Pengantar

Penapisan Paket

Rahmatul Irfan

irfan@mti.gadjahmada.edu

Pengantar Penapisan Paket

Rahmatul Irfan
irfan@mti.gadjahmada.edu

PENAPISAN PAKET

Penapisan paket adalah mekanisme sekuritas jaringan yang bekerja dengan mengontrol data apa saja yang bisa keluar masuk jaringan. Untuk memahami penapisan paket, yang perlu dipelajari terlebih dahulu konsep jaringan IP tingkat-tinggi.

Dalam mengirim informasi melewati jaringan, informasi harus dipecahkan menjadi beberapa bagian kecil, masing-masing dikirim secara terpisah. Pemecahan informasi menjadi beberapa potongan membolehkan beberapa sistem membagi jaringannya, masing-masing mengirimkan bagian-bagiannya secara bergiliran. Dalam jaringan IP, bagian-bagian kecil data ini disebut sebagai paket. Semua data yang dikirim melewati jaringan IP berbentuk paket.

Peralatan dasar yang saling menghubungkan jaringan IP disebut dengan *router*. *Router* bisa berupa potongan perangkat-keras yang hanya mempunyai satu tujuan dan tidak mempunyai tujuan yang lain, atau bisa berupa potongan perangkat-lunak yang bekerja pada sistem UNIX atau PC yang memiliki tujuan umum (MS-DOS, *Windows*, *Macintosh*, atau yang lain). Paket yang melintasi interjaringan (jaringan antar jaringan-jaringan) berjalan dari *router* ke *router* sampai mereka mencapai tujuannya. Internet sendiri merupakan bentuk moyang interjaringan, "jaringan atas jaringan-jaringan" yang terakhir.

Router harus membuat keputusan *routing* tentang setiap paket yang diterimanya, ia harus memutuskan bagaimana cara mengirimkan paket menuju tujuannya yang terakhir. Secara umum, paket tidak membawa informasi yang bisa membantu *router* dalam keputusan ini, selain dari alamat IP tujuan akhir paket. Paket memberitahu arah tujuannya kepada *router*, tetapi bukan bagaimana cara untuk sampai disana. *Router* melakukan komunikasi satu sama lain dengan menggunakan protokol *routing* seperti *Routing Information Protocol* (RIP) dan *Open Shortest Path First* (OSPF) untuk membangun tabel *routing* dalam memori guna menentukan

bagaimana membawa paket sampai pada tujuannya. Ketika suatu paket di-*routing*, *router* membandingkan alamat tujuan paket dengan *entry* dalam tabel *routing* dan mengirimkan paket ke depan seperti ditunjukkan oleh tabel *routing*. Seringkali tidak terdapat *route* khusus untuk tujuan tertentu, dan *router* akan menggunakan "*default route*". Biasanya *route* semacam ini menunjukkan paket pada *router* yang lebih cepat atau *router* yang mempunyai hubungan yang lebih baik (sebagian besar *dafault router* merujuk pada Internet).

Dalam menentukan cara pengiriman suatu paket ke tujuannya, *router* normal hanya terlihat pada alamat tujuan paket normal dan hanya menanyakan "Bagaimana saya dapat mengirim paket ini?" *Router* penapisan paket juga mempertimbangkan pertanyaan "Apakah saya akan mengirim paket ini?" *Router* penapisan paket menjawab pertanyaan ini sesuai dengan kebijakan sekuritas yang telah diprogramkan ke dalam *router* melalui aturan-aturan penapisan paket.

Beberapa paket yang biasanya tidak berisi *routing* informasi tentang bagaimana cara mereka mencapai tujuannya, menggunakan IP *option* "*source route*". Paket-paket ini biasanya disebut dengan paket *source route* atau IP *option*.

1. Keunggulan dan Kelemahan Penapisan Paket

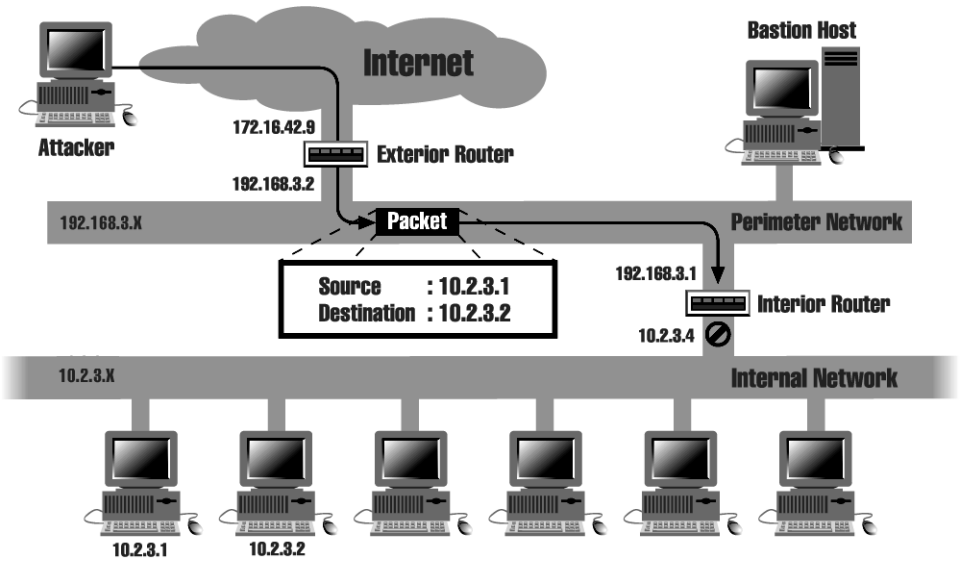
Penapisan paket membantu mengontrol (boleh atau tidak boleh) pengiriman data berdasarkan:

- alamat data yang (menurut dugaan) datang dari,
- alamat data menuju ke, dan
- protokol aplikasi dan *session* yang digunakan untuk mengirim data.

Hampir sebagian besar sistem penapisan paket tidak melakukan sesuatu apapun berdasarkan data itu sendiri, mereka tidak membuat keputusan berdasarkan isi. Penapisan paket akan menyatakan: "jangan biarkan siapapun menggunakan Telnet (protokol aplikasi) untuk melakukan *log in* dari luar", atau "biarkan siapapun mengirim *email* kepada *user* perusahaan lewat SMTP (protokol aplikasi lain)", atau

bahkan menyatakan "mesin itu dapat mengirim berita kepada *user* perusahaan melalui NNTP (protokol aplikasi lain), tetapi tidak ada mesin lain yang dapat berbuat serupa". Namun, penapisan paket tidak akan menyatakan: "*user* ini dapat menggunakan Telnet *in* dari luar, tetapi tidak ada *user* lain yang dapat melakukan langkah serupa", karena *user* bukan sesuatu yang dapat diidentifikasi oleh sistem penapisan paket. Ia tidak akan menyatakan: "anda bisa mengirim *file-file* tersebut, tetapi bukan *file-file* itu", karena *file* bukan sesuatu yang dapat dikenali olehnya.

Keuntungan utama penapisan paket adalah *leverage*, yaitu ia mengijinkan pemberian perlindungan khusus terhadap jaringan secara keseluruhan dalam satu tempat. Perhatikan layanan Telnet sebagai contoh. Jika Telnet tidak diijinkan untuk mematikan *server* Telnet pada semua *host*, maka yang harus dikhawatirkan adalah seseorang dalam perusahaan yang melakukan *install* mesin baru (atau *reinstall* mesin lama) dengan menghidupkan *server* Telnet. Sebaliknya, jika Telnet tidak diijinkan oleh *router* penapisan, maka mesin baru ini akan diproteksi segera sejak *start*, baik *server* Telnet sedang diaktifkan atau tidak. Ini merupakan contoh *fail safe stance*.



Gambar 1 Pemalsuan alamat sumber.

