuple

Diese Funktion ist das Herzstueck von NAT: Wir erhalten ein Tupel und einen Bereich und sollen den Per-Protocol Teil des Tupels veraendern, um es im Bereich zu plazieren und somit einzigartig zu machen. Wenn wir kein unbenutztes Tupel in diesem Bereich finden, wird 0 geliefert. Ausserdem bekommen wir einen Pointer auf die Conntrack Struktur, welcher fuer ip_nat_user_tuple() benoetigt wird.

Gewoehnlich wird einfach der Per-Protocol Teil des Tupels durch den Bereich iteriert, und 'ip_nat_user_tupel()' wird solange ueberprueft, bis es einmal false liefert.

Beachte, dass der Null-Mapping Fall bereits ueberprueft wurde: Entweder liegt er ausserhalb des gegebeben Bereichs, oder er ist schon besetzt.

Wenn IP_NAT_RANGE_PROTO_SPECIFIED gesetzt ist, bedeutet dies, dass der Anwender NAT, und nicht NAPT macht: Etwas Vernuenftiges mit dem Bereich tun. Wenn kein Mapping erwuenscht ist (In TCP zum Beispiel sollte kein Ziel-Mapping den TCP-Port aendern, wenn es nicht ausdruecklich verlangt wird), wird 0 geliefert.

print

Given a character buffer, a match tuple and a mask, write out the per-protocol parts and return the length of the buffer used.

print_range

Mit einem gegebenen Charakter-Buffer, einem Match-Tupel und einer Maske wird dies den Per-Protocol Teil ausdrucken und die Laenge des Buffers liefern.

4.4.3 Neue NAT-Targets

Das ist der wirklich interessante Teil. Du kannst neue NAT-Targets schreiben, die einen neuen Mapping-Typ bieten: Zwei Extra-Targets werden im Standard Package geliefert: MASQUERADE und REDIRECT. Diese illustrieren recht einfach das Potential und die Macht des Schreibens neuer NAT-Targets.

Sie werden genauso wie alle anderen iptables Targets geschrieben, nur intern werden sie 'ip_nat_setup_info()' aufrufen und die Verbindung extrahieren.

4.4.4 Protokoll-Hilfen fuer TCP und UDP

Dieser Teil befindet sich noch in der Entwicklung.

4.5 Understanding Netfilter

Netfilter ist ziemlich simpel und wurde in den vorangegangenen Sektionen recht ausfuehrlich beschrieben. Wie auch immer, manchmal ist es notwendig, weiter zu gehen als die NAT oder die ip_tables Infrastruktur bietet, oder vielleicht moechtest Du sie auch vollstaendig ersetzen.

Ein wichtiger Punkt bei Netfilter (Naja, in der Zukunft) ist Caching. Jedes skb hat ein 'nfcache' Feld: Eine Bitmaske, die sagt, welche Felder im Header untersucht wurden und ob das Paket veraendert wurde oder nicht. Die Idee ist, dass auf jeden Netfilter Hook in den dazu relevanten Bits eine OR-Verknuepfung angewandt werden kann, so dass wir spaeter ein Cache-System schreiben koennen, das clever genug sein wird, um zu erkennen, wann Pakete nicht an Netfilter gereicht werden muessen.

Die wichtigsten Bits sind NFC_ALTERED, was bedeutet, dass das Paket veraendert wurde (Dies wird fuer den NF_IP_LOCAL_OUT Hook von IPv4 bereits verwendet, um geaenderte Pakete erneut zu routen), und NFC_UNKNOWN, was aussagt, dass Caching nicht angewandt werden sollte, da einige Eigenschafte untersucht wurden, die nicht ausgedrueckt werden konnten. Im Zweifelsfall solltest Du das NFC_UNKNOWN Flag in das nfcache Feld von skb in Deinem Hook setzen.

4.6 Neue Netfilter-Module schreiben

4.6.1 Netfilter-Schnittstellen benutzen

Um Pakete im Kernel zu empfangen/behandeln, kannst Du einfach ein Modul schreiben, welches beim "Netfilter Hook" registriert. Im Grunde ist das ein Ausdruck von Interesse an einem gegebenen Punkt; der genaue Punkt ist protokollspezifisch und wird in protokollspezifischen Netfilter-Headern, wie "netfilter_ipv4.h", definiert.