Ada perlindungan tertentu yang hanya dapat diberikan dengan *router* penapisan, dan hanya dapat diberikan jika mereka tersebar dalam lokasi khusus di dalam jaringan. Sebagai contoh, merupakan ide yang bagus apabila semua paket yang mempunyai alamat sumber internal ditolak, yaitu paket-paket yang mengklaim berasal dari mesin internal tetapi sesungguhnya berasal dari luar, karena paket-paket semacam itu biasanya merupakan bagian serangan *address-spoofing*. Dalam serangan ini, penyerang dianggap berasal dari mesin internal. Pengambilan keputusan untuk bentuk ini hanya dapat dilakukan dalam penapisan *router* pada perimeter jaringan. Hanya penapisan *router* dalam lokasi ini saja (yang sudah ditentukan batasan antara "bagian dalam" dan "bagian luar") yang dapat mengenali paket tersebut, dengan melihat pada alamat sumber yang sama dan mengenali apakah paket berasal dari dalam (sambungan jaringan internal) atau berasal dari luar (sambungan jaringan eksternal). Gambar 1 mengilustrasikan tipe pemalsuan alamat sumber.

1.1 Keunggulan Penapisan Paket

Penapisan paket mempunyai sejumlah keunggulan yang membawa manfaat dan keuntungan berikut.

Satu *screening router* dapat membantu melindungi seluruh jaringan

Salah satu keunggulan penapisan paket adalah bahwa *router* penapisan paket tunggal yang ditempatkan secara strategis dapat membantu melindungi seluruh jaringan. Jika hanya ada satu *router* yang menyambungkan situs dengan Internet, *leverage* pada sekuritas jaringan bisa didapatkan tanpa menghiraukan ukuran situs yang ada, apabila mempekerjakan penapisan paket pada *router* itu.

Penapisan paket tidak memerlukan pengetahuan atau kerjasama *user*

Penapisan paket tidak memerlukan perangkat-lunak atau konfigurasi apapun dari mesin *client*, maupun pelatihan atau prosedur khusus bagi *user*. Ketika *router* penapisan paket memutuskan untuk membiarkan paket lewat, *router* ini tidak dapat

dibedakan dari *router* normal. Idealnya, *user* tidak akan mengetahui keberadaannya, kecuali jika mereka berusaha untuk melakukan sesuatu yang dilarang (yang sudah diperkirakan dalam masalah sekuritas) oleh kebijakan penapisan pada *router* penapisan paket.

Hal ini berarti bahwa penapisan paket dapat dilakukan tanpa kerjasama, dan seringkali tanpa pengetahuan *user*. Persoalannya bukan karena tidak dapat dilakukan secara subversif dibelakang punggung *user* (namun tindakan seperti ini kadangkala diperlukan juga, tergantung pada keadaan). Persoalannya adalah penapisan paket dapat dilakukan tanpa harus *user* mempelajari suatu hal yang baru untuk membuatnya bekerja, dan tanpa harus tergantung pada *user* untuk berbuat (atau tidak berbuat) sesuatu untuk membuatnya bekerja.

Penapisan paket tersedia secara luas di banyak *router*

Kemampuan penapisan paket tersedia pada banyak produk *routing* perangkat-keras dan perangkat-lunak, baik yang bersifat komersial maupun yang tersedia bebas di Internet. Sebagian besar situs telah mempunyai kemampuan penapisan paket yang tersedia dalam *router* yang mereka gunakan.

Sebagian besar produk *router* komersial, seperti *router* dari *Livingston Enterprises and Cisco System*, memasukkan kemampuan penapisan paket. Kemampuan penapisan paket juga tersedia di dalam sejumlah paket yang lain, seperti *Drawbridge*, *KarlBridge*, dan *screend,* yang didistribusikan bebas pada Internet.

Catatan: Tidak mungkin dalam pembahasan ini bisa diberikan daftar paket lengkap yang tersedia secara umum dan komersial, karena produk baru selalu diperkenalkan dan kemampuan penapisan paket selalu ditambahkan pada produk yang ada, (bisa dilihat pada lampiran). Pembahasan ini hanya memusatkan perhatian pada ciri-ciri dan kemampuan penapisan paket generik, dan konsekwensi mempunyai kemampuan khusus atau tidak. Dengan demikian diharapkan pembahasan ini dapat dijadikan satu pegangan dan pertimbangan dalam menetukan produk yang tersedia di pasaran.

1.2 Kelemahan Penapisan Paket

Meskipun penapisan paket memberikan banyak keunggulan yang membawa keuntungan, terdapat juga kelemahan yang bisa membawa kerugian dalam menggunakan penapisan paket.

Beberapa protokol tidak begitu sesuai dengan penapisan paket

Bahkan dengan implementasi penapisan paket yang sempurna sekalipun, masih bisa ditemukan bahwa sebagian protokol tidak begitu sesuai dengan sekuritas melalui penapisan paket, dengan alasan yang akan dibahas selanjutnya dalam pembahasan ini. Misalnya protokol yang memasukkan perintah "Berkeley r" (*rcp, rlogin rdist, rsh,* dst) dan protokol yang berdasarkan RPC seperti NFS dan NIS/YP.

Beberapa kebijakan tidak dapat dilaksanakan dengan *router* penapisan paket normal

Informasi bahwa *router* penapisan paket telah tersedia tidak mengijinkan penentuan beberapa aturan yang diinginkan sendiri. Sebagai contoh, paket menyatakan dari *host* apa mereka didatangkan, bukan menanyakan *user* apa. Oleh karena itu, tidak dapat diadakan suatu pembatasan terhadap *user* khusus. Demikian halnya, paket menyatakan *port* apa yang akan mereka tuju, tetapi tidak mengatakan apa aplikasinya. Pada saat batasan-batasan dipaksakan pada protokol tingkat tinggi, maka hal itu akan dilakukan dengan bilangan *port*, dengan mengaharapkan bahwa tidak ada sesuatu yang dijalankan pada *port* yang menunjukkan protokol itu. *Malicious insiders* dapat dapat dengan mudah mengalahkan bentuk kontrol ini.

Beberapa kelemahan lain

Meskipun penapisan paket tersedia luas dalam berbagai macam paket perangkat-keras dan perangkat-lunak, penapisan paket bukanlah alat yang sempurna. Kemampuan penapisan paket pada kebanyakan produk tersebut mempunyai batasan-batasan umum berikut.

- Penapisan paket cukup rentan terhadap serangan yang diarahkan ke protokol yang lebih tinggi daripada protokol level jaringan, padahal level itulah yang satu-satunya dimengerti.
- Penapisan paket tidak menyembunyikan topologi jaringan pribadi dan sering membeberkan jaringan pribadi ke dunia luar.
- Penapisan paket mempunyai kemampuan *auditing* yang sangat terbatas, padahal *auditing* memainkan peranan yang penting dalam kebijakan sekuritas perusahaan. Protokol level jaringan memerlukan pengetahuan tertentu tentang rincian teknis, dan tidak semua administrator mempunyai pengetahuan tersebut, karena penapisan paket *firewall* mempunyai konfigurasi dan pengujian yang rumit mengenai peningkatan resiko kesalahan konfigurasi sistem, lubang-lubang sekuritas, dan kegagalan.
- Beberapa produk penapisan paket mempunyai kemampuan yang tidak lengkap, yang membuat implementasi tipe-tipe penapisan tertentu menjadi sulit dan mustahil.
- Seperti halnya yang lain, di antara paket-paket penapisan paket bisa ditemukan kesalahan, dan kesalahan ini lebih mungkin daripada kesalahan *proxying* yang dihasilkan dalam persoalan-persoalan sekuritas. Biasanya, *proxy* yang gagal menghentikan data yang lewat, sementara implementasi penapisan paket yang keliru dapat mengijinkan paket yang seharusnya ditolak.
- Tidak semua aplikasi Internet didukung oleh penapisan paket.

2. Membuat Konfigurasi *Router* Penapisan Paket

Untuk mengkonfigurasikan *router* penapisan paket, pertama kali yang harus diputuskan adalah layanan apa saja yang akan diijinkan dan yang ditolak, dan selanjutnya menerjemahkan keputusan ini ke dalam aturan-aturan paket. Dalam realitasnya, perincian paket mungkin sama sekali tidak diperhatikan. Karena yang penting adalah pekerjaan selesai dilaksanakan. Sebagai contoh, *user* hanya ingin

menerima *mail* dari Internet, sedangkan yang dilaksanakan paket tidak relevan dengan hal itu. Di sisi yang lain, *router* hanya peduli dengan paket-paket dan bagian-bagian yang sangat terbatas darinya. Dalam menyusun aturan-aturan untuk *router* perusahaan, pernyataan "menerima *mail* dari Internet" harus diterjemahkan ke dalam suatu deskripsi tentang bentuk-bentuk paket khusus yang diijinkan lewat *router*.

Bagian berikut ini menggarisbawahi konsep umum yang perlu diingat ketika menerjemahkan keputusan-keputusan layanan ke dalam aturan-aturan paket (rincian tertentu untuk setiap layanan dapat dilihat pada lampiran).

2.1 Protokol *Birectional*

Protokol-protokol pada umumnya merupakan *birectional*; mereka hampir selalu melibatkan satu sisi untuk mengirimkan suatu penyelidikan atau perintah, dan satu sisi lain untuk mengirim tanggapan terhadap beberapa bentuk. Pada saat aturan-aturan penapisan paket direncanakan, yang perlu diingat adalah bahwa paket-paket berjalan pada dua arah. Sebagai contoh, ia tidak akan melakukan kebaikan apapun yang mengijinkan paket *outbound* Telnet membawa *keystroke* ke *host* yang jauh, apabila paket-paket yang masuk tidak diijinkan untuk sambungan yang membawa kembali layar *display*.

Sebaliknya, ia juga tidak menginginkan *user* melakukan kebaikan apapun untuk memblok hanya separuh sambungan. Banyak serangan dapat dilakukan jika penyerang dapat membawa paket masuk ke dalam jaringan, sekalipun penyerang tidak dapat mengirim kembali respons apapun. Hal ini mungkin terjadi, karena respons cukup dapat diprediksikan untuk mengijinkan penyerang melihat pembicaraan tanpa harus benar-benar melihat keseluruhan respons. Jika respons dapat diprediksikan, maka penyerang tidak perlu melihat mereka. Penyerang tidak akan mampu menyaring informasi apapun secara langsung jika mereka tidak melihat respons, tetapi mereka akan mampu melakukan sesuatu yang memberinya data secara tidak langsung.

2.2 *Inbound* versus *Outbound*

Pada saat strategi penapisan paket direncanakan, perlu pembahasan tentang '*inbound*' versus '*outbound*' yang teliti. Paket-paket dan layanan-layanan *inbound* dan *outbound* harus dibedakan dengan hati-hati. Layanan *outbound* (misalnya layanan Telnet) meliputi paket-paket *outbound* (*keystroke*) dan paket-paket *inbound* (respons yang akan ditunjukkan pada layar). Ketika berbicara dengan orang lain mengenai penapisan, maka komunikasi harus jelas apakah yang dibicarakan adalah paket-paket *inbound* versus *outbound*, atau layanan-layanan *inbound* versus *outbound*.

2.3 *Default Permit* versus *Default Deny*

Dalam pembahasan strategi sekuritas, ada pembedaan antara kedua *stance* yang dapat dipilih dalam meletakkan kebijakan sekuritas; *default deny stance* (yang tidak diperbolehkan atau dilarang secara jelas) dan *default permit stance* (yang tidak dilarang dan diijinkan secara eksplisit). Dari sudut pandang sekuritas, jauh lebih selamat jika mengambil sikap bahwa segala sesuatu seharusnya ditolak oleh *default*. Aturan-aturan penapisan paket seharusnya mencerminkan *stance* ini, yaitu dimulai dari posisi menolak segala sesuatu dan selanjutnya menyusun aturan-aturan yang hanya mengijinkan protokol-protokol yang diinginkan saja dan yang menurut implikasi sekuritas, serta yang dianggap cukup aman (sesuai dengan definisi khusus mengenai "cukup aman").

Default deny stance lebih aman dan lebih efektif dibandingkan dengan *default permit stance*, yang mengakui dan memberi ijin segala sesuatu dengan *default* dan berusaha untuk memblok segala sesuatu yang dianggap sebagai permasalahan. Kenyataan menunjukkan bahwa dengan pendekatan semacam ini, tidak semua permasalahan dapat diketahui, dan dengan demikian tidak akan ada kemampuan untuk melakukan semua pekerjaan.

Default deny stance mempunyai arti bahwa aturan penapisan seharusnya berupa daftar kecil tentang hal-hal khusus yang diijinkan, diharapkan dengan adanya

beberapa hal khusus yang ditolak seluruhnya akan memunculkan logika yang benar, diikuti dengan *default deny* yang menutupi segala sesuatu yang lain.

3. Penapisan Paket pada Protokol TCP/IP

Untuk memahami penapisan paket, pertama kali yang harus dilakukan adalah memahami paket-paket tersebut dan bagaimana mereka ditangani pada setiap lapisan pada *stack* protokol TCP/IP:

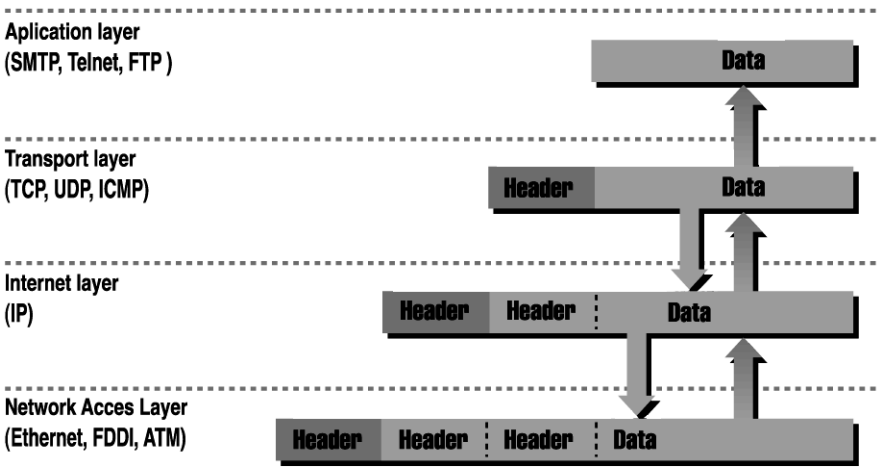
- lapisan Aplikasi (misalnya FTP, Telnet, HTTP),
- lapisan *Transport* (TCP atau UDP),
- lapisan Internet (IP), dan
- lapisanan Akses Jaringan (misalnya Rthernet, FDDI, ATM).

Paket-paket disusun dengan cara sedemikian rupa sehinga lapisan setiap protokol yang digunakan untuk sambungan khusus dibungkus mengelilingi paket-paket, seperti lapisan kulit pada bawang.