Um Netfilter Hooks zu registrieren (und das wieder aufzuheben), benutzt Du die 'nf_register_hook' und die 'nf_unregister_hook' Funktionen. Beide benoetigen einen Pointer auf 'struct nc_hook_ops', welchen Du wie folgt benutzt:

list

Setz es auf '{NULL, NULL}'; verwendet, um sich der Liste zu naehern.

hook

Die Funktion, die aufgerufen wird, wenn ein Paket auf diesen Hook- Punkt trifft. Deine Funktion muss NC_ACCEPT, NC_DROP oder NC_QUEUE liefern. Wenn NC_ACCEPT geliefert wird, wird der naechste an diesen Punkt angehaengte Hook aufgerufen werden. Wenn NC_DROP geliefert wird, wird das Paket verworfen werden. Wenn NC_QUEUE geliefert wird, wird das Paket gequeued werden. Wenn Du einen Pointer auf einen skb Pointer erhaelst, kannst Du skb, wenn Du willst, vollstaendig ersetzen.

flush

Zur Zeit unbenutzt: Es wurde entworfen, um Paket-Treffer weiterzureichen, wenn der Cache geloescht wird. Vielleicht wird es niemals implementiert werden: Setz es auf NULL.

pf

Die Protokollfamilie, zum Beispiel 'PF_INET' fuer IPv4.

hooknum

Die Nummer des Hooks, fuer den Du Dich interessierst, z.B. 'NC_IP_LOCAL_OUT'.

4.6.2 Eingereihte Pakete behandeln

Dies ist die zur Zeit von 'ip_queue' verwendete Schnittstelle; Du kannst hier registrieren, um Pakete fuer ein gegebenes Protokoll zu behandeln. Hier herrscht eine aehnliche Semantik wie bei der Registrierung fuer einen Hook, ausser, dass Du eine Behandlung des Pakets blockieren kannst. Ausserdem siehst Du nur Pakete, fuer die ein Hook mit 'NC_QUEUE' geant- wortet hat.

Die zwei Funktionen, die verwendet werden, um Interesse an gequeueten Paketen zu registrieren, heissen 'nc_register_queue_handler()' und 'nf_unregister_queue_handler()'. Die Funktion, die Du registrieren wirst, wird mit dem 'void *' Pointer aufgerufen werden, mit dem Du sie an 'nf_register_queue_handler()' uebergeben hast.

Wenn niemand registriert ist, ein Paket zu behandeln, ist das Liefern von NF_QUEUE aequivalent zum Liefern von NF_DROP.

Die Pakete werden eingereiht werden, sobald Du Dich dafuer registriert hast. Du kannst mit ihnen tun, wasimmer Du willst, wenn Du aber fertig bist mit ihnen, musst Du 'nf_reinject()' aufrufen (benutz nicht einfach nur 'kfree_skb()'). Wenn Du ein skb reinjizierst, uebergibst Du ihm das skb, das dem Queue-Handler gegebene 'struct nf_info' und ein Urteil: NF_DROP verwirft es, NF_ACCEPT laesst es weiter durch die Hooks iterieren, NF_QUEUE reiht sie erneut ein und NF_REPEAT bewirkt, dass der Hook, der das Paket eingereiht hat, erneut konsultiert wird (Pass auf unendliche Loops auf!).

Du kannst in 'struct nf_info' sehen, um hilfreiche Informationen ueber das Paket zu bekommen, z.B. ueber die Schnittstelle oder den betreffenden Hook.

4.6.3 Kommandos vom Anwender empfangen

Netfilter Komponenten wollen mit dem Anwender interagieren. Die Methode hierfuer ist die Verwendung des setsockopt Mechanismus. Beachte, dass jedes Protokoll modifiziert werden muss, um nf_setsockopt() fuer nicht verstandene setsockopt Nummern (und nf_getsockopt() fuer getsockopt Nummern) aufrufen zu koennen, und das bis jetzt nur IPv4, IPv6 und DECnet modifiziert wurden.

Wir verwenden eine mittlerweile bekannte Technik: Wir registrieren ein 'struct nf_sockopt_ops', indem wir nf_register_sockopt() aufrufen. Die Felder dieser Struktur sind wie folgt:

list

Setz es auf '{NULL, NULL}'; verwendet, um sich der Liste zu naehern.

pf

Die Protokollfamilie, die Du behandelst, zum Beispiel PF_INET.

set_optmin

und

set_optmax

Diese bestimmen den (exklusiven) Bereich der verwendeten setsockopt Nummern. 0 und 0 bedeutet also, dass Du keine setsockopt Nummern benutzt.

set

This is the function called when the user calls one of your setsockopts. You should check that they have NET_ADMIN capability within this function.

get_optmin

and

get_optmax

These specify the (exclusive) range of getsockopt numbers handled. Hence using 0 and 0 means you have no getsockopt numbers.

get

Das ist die Funktion, die aufgerufen wird, wenn der User eine Deiner getsockopts aufruft. Du solltest ueberpruefen, ob es eine NET_ADMIN Capability in dieser Funktion gibt.