Pada setiap lapisan, sebuah paket mempunyai dua bagian: *header* dan *body*. *Header* mengandung informasi protokol yang relevan dengan lapisan itu, sedangkan *body* mengandung data untuk lapisan itu yang seringkali berisi seluruh paket dari lapisan berikutnya dalam *stack*. Setiap lapisan memberikan informasi yang diperolehnya dari lapisan di atasnya sebagai data, dan menggunakan *header*-nya sendiri untuk data ini. Pada setiap lapisan, paket memuat semua informasi yang disampaikan dari lapisan yang lebih tinggi, dan tidak satupun yang terlewatkan. Proses melindungi data pada saat menggunakan *header* baru dikenal sebagai *encapsulation.*

Pada lapisan aplikasi, paket berisi data yang akan dikirim (misalnya, bagian *file* yang akan dikirim selama FTP *session*). Karena menuju ke lapisan *transport*, maka *Transmission Control Protocol* (TCP) atau *User Datagram Protocol* (UDP) menjaga data dari lapisan sebelumnya dan mengenakan *header* padanya. Pada lapisan selanjutnya, IP menganggap seluruh paket (sekarang berisi *header* TCP atau UDP

dan data) sebagai data, dan sekarang menggunakan *header* IP-nya sendiri. Terakhir, pada lapisan akses jaringan, Ethernet atau protokol jaringan lain menganggap seluruh paket IP yang disampaikan padanya sebagi data, dan menggunakan *header*-nya sendiri. Gambar 2 menunjukkan bagaimana mereka bekerja.



Gambar 2 Enkapsulasi data.

Pada sisi lain dari sambungan tersebut, proses ini dikembalikan. Karena data disampaikan dari satu lapisan ke lapisan selanjutnya yang lebih tinggi, maka setiap *header* (setiap kulit bawang) dilepas melalui lapisannya masing-masing. Sebagai contoh, lapisan Internet memindahkan IP *header* sebelum melewati data yang dienkapsulasi sampai lapisan *transport* (TCP atau UDP).

Dalam usaha memahami penapisan paket, informasi yang paling penting dari sudut pandang *user* adalah *header* dari berbagai macam lapisan. Bagian di bawah ini melihat pada beberapa contoh paket dengan tipe yang berbeda dan memperlihatkan isi setiap *header* yang akan diuji oleh *router* penapisan paket. Pengenalan yang rinci mengenai TCP/IP tidak dijelaskan di sini, yang dijelaskan hanya beberapa pengetahuan tertentu mengenai fundamental TCP/IP, dan memusatkan perhatian pada pembahasan persoalan-persoalan khusus yang berhubungan dengan penapisan paket.

Dimulai dengan pembahasan yang menggambarkan TCP/IP pada Ethernet. Dilanjutkan dengan pembahasan karakteristik-karakteritik penapisan paket, protokol-protokol diatas IP (seperti TCP, UDP, ICMP, dan RPC), protokol-protokol di bawah IP (seperti Ethernet), dan terakhir mengenai protokol-protokol non-IP (seperti *AppleTalk* atau IPX).

3.1 TCP/IP/Ethernet

Perhatikan contoh paket TCP/IP (sebagai contoh paket yang merupakan bagian dari sambungan Telnet) pada Ethernet. Terdapat empat lapisan yang menarik disini, yaitu lapisan Ethernet, lapisan IP, lapisan TCP, dan lapisan data. Pada bagian ini, pembahasan dimulai dari bawah ke atas dan melihat isi *header* yang akan diuji *router* penapisan paket.

Lapisan Ethernet

Lapisan Ethernet merupakan paket yang terdiri atas dua bagian: Ethernet *header* dan Ethernet *body*. Secara umum, penapisan paket tidak akan dilakukan berdasarkan informasi dalam Ethernet *header*. Pada dasarnya *header* akan memberitahukan:

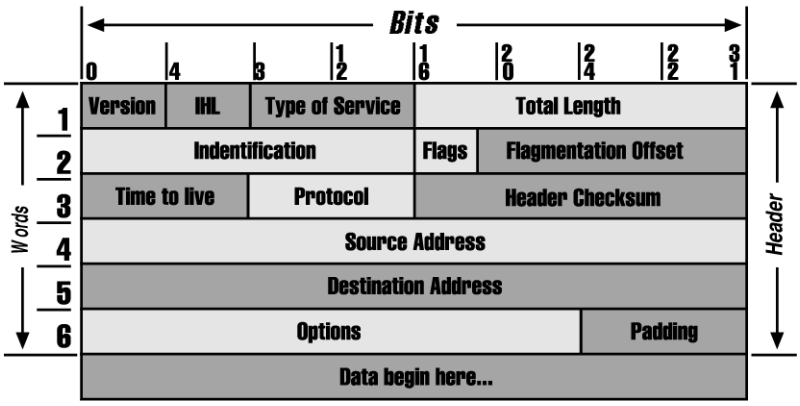
- jenis paket, yang dimaksud dalam contoh ini adalah paket IP, yang dibedakan dengan paket *AppleTalk*, paket *Novell*, paket *DECNET*, atau bentuk-bentuk paket yang lain;
- alamat Ethernet pada mesin yang meletakkan paket ke dalam golongan jaringan Ethernet khusus, merupakan mesin dengan sumber awal; dengan kata lain, *router* terakhir pada *path* berasal dari mesin sumber ke sini;
- alamat Ethernet untuk tujuan paket pada golongan jaringan Ethernet khusus, kemungkinan merupakan mesin tujuan jika ia dipasang pada golongan ini; dengan kata lain, *router* selanjutnya pada *path* dari sini menuju ke mesin tujuan.

Yang dimaksud dalam contoh ini adalah paket IP, Ethernet *body* memuat paket IP.

Lapisan IP

Pada lapisan IP, paket IP disusun menjadi dua bagian, yaitu IP *header* dan IP *body*, seperti ditunjukkan dalam Gambar 3. Dari sudut pandang penapisan paket, IP *header* memuat empat potongan informasi yang menarik berikut.

- Alamat sumber IP, panjang empat *bite* dan ditulis secara khusus seperti 172.16.244.34.
- Alamat tujuan IP, sama seperti alamat sumber.
- Tipe protokol IP, mengidentifikasi IP *body* sebagai paket TCP, seperti dibedakan dengan paket UDP, paket *Internet Control Message Protokol* (ICMP), atau beberapa tipe paket lain.
- IP *option field*, yang hampir selalu kosong, tetapi jika ada pilihan seperti *route* sumber IP dan pilihan sekuritas IP akan dikhususkan apabila mereka digunakan untuk paket tertentu. (Lihat pembahasan mengenai "IP *option*").



Gambar 3 IP *header* dan IP *body*.

IP dapat membagi paket yang terlalu besar untuk mempertemukan jaringan tertentu ke dalam rangkaian paket yang lebih kecil yang disebut dengan fragmen. Fragmentasi paket tidak mengubah strukturnya pada lapisan IP (IP *header* disalin kedalam setiap fragmen), tetapi ini bisa berarti bahwa *body* hanya memuat sebagian paket saja pada lapisan selanjutnya. (Lihat pembahasan mengenai "fragmentasi IP").

IP *body* dalam contoh ini memuat paket TCP yang tidak difragmentasikan, meskipun ia memuat fragmen pertama dari paket TCP yang difragmentasikan.

Lapisan TCP

Pada lapisan TCP, paket memuat dua bagian, yaitu TCP *header* dan TCP *body*. Dari sudut pandang penapisan paket, TCP *header* memuat tiga potongan informasi yang menarik, yaitu:

- *port* sumber TCP, bilangan dua *bite* yang menentukan *server* atau *client* apa yang memproses paket yang berasal dari mesin sumber,
- *port* tujuan TCP, seperti halnya *port* sumber TCP, dan
- TCP *field flag*.

TCP *field flag* memuat satu bit kepentingan untuk penapisan paket, yaitu bit ACK. Dengan menguji bit ACK, *router* penapisan paket dapat menentukan apakah paket tertentu merupakan paket pertama yang memulai sambungan TCP (jika bit ACK tidak disusun) atau paket selanjutnya (jika bit ACK disusun). Bit ACK merupakan bagian mekanisme TCP yang menjamin pengiriman data, yang disusun ketika satu sisi suatu sambungan telah menerima data dari sisi sambungan lain (jika data yang diterima diketahui). Oleh karena itu, bit ACK disusun pada semua paket yang memasuki arah tersebut kecuali paket yang paling pertama yang berasal dari *client* ke *server*.

TCP *body* memuat data aktual yang ditransmisikan. Misalnya untuk *keystroke* atau layar *display* Telnet yang merupakan bagian pada Telnet *session*, atau untuk data FTP yang dikirim atau perintah yang dikeluarkan sebagai bagian dari FTP *session*.

IP

IP merupakan dasar menengah umum untuk Internet. IP mempunyai beberapa lapisan yang berbeda dibawahnya seperti Ethernet, *token ring*, FDDI, PPP, atau *carrier pigeon*. IP juga dapat mempunyai lapisan protokol lain di atasnya seperti

TCP, UDP, dan ICMP. Pada pembahasan ini, yang dibicarakan adalah karakteristik khusus IP yang berhubungan dengan penapisan paket.

IP *Option*

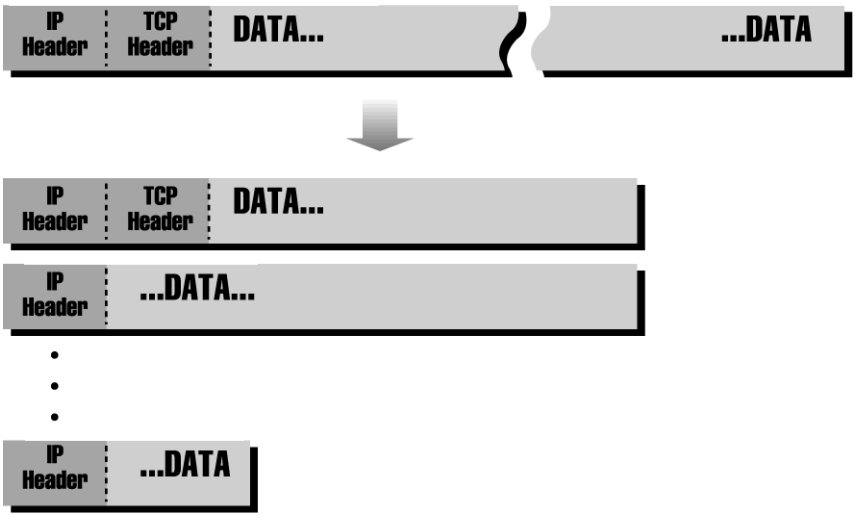
Seperti yang sudah dibahas mengenai lapisan IP, IP *header* mencakup *option field* yang biasanya kosong atau tidak berisi. Dalam perancangannya, IP *option filed* dimaksudkan sebagai tempat informasi khusus atau penanganan instruksi yang tidak mempunyai medan khususnya sendiri dalam *header*. Bagaimanapun juga, para perancang TCP/IP telah melakukan pekerjaan yang baik dalam menyediakan daerah segala sesuatu yang diperlukan, sehingga *option field* hampir selalu kosong. Dalam prakteknya, IP *option* jarang digunakan kecuali untuk usaha penerobosan dan (sangat jarang) untuk jaringan *debugging*.

IP *option* pada *firewall* yang paling umum akan dihadapkan dengan *option route* sumber IP. *Routing* sumber lebih membantu sumber paket menentukan *route* paket yang akan diantar ke tujuannya, daripada membantu setiap *router* di sepanjang penggunaan tabel *routing*-nya sendiri dalam hal memutuskan paket selanjutnya akan dikirim ke tujuan yang mana. Jadi, *routing* sumber mengesampingkan instruksi tabel *routing*. Menurut teori, *option routing* sumber berguna untuk pekerjaan di sekitar *router* dengan tabel *routing* yang rusak atau yang tidak benar. Jika diketahui *route* paket yang harus diambil, tetapi tabel *routing* rusak, informasi yang buruk dapat ditolak pada tabel *routing* dengan menentukan *option route* sumber IP yang tepat pada semua paket. Dalam prakteknya, *routing* sumber biasanya digunakan oleh penyerang yang berusaha mengelakkan ukuran-ukuran sekuritas dengan mendorong paket mengikuti jalan yang tidak diharapkan.

Banyak sistem penapisan paket mengambil pendekatan pendropan paket apapun yang telah diatur oleh IP *option*, tanpa berusaha untuk menunjukkan *option* apa itu atau apa arti dari *option* itu. Pada umumnya, hal ini tampak bekerja dengan baik tanpa menimbulkan persoalan-persoalan khusus.

Fragmentasi IP

Pertimbangan level IP yang lain dalam penapisan paket adalah fragmentasi. Salah satu karakteristik IP adalah kemampuannya dalam membagi paket yang besar yang tidak dapat melintasi beberapa mata rantai jaringan (karena pembatasan ukuran paket sepanjang mata rantai tersebut) kedalam paket yang lebih kecil, yang disebut dengan fragmen, yang dapat melewati mata rantai itu. Kemudian fragmen ini dikumpulkan lagi ke dalam paket yang utuh dengan menggunakan mesin tujuannya sendiri (bukan dengan menggunakan mesin dengan tujuan lain dari mata rantai yang terbatas; begitu paket difragmentasikan, ia tetap terfragmentasi sampai ia mencapai tujuannya). Gambar 4 menggambarkan fragmentasi IP.



Gambar 4 Fragmentasi data.

Dari sudut pandang penapisan paket, permasalahan dengan fragmentasi adalah bahwa hanya fragmen pertama yang akan memuat informasi *header* dari protokol tingkat tinggi seperti TCP, sehingga sistem penapisan paket diperlukan untuk memutuskan apakah paket utuh tersebut diijinkan atau tidak. Pendekatan penapisan paket umum dihubungkan dengan fragmentasi untuk mengijinkan fragmen apapun

yang bukan fragmen pertama lewat, dan melakukan penapisan paket pada fragmen pertama suatu paket. Hal ini aman karena, jika penapisan paket memutuskan untuk mendrop fragmen pertama, maka sistem tujuan tidak akan mampu mengumpulkan kembali sebagian fragmen yang tersisa kedalam paket asli, tanpa menghiraukan berapa banyak fragmen yang diterima. Jika paket asli tidak dapat direkonstruksi, maka paket yang dikumpulkan kembali sebagian tidak akan ke sini.

Host tujuan akan menyimpan fragmen untuk sementara waktu di dalam memorinya, menunggu untuk melihat apakah ada potongan-potongan yang hilang. Hal inilah yang membuat satu kemungkinan bagi penyerang untuk menggunakan paket yang difragmentasikan sebagai serangan penolakan layanan. Pada saat *host* tujuan menyerah dalam pengumpulan kembali paket, ia akan mengirimkan pesan ICMP "waktu untuk pengumpulan kembali paket habis" kembali ke *host* sumber. Keadaan ini akan memberitahukan mengapa sambungan tidak berhasil dan keberadaan *host* kepada penyerang. Tidak ada sesuatu yang dapat dilakukan terhadap serangan penolakan layanan, kecuali jika pesan ICMP terkait dapat ditapis.

Fragmen *outbound* mungkin memuat data yang sama sekali tidak ingin disiarkan ke seluruh dunia. Sebagai contoh, paket NFS *outbound* hampir dipastikan akan difragmentasikan, dan jika *file*-nya merupakan sesuatu yang rahasia, maka informasi tersebut akan disiarkan. Jika hal ini terjadi melalui suatu kecelakaan, maka mungkin tidak akan menjadi satu persoalan besar. Karena orang-orang biasanya tidak akan berkeliaran untuk melihat data dalam paket yang acak yang bukan merupakan suatu kasus yang menarik bagi mereka. Juga karena diperlukan waktu yang sangat lama dalam menantikan seseorang yang salah mengirimkan fragmen dengan data yang menarik didalamnya.