Die zwei letzten Felder werden intern verwendet.

4.7 Behandlung von Paketen aus Sicht der Anwender

Durch das Verwenden der libipq Library und des 'ip_queue' Moduls kann nun fast alles, was im Kernel getan werden kann, auch von Anwenderseite her geschehen. Das bedeutet, dass Du, mit etwas Geschwindigkeitsverlust, Deinen Code ausschliesslich im Userspace entwickeln kannst. Wenn Du keine grosse Bandbreite filtern musst, wirst Du das angenehmer finden, als die Pakete im Kernel zu behandeln.

In den fruehen Tagen von Netfilter habe ich das bewiesen, indem ich eine Embryo-Version von iptables zum Userspace portierte. Netfilter oeffnet den Leute die Tuer, um ihre eigenen recht effizienten Module zu schreiben, und das in welcher Sprache sie wollen.

5 2.0er und 2.2er Paketfilter-Module uebersetzen

Sieh Dir die ip_fw_compat.o Datei an; diese sollte eine Portierung recht einfach machen.

6 Die Testsuite

Auf dem CVS Server gibt es eine Test-Suite: Je groesser der Bereich ist, den diese Testsuite abdeckt, desto groesser wird Deine Zufriedenheit sein, zu sehen, dass Aenderungen am Code nichts komplett zerbrochen haben. Triviale Tests sind mindestens so wichtig, wie trickreiche Tests: Es sind die trivialen Tests, die die komplexen vereinfachen (da Du weisst, dass die Grundlagen gut funktionieren, bevor die trickreichen Tests ausgefuehrt werden).

Die Tests sind einfach: Sie sind nur Shell-Scripte im /testsuite Verzeichnis, die erfolgreich sein sollen. Die Scripte werden in alphabetischer Reihen- folge ausgefuehrt, '01test' kommt also vor '02test'. Zur Zeit gibt es fuenf Testverzeichnisse:

00netfilter/

Generelle Tests zum Netfilter-Rahmenwerk.

01iptables/

Tests zu iptables.

02conntrack/

Tests zu Connection Tracking.

03NAT/

Tests zu NAT.

04ipchains-compat/

Tests zur Kompatibilitaet von ipchains/ipfwadm.

Im testsuite/ Verzeichnis gibt es ein Script mit dem Namen 'test.sh'. Es konfiguriert zwei Dummy-Schnittstellen (tap0 und tap1), aktiviert Forwarding und beseitigt alle Netfilter-Module. Dann geht es durch die oberen Verzeichnisse und fuehrt dort alle test.sh Scripte aus, bis eins davon nicht funktioniert. Dieses Script hat zwei optionale Argumente: '-v' druckt jeden gerade ausgefuehrten Test aus. Ausserdem kann ein optionaler Testname angegeben werden: Ist dieser gegeben, werden alle anderen Tests ausgelassen, bis dieser eine gefunden wird.

6.1 Einen Test schreiben

Erstelle in dem betreffenden Verzeichnis eine neue Datei: Versuche, Deinen Test so zu numerieren, dass er zur richtigen Zeit ausgefuehrt wird. Um zum Beispiel ICMP Reply Tracking zu testen (02conntrack/02reply.sh), muessen wir zuerst sicher sein, dass ausgehende ICMP Pakete sauber getrackt werden (02conntrack/01simple.sh).

Gewoehnlich ist es besser, mehrere kleine Dateien zu erstellen, von denen jede einen Bereich abdeckt, da das hilft, Probleme fuer die Leute, die auf der Testsuite arbeiten, sofort zu isolieren.

Wenn etwas in dem Test schief laeuft, gib ein 'exit 1', das verursacht einen Fehler; wenn dies etwas ist, das Du schon erwartet hast, solltest Du eine eindeutige Meldung hinterlassen. Dein Test sollte mit 'exit 0' enden, wenn alles gut laeuft. Du solltest den Erfolg eines **jeden** einzelnen Befehls ueberpruefen, indem an den Anfang des Scripts die Option 'set -e' setzt oder indem Du ein '|| exit 1' an jeden Befehl anhaengst.