Jika seseorang yang berada di dalam secara sengaja menggunakan fragmentasi untuk mengirim data, berarti ada *user* yang berlawanan dalam *firewall*, dan tidak ada *firewall* yang secara sukses dapat mengatasi permasalahan ini. (Mereka mungkin bukan *user* lawan yang sangat cerdik, tetapi hanya karena terdapat cara yang mudah untuk mendapatkan data).

Satu-satunya situasi yang perlu dikhawatirkan mengenai fragmen *outbound* adalah situasi di saat suatu permintaan diijinkan, tetapi jawaban *outbound* diblok. Berarti jawaban dari fragmen bukan pertama akan keluar, sehingga penyerang mempunyai satu pengharapan dan ia akan berusaha mencarinya.

3.2 Protokol di Atas IP

IP merupakan dasar bagi sejumlah protokol yang berbeda, misalnya TCP, UDP, dan ICMP. Pembahasan juga mengenai *Remote Procedure* (RPCs) meskipun RPCs secara tegas didasarkan pada TCP atau UDP, bukan pada IP-nya sendiri. Karena RPCs, sama halnya dengan TCP dan UDP, dimaksudkan untuk beroperasi sebagai protokol *session* dengan tujuan umum, dengan protokol aplikasi dapat dijadikan lapisan. Pembahasan juga mengenai IP di atas IP secara ringkas (misalnya paket IP yang dienkapsulasi dengan paket IP lain), yang terutama digunakan untuk penyaluran paket IP *multicast* pada jaringan IP *non multicast*.

TCP

TCP merupakan protokol yang paling umum digunakan untuk layanan pada Internet. Sebagai contoh, Telnet, FTP, SMTP, NNTP, dan HTTP kesemuanya adalah layanan berbasis TCP. TCP memberikan sambungan dua arah yang dapat diandalkan antara dua *endpoints* (titik tujuan). Membuka sambungan TCP sama seperti membuat sambungan telepon. Yaitu, langkah pertama adalah memutar nomornya, dan setelah periode waktu *setup* yang singkat, sambungan yang dapat diandalkan terselenggara antara *user* dan siapa yang dipanggil *user*. TCP dapat diandalkan karena ia membuat tiga jaminan ke lapisan aplikasi, yaitu:

- tujuan akan menerima data aplikasi sesuai dengan permintaan yang dikirimkan,
- tujuan akan menerima semua data aplikasi, dan
- tujuan tidak akan menerima duplikat data aplikasi mana pun.

TCP lebih memilih mematikan sambungan daripada melanggar jaminan-jaminan tersebut. Sebagai contoh, jika paket TCP hilang di pertengahan *session* saat perjalanannya menuju ke tujuan, maka lapisan TCP akan menyusun paket tersebut untuk dikirim kembali sebelum data ditangani sampai ke lapisan aplikasi. Ia tidak akan menangani data yang diikuti dengan suatu kehilangan sampai ia mendapatkan data yang hilang itu. Jika sebagian data tidak dapat ditemukan, meskipun telah diusahakan berulang-ulang, lapisan TCP akan memilih mematikan sambungan dan melaporkan hal ini ke lapisan aplikasi daripada menangani data sampai ke lapisan aplikasi dengan terdapat celah kekosongan didalamnya.

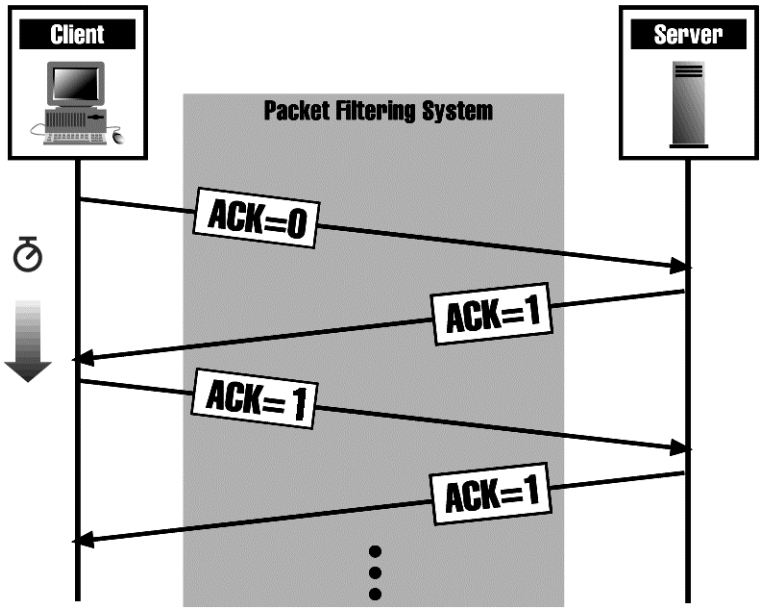
Jaminan-jaminan tersebut membutuhkan biaya tertentu pada waktu *setup* (dua sisi sambungan harus menukarkan informasi *startup* sebelum mereka benar-benar dapat mulai memindahkan data) dan pada saat pelaksanaan yang sedang dijalankan (dua sisi sambungan harus tetap mempertahankan status sambungan, untuk menentukan apa yang diperlukan data untuk dikirimkan kembali ke sisi lain dalam rangka memenuhi celah kekosongan dalam pembicaraan tersebut).

TCP bersifat dua-arah pada saat sambungan diselenggarakan, karena *server* dapat menjawab *client* pada sambungan yang sama. Tidak perlu menyelenggarakan satu sambungan dari *client* ke *server* untuk pertanyaan atau perintah dan satu sambungan lagi dari *server* kembali ke *client* sebagai jawaban.

Memblok sambungan TCP merupakan pekerjaan yang cukup sederhana, yaitu dengan memblok paket sambungan pertama. Karena tanpa paket pertama (yang merupakan butir penting karena berisi informasi *startup* sambungan), maka paket selanjutnya pada sambungan itu tidak akan dikumpulkan kembali ke dalam aliran data yang dibawa oleh penerima, dan sambungan itu tidak akan pernah dibuat. Paket pertama tersebut dapat dikenal karena bit ACK di dalam *header* TCP-nya tidak disusun. Sedangkan setiap paket lain dalam sambungan, ke arah mana pun ia berjalan tetap mempunyai susunan bit ACK.

Dengan pengenalan sambungan awal paket TCP, suatu kebijakan dapat ditetapkan untuk mengijinkan *client* internal berhubungan dengan *server* eksternal,

tetapi menghalangi *client* eksternal berhubungan dengan *server* internal. Hal ini memperkenankan sambungan awal paket TCP (paket tanpa bit ACK) diterapkan hanya pada *outbound* dan bukan *inbound*. Paket sambungan awal diijinkan keluar dari *client* eksternal ke *server* internal. Penyerang tidak dapat menumbangkan pendekatan ini hanya dengan menghidupkan bit ACK pada paket sambungan awal, karena tidak ada bit ACK yang dapat mengidentifikasi paket-paket tersebut sebagai paket sambungan awal.



Gambar 5 Bit ACK pada paket TCP.

Implementasi penapisan paket bervariasi dalam pelayanan dan bantuannya dalam menangani bit ACK. Beberapa implementasi penapisan paket memberikan akses langsung ke bit ACK, misalnya dengan membiarkan perusahaan memasukkan "*ack*" sebagai kata kunci dalam aturan penapisan paket. Beberapa implementasi lain memberikan akses tidak langsung ke bit ACK. Sebagai contoh, Cisco menggunakan kata kunci "*established*" bekerja dengan memeriksa bit ini, (*established* akan "benar" jika bit ACK disusun, dan akan "salah" jika bit ACK tidak disusun). Namun ada juga

beberapa implementasi yang sama sekali tidak mengijinkan adanya pengujian bit ACK. Gambar 5 menunjukkan ACK apa saja yang diatur pada paket yang merupakan bagian pada sambungan TCP.

UDP

Body pada paket IP bisa memuat paket UDP sebagai ganti paket TCP. UDP merupakan alternatif *overhead* rendah bagi TCP.

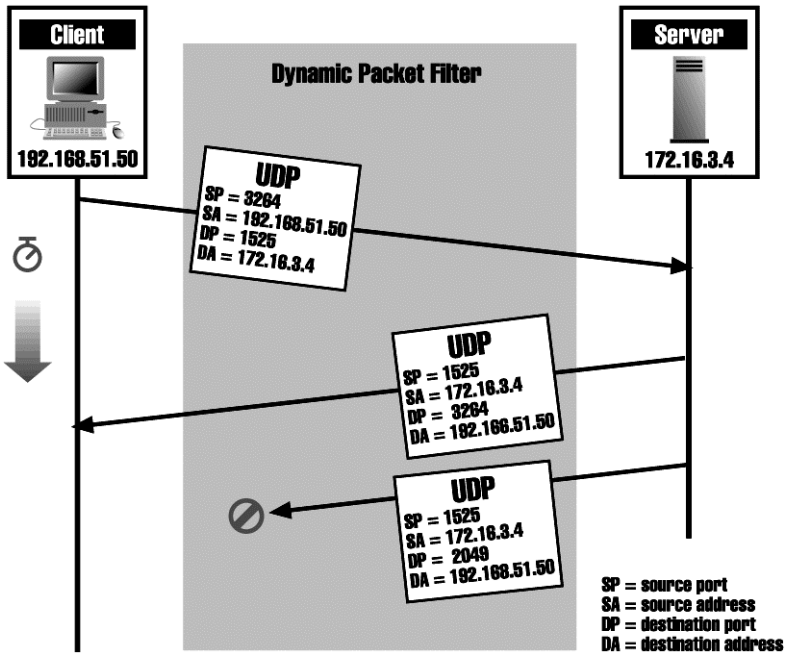
UDP adalah *overhead* rendah yang tidak membuat jaminan apapun yang dapat dipercaya (pengiriman, pemesanan, non duplikasi) seperti yang dapat dilakukan oleh TCP, dan oleh karenanya, ia tidak memerlukan mekanisme untuk membuat jaminan-jaminan tersebut. Setiap paket UDP bersifat independen dan bukan merupakan bagian dari *virtual circuit* seperti paket TCP. Pengiriman paket UDP seperti pendropan kartu positif dalam satu pengiriman, yaitu jika 100 kartu pos didrop dalam satu pengiriman, sekalipun mereka semuanya dialamatkan ke tempat yang sama, tetap saja tidak bisa dipastikan bahwa kesemuanya akan menuju kesana, dan kemungkinan kartu pos tersebut sampai ke tujuan sudah tidak berada dalam urutan yang sama persis ketika kartu pos dikirim.

Tidak sepenuhnya sama dengan kartu pos, paket UDP sesungguhnya dapat menyampaikan lebih dari sekali (tanpa harus merobek menjadi lembaran-lembaran, yang biasanya merupakan satu-satunya cara pengiriman kartu pos pada berbagai pengiriman). Dapat terjadi berbagai macam salinan karena paket dapat diduplikasikan melalui jaringan yang mendasarinya. Misalnya pada Ethernet, suatu paket akan diduplikasikan jika *router* memandang bahwa paket tersebut mungkin menjadi korban benturan Ethernet. Jika *router* salah, dan paket asli belum menjadi korban benturan Ethernet, maka baik yang asli maupun yang duplikat pada akhirnya akan disampaikan ke tujuan.

Semua persoalan tersebut bisa terjadi pada paket TCP, tetapi data akan dikoreksi sebelum disampaikan ke aplikasi. Lain halnya dengan UDP, dengan

aplikasi yang bertanggung jawab atas segala sesuatu yang berhubungan dengan paket, bukan dengan data yang dikoreksi.

Struktur paket UDP hampir sama dengan struktur paket TCP. UDP *Header* memuat bilangan *port* tujuan dan sumber UDP, seperti bilangan *port* tujuan dan sumber TCP. Namun, *header* UDP tidak memuat apapun yang menyerupai bit ACK. Bit ACK merupakan bagian mekanisme TCP untuk menjamin pengiriman data yang dapat dipercaya. Karena UDP tidak memberikan jaminan semacam itu, maka ia tidak membutuhkan bit ACK, sehingga tidak ada langkah-langkah yang dapat dilakukan oleh *router* penapisan paket untuk menentukan pengujian *header* paket UDP yang baru masuk dalam menentukan apakah paket itu merupakan paket pertama dari *client* eksternal ke *server* internal, atau tanggapan dari *server* eksternal kembali ke *client* internal.



Gambar 6 Penapisan paket dinamis pada lapisan UDP.

Beberapa implementasi penapisan paket seperti *CheckPoint's FireWall-1 product*, *Janus*, *Morning Star's SecureConnect Router*; dan the *KarlBridge /KarlBrouter*, mempunyai kemampuan "mengingat" paket UDP yang keluar yang mereka lihat. Namun selanjutnya mereka hanya dapat mengirimkan paket respons kembali melalui mekanisme penapisan. Agar diperhitungkan sebagai respons, paket yang masuk harus berasal dari *host* dan *port* tempat paket *outbound* dikirim, dan harus langsung dihubungkan ke *host* dan *port* yang mengirimkan paket *outbound*. Kemampuan ini seringkali disebut sebagai "penapisan paket dinamis", karena *router* memodifikasi aturan-aturan penapisan di udara untuk mengakomodasikan paket-paket yang dikembalikan disertai dengan pembatasan waktu. Dapat juga digunakan untuk situasi apapun meski aturan-aturan penapisan paket berubah asal tanpa secara eksplisit mengubah konfigurasi; produk yang berbeda menunjang kemampuan yang berbeda. Gambar 6 mengilustrasikan penapisan paket dinamis pada lapisan UDP.

ICMP

ICMP digunakan untuk pesan kontrol dan status IP. Paket ICMP dilaksanakan dalam *body* paket IP, sama halnya pada paket TCP dan UDP. Contoh-contoh pesan ICMP meliputi:

- permintaan *echo*, apa yang dikirimkan *host* ketika *ping* dijalankan;
- respons *echo*, apa tanggapan *host* terhadap "permintaan *echo*";
- melebihi batas waktu, apakah yang akan dikembalikan *router* pada saat ada ketentuan paket yang muncul harus di-*looping*;
- tujuan yang dapat dijangkau*,* apakah yang akan dikembalikan *router* pada saat tujuan paket tidak bisa dijangkau oleh karena beberapa alasan; dan
- *redirect*, apa yang akan disampaikan *router* kepada *host* dalam tanggapannya terhadap paket yang seharusnya dikirimkan ke *router* yang berbeda; *router* menangani paket asli dan *redirect* menyampaikan kepada *host* mengenai jalan yang lebih efisien yang harus ditempuh pada waktu lain.

Tidak seperti TCP atau UDP, ICMP tidak mempunyai sumber atau *port* tujuan, dan tidak ada protokol lain yang melapisi bagian atasnya. Sebagai gantinya terdapat seperangkat kode tipe pesan ICMP tertentu, yaitu kode khusus yang digunakan untuk mendikte interpretasi paket ICMP yang ada.

Banyak sistem penapisan paket mengijinkan paket ICMP ditapis berdasarkan *field* tipe pesan ICMP, banyak juga yang mengijinkan paket TCP atau UDP ditapis berdasarkan *field port* tujuan dan sumber TCP atau UDP.