Die Hilfsfunktionen 'load_module' und 'remove_module' koennen verwendet werden, um Module zu laden: Du solltest Dich nicht auf das automatische Laden von Modulen in der Testsuite verlassen, es sei denn, das ist es spezifisch, was Du testen willst.

6.2 Variablen und Umgebung

Du hast zwei Schnittstellen zum Spielen: tap0 und tap1. Ihre Schnittstellen- adressen stehen jeweils in `$TAP0` und `$TAP1`. Beide haben die Netzmaske 255.255.255.0; ihre Netzwerke stehen jeweils in $TAP0NET und $TAP1NET.

Es gibt eine leere Temporary Datei in $TMPFILE. Sie wird am Ende Deines Tests geloescht.

Dein Script wird aus dem /testsuite Verzeichnis heraus gestartet werden, woimmer das auch sein mag. Also solltest Du Tools (wie iptables) mit Pfadnamen wie '../userspace' verwenden.

Dein Script kann mehr Informationen ausdrucken, wenn $VERBOSE gesetzt ist (was bedeutet, dass der User ein '-v' auf der Kommandozeile angegeben hat.

6.3 Nuetzliche Tools

Es gibt verschiedene nuetzliche Testsuite-Tools im Unterverzeichnis 'Tools': Jedes davon endet mit einem Nicht-Null Exit Status, wenn es ein Problem gibt.

6.3.1 gen_ip

Du kannst IP-Pakete mit 'gen_ip' generieren, was IP-Pakete an Standard- Input liefert. Du kannst IP-Pakete an tap0 und tap1 liefern, indem Du Standard-Output an /dev/tap0 oder /dev/tap1 sendest (Falls diese nicht existieren sollten, werden sie beim ersten Verwenden der Testsuite erstellt).

gen_ip ist ein einfaches Programm, das zur Zeit sehr penibel mit der Reihenfolge seiner Argumente ist. Zuerst gibt es die allgemeinen optionalen Argumente:

FRAG=offset,length

Erstelle ein Paket, zerlege es dann in Fragmente von folgendem Offset und folgender Laenge.

MF

Setz das 'More Fragments' Bit in dem Paket.

MAC=xx:xx:xx:xx:xx:xx

Setz die Quell-MAC-Adresse des Pakets.

TOS=tos

Setz das TOS-Feld im Paket (0 - 255).

Als naechstes kommen die erforderlichen Argumente:

source ip

Quell-IP-Adresse des Pakets.

dest ip

Ziel-IP-Adresse des Pakets.

length

Gesamtlaenge des Pakets, einschliesslich Header.

protocol

Protokollnummer des Pakets, 17 ist zum Beispiel UDP.

Die Argumente haengen dann vom Protokoll ab: fuer UDP (17) gibt es Quell- und Ziel-Portnummer. Fuer ICMP (1) gibt es Typ und Code der ICMP Meldung: ist der Typ 0 oder 8 (ping-Antwort oder ping), werden zwei zusaetzliche Argumente (das ID und das Sequenzfeld) benoetigt. Fuer TCP werden Quell- und Zielport, sowie Flags ("SYN", "SYN/ACK", "ACK", "RST" or "FIN") benoetigt. Hier gibt es drei optionale Argumente: "OPT=", gefolgt von einer durch Kommata getrennten Liste von Optionen, "SYN=" gefolgt von einer Sequenznummer, und "ACK=" gefolgt von einer Sequenznummer. Schliesslich sagt das optionale Argument "DATA", dass der Payload des TCP-Pakets mit den Inhalt von Standard-Input gefuellt werden soll.

6.3.2 rcv_ip

Du kannst Dir IP-Pakete mit rcv_ip ansehen, was eine Kommandozeile ausgibt, die so nah wie moeglich an die Originalwerte herankommt, die an gen_ip uebergeben wurden (mit Ausnahme von Fragmenten).

Das ist extrem nuetzlich fuer das Analysieren von Paketen. Es erfordert zwei Argumente:

wait time

Die maximale Zeit in Sekunden, um auf ein Paket von Standard Input zu warten.

iterations

Die Anzahl der zu empfangenden Pakete.

Es gibt ein optionales Argument "DATA", was den Payload des Pakets nach dem Paketheader auf Standard-Output ausdruckt.