RPC

Terdapat protokol *multiple remote procedure call* yang dikenal sebagai RPCs. Yang biasanya lebih dikenal sebagai "Sun RPC", karena pada mulanya dikembangkan oleh *Sun Microsystem*. Protokol inilah yang akan dibahas di sini dan merupakan protokol yang paling sering dijadikan acuan oleh RPC sederhana. Mekanisme *procedure remote call* yang lain lebih spesifik ke implementasi UNIX tertentu atau keluarga implementasinya (sebagai contoh, OSF DCE mempunyai protokol *procedure remote call* tersendiri). Mekanisme ini mempunyai perincian yang berbeda dengan RPC, tetapi cenderung mempunyai permasalahan yang sama.

Mekanisme RPC tidak dibangun di atas IP, tetapi agaknya ada pada bagian atas UDP dan TCP. Sama halnya dengan TCP dan UDP, RPC digunakan sebagai protokol *transport* yang mempunyai tujuan umum dengan variasi protokol aplikasi (seperti NFS dan NIS/YP, bisa dilihat pada lampiran). NFS dan NIS/YP merupakan layanan yang rentan terhadap serangan jika dilihat dari sudut pandang sekuritas jaringan. Seorang penyerang yang mempunyai akses pada *server* NFS dapat membaca *file* apapun dalam sistem. Dan penyerang yang mempunyai akses pada *server* NIS/YP dapat memperoleh *password file*, yang dapat dia gunakan untuk menjalankan serangan pemecahan *password* pada sistem.

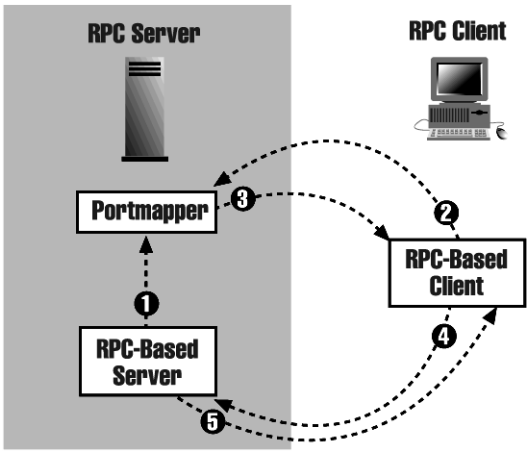
Dalam protokol TCP dan UDP, bilangan *port* merupakan medan dua *bite*. Ini berarti bahwa hanya terdapat 65.536 bilangan *port* yang mungkin untuk layanan TCP dan UDP. Tidak terdapat *port* yang cukup yang dapat digunakan untuk mengenali

bilangan *port* khusus pada setiap layanan dan aplikasi yang mungkin dikehendaki seseorang. RPC dapat mengatasi batasan ini. Setiap layanan yang berbasis RPC ditunjukkan dengan "bilangan layanan RPC" 4 bite yang unik. Hal ini memperkenankan 4.294.967.296 layanan yang berbeda, masing-masing dengan bilangan yang unik. Angka di atas lebih dari cukup untuk menentukan bilangan yang unik pada setiap layanan dan aplikasi yang dibutuhkan.

RPC dibangun di atas TCP dan UDP, sehingga dibutuhkan beberapa langkah pemetaan sejumlah layanan RPC pada *server* berbasis RPC yang digunakan pada mesin dengan *port* TCP atau UDP khusus yang digunakan *server*.

Portmapper merupakan satu-satunya *server* yang berhubungan dengan RPC yang dijamin untuk berjalan pada bilangan *port* TCP atau UDP khusus (keduanya pada bilangan *port* 111). Pada saat *server* berbasis RPC seperti *server* NFS atau NIS/YP dimulai, ia akan mengalokasikan *port* TCP dan/atau UDP secara acak (sebagian menggunakan salah satu, sebagian menggunakan yang lainnya, dan sebagian menggunakan keduanya) untuk kepentingannya sendiri. Kemudian ia akan menghubungi *server portmapper* pada mesin yang sama untuk mendaftar bilangan layanan RPC-nya yang unik dan *port* khusus yang digunakan pada waktu itu.

Program *client* berbasis RPC menghendaki hubungan antara *server* berbasis RPC khusus pada mesin pertama dengan *server portmapper* pada mesin itu (ingat bahwa TCP dan UDP dijalankan pada *port* 111). *Client* akan memberitahu *portmapper* bilangan layanan RPC yang unik untuk *server* yang hendak diakses, dan *portmapper* menanggapinya dengan suatu pesan, misalnya "maafkan saya, karena layanan itu untuk sementara waktu belum tersedia pada mesin ini", atau "layanan itu terakhir dijalankan pada *port* N TCP (atau UDP) pada mesin saat ini". Untuk butir ini, *client* menghubungi *server* pada bilangan *port* yang diperoleh dari *portmapper*, dan melanjutkan percakapannya secara langsung dengan *server*, tanpa keterlibatan yang lebih jauh dengan *portmapper*. Gambar 7 menunjukkan proses ini.



Gambar 7 RPC dan *portmapper*.

Keterangan Gambar 7.

1. *Server port* register dengan *port*mapper.
2. *Client* menghubungi *port*mapper, menanyakan tentang *server*.
3. *Port*mapper memberitahukan kepada *client*, *port server* apa yang digunakan.
4. *Client* menghubungi *server*.
5. *Server* memberikan respons ke *client*.

Sulit menggunakan penapisan paket untuk mengontrol layanan berbasis RPC, karena *port* layanan yang akan digunakan pada mesin khusus tidak diketahui. Karena kesempatan yang digunakan oleh *port* akan mengubah mesin setiap saat, yaitu *reboot*. Tidak cukup hanya dengan memblok akses pada *portmapper*. Penyerang dapat melewati langkah pembicaraan pada *portmapper*, dan akan mencoba semua *port* TCP dan UDP (65.536 *port* dapat diperiksa pada mesin khusus dalam beberapa menit), untuk mencari respons yang diharapkan dari *server* berbasis RPC seperti NFS atau NIS/YP.

Beberapa produk penapisan paket yang terbaru bisa berbicara dengan *portmapper* untuk menentukan layanan apa dan dapat menapis pada basis tersebut.

Dengan catatan sudah diuji pada basis persamaan-persamaan paket untuk layanan basis UDP. Penapisan paket akan menghubungi *portmapper* setiap kali ia menerima sebuah paket, karena jika mesin telah di-*reboot*, layanan akan hilang. Karena TCP mempunyai orientasi sambungan, bilangan *port* hanya diuji pada basis per sambungan. Penggunaan mekanisme ini mengijinkan layanan berbasis UDP menghasilkan *overhead* yang tinggi dan kemungkinan tidak bijaksana untuk aplikasi *data-intense* seperti NFS.

Meskipun hal ini belum mencukupi, namun akses pada *portmapper* tetap harus diblok, karena beberapa versi *portmapper* mampu digunakan sebagai *proxy* untuk *client* penyerang.

Jadi, apa yang harus dilakukan untuk melindungi layanan berbasis RPC? Terdapat dua observasi, pertama, ditemukan bahwa banyak layanan berbasis RPC yang membahayakan (terutama NIS/YP dan NFS) yang hanya ditawarkan melalui UDP. Kedua, sebagian besar layanan yang ingin diakses melalui penapisan paket berbasis TCP dan bukan berbasis UDP, keculai DNS, NTP, *syslog*, dan *Archie*. Observasi kembar ini membawa ke pendekatan situs-situs umum yang berhubungan dengan RPC yang menggunakan penapisan paket, yaitu blok UDP secara bersamaan, kecuali untuk “peepholes” yang dikontrol secara ketat dan khusus untuk DNS, NTP, *syslog* dan *Archie*.

Dengan pendekatan ini, jika perusahaan ingin mengijinkan layanan RPC berbasis TCP, maka perusahaan