Der Standardweg, rcv_ip in einem Shellscript zu verwenden, ist wie folgt:

```
# Jobkontrolle aufsetzen, damit mir $ in Shellscripten verwenden koennen.
set -m

# Zwei Sekunden auf ein Paket von tap0 warten.
../tools/rcv_ip 2 1 < /dev/tap0 > $TMPFILE &

# Sichergehen, dass rvp_ip laeuft.
sleep 1

# Ein ping-Paket senden
../tools/gen_ip $TAP1NET.2 $TAP0NET.2 100 1 8 0 55 57 > /dev/tap1 || exit 1

# Auf rvp_ip warten
if wait %../tools/rcv_ip; then :
else
    echo rcv_ip failed:
    cat $TMPFILE
    exit 1
fi
```

6.3.3 gen_err

Dieses Programm nimmt ein Paket (wie zum Beispiel von ip_gen generiert) und wandelt es in einen ICMP Fehler um.

Es hat drei Argumente: Die Quell-IP-Adresse, einen Typ und einen Code. Die Ziel-IP-Adresse wird auf die Quell-IP-Adresse des von Standard- Input uebergebenen Pakets gesetzt.

6.3.4 local_ip

Dies nimmt ein Paket von Standard-Input und injiziert es von einem einfachen Socket ins System. Dies gibt ihm den Anschein eines lokal-generierten Pakets (anders, als ein Paket an eines der Ethertap Devices zu uebergeben, was wie ein Paket aussehen wuerde, das woanders generiert wurde).

6.4 Einige Ratschlaege

All diese Tools nehmen an, dass sie alles mit einem Lesen und Schreiben tun koennen: Dies ist wahr fuer die Ethertap Devices, koennte aber nicht wahr sein, wenn Du etwas Trickreiches mit Pipes machst.

dd can verwendet werden, um Pakete auszuschneiden: dd hat eine obs (output block size) Option, die verwendet werden kann, um Pakete mit einem einzigen Schreiben auszugeben.

Teste zuerst auf Erfolg: z.B. ein Test darauf, ob Pakete erfolgreich geblockt werden. Teste zuerst, ob Pakete normal durchgehen, teste **anschliessend**, ob Pakete geblockt werden. Andernfalls koennte ein Fehler, der nichts damit zu tun hat, die Pakete aufhalten...

Versuche, exakte Tests zu schreiben, keine 'Schreib zufaelliges Zeug und warte ab, was passiert' Tests. Wenn ein exakter Tests nicht funktioniert, ist es sehr nuetzlich, das zu wissen. Wenn ein zufaelliger Test nicht funktioniert, hilft das nicht viel.

Wenn ein Tests ohne ein Meldung nicht gelingt, kannst Du ein '-x' (ich meine '#! /bin/sh -x') in die erste Zeile des Scripts setzen, um zu sehen, welches Kommando gerade ausgefuehrt wird.

Wenn ein Test mal gelingt, mal nicht, ueberpruefe zufaelligen Netzwerkverkehr, der dazwischenkommen koennte (versuche, alle externen Schnittstellen zu deaktivieren). Da ich an das gleiche Netzwerk wie Andrew Trigell angeschlossen bin, werde ich zum Beispiel staendig von Windows Broadcasts geplagt.

7 Motivation

Als ich ipchains entwickelte, merkte ich (in einem dieser Flash-Momente- waehrend-man-auf-Einlass-wartet in einem chinesischen Restaurant), dass Paketfiltern an der falschen Stelle ansetzte. Ich kann es jetzt nicht mehr finden, aber ich erinnere mich daran, dass ich eine Mail an Alan Cox schrieb, der mir antwortete 'Warum beendest Du nicht erst das, was Du im Moment tust, obwohl Du wahrscheinlich recht hast'. Um es kurz zu sagen, sollte Pragmatismus ueber die Richtige Sache siegen.

Als ich ipchains, was urspruengliche eine kleine Modifikation des Kernelteils von ipfwadm war, und dann doch eine groessere Neu-Ueberarbei- tung wurde, beendete und das HOWTO schrieb, bemerkte ich die Verwirrung, die in der grossen Linux-Community ueber Dinge wie Paketfiltern, Masque- rading, Portforwarding und aehnliches herrscht.

Das ist das Gute daran, wenn Du Dein eigener Support bist: Du bekommst ein besseres Gefuehl dafuer, was die User versuchen zu tun, und womit sie noch straucheln. Es ist die groesste Auszeichnung fuer Freie Software, wenn sie von vielen Usern eingesetzt wird (darum geht es, richtig?), und das bedeutet, dass man es leicht machen muss. Die Architektur, und nicht die Dokumenation, ist der Schluessel hierzu.

Ich hatte also die Erfahrung, sowohl mit dem ipchains-Code, als auch mit einer guten Vorstellung davon, was die Leute dort draussen taten. Es gab nur zwei Probleme:

Zuerst wollte ich nicht in den Security-Bereich zurueckgehen. Ein Security-Consultant zu sein ist ein konstantes moralischen Tauziehen zwischen Deinem Gewissen und Deinem Konto. Auf einem fundamentalen Level verkaufst Du das Gefuehl von Sicherheit, was alles andere ist als wirkliche Sicherheit. Vielleicht waere es anders, in einer Militaereinrichtung zu arbeiten, wo man Sicherheit versteht.

Das zweite Problem ist, dass es nicht nur um Newbies geht; eine wachsende Anzahl von grossen Unternehmen und ISPs benutzen dieses Zeug. Ich brauchte zuverlaessige Informationen von dieser Klasse von Benutzern, wenn es sich auf die Home-User von Morgen ausbreiten sollte.

Diese Probleme wurden geloest, als ich auf der Usenix 1998 auf David Bonn von Watchguard traf. Sie suchten einen Linux-Kernel Programmierer; Im Ende einigten wir uns darauf, dass ich fuer einen Monat zu ihrer Niederlassung in Seattle kommen wuerde, wo wir sehen wuerde, ob wir eine Vereinbarung rausschlagen koennten, in der sie meinen neuen Code und meine jetzige Support-Arbeit sponsorn wuerden. Das Gehalt, das sie mir anboten, war mehr als das, nachdem ich gefragt hatte, ich hatte also keine Einnahmeverluste. Das bedeutet, dass ich mir fuer eine Weile keine Sorgen um Geld zu machen brauche.

Die Naehe zu Watchguard gab mir die Naehe zu den grossen Clients, die ich brauchte, und dass ich unabhaenging von ihnen war, erlaubte mir, alle User (z.b. Watchguard Competitors) gleich zu supporten.

Ich haette also einfach Netfilter schreiben und ipchains portieren koennen und waere fertig damit gewesen. Leider haette das den ganzen Masquerading- Code im Kernel gelassen: Die Unabhaengigkeit zwischen Paketfiltern und Masquerading ist ein wichtiger Punkt beim Paketfiltern, aber Masquerading muss auch auf das Netfilter-Rahmenwerk uebertragen werden.

Meine Erfahrung mit dem ipfwadm 'Schnittstellen-Adressierungs' Feature hat mich auch gelehrt, dass es keine Hoffnung dafuer gab, den Masquerading Code einfach auszulassen und zu erwarten, dass ihn jemand brauchte und somit selbst die Arbeit einer Portierung zu Netfilter uebernehmen wuerde.

Ich brauchte also mindestens soviele Features wie im damaligen Code; vorzugsweise ein paar mehr, um mutige und kreative User zu ermutigen, es mir nachzumachen. Dies bedeutete, transparente Proxies (zum Glueck!), Masquerading und Port-Forwarding zu ersetzen. Mit anderen Worten, ein komplettes NAT-Layer.

Sogar, wenn ich mich dazu entschieden haette, das existierenden Masquerading Layer zu Portieren, anstatt ein generisches NAT-System zu schreiben, zeigte der Masquerading Code doch sein Alter und den Mangel an Wartung. Es gab keinen Masquerading Maintainer, und das zeigte sich jetzt. Es scheint, dass ernsthafte User Masquerading generell nicht verwenden und dass es nicht viele Home-User gibt, die die Arbeit der Wartung uebernehmen wuerden. Tapfere Leute wie Juan Ciarlante schrieben Fixes, aber der Punkt (der immer weiter nach hinten verschoben wurde) war erreicht, an dem eine komplette Ueberarbeitung noetig war.

Beachte bitte, das nicht ich die Person war, die NAT ueberarbeitet hat: Ich verwendete Masquerading nicht mehr und hatte den existierenden Code zu der Zeit nicht studiert. Das ist wahrscheinlich der Grund, warum es laenger dauerte, als es dauern sollte. Aber das Resultat ist ziemlich gut, meiner Meinung nach, und ich habe sicher verdammt viel gelernt. Ohne Zweifel wird die zweite Version noch besser werden, wenn wir sehen, wie die Leute es einsetzen.

8 Danke

Danke denen, die geholfen haben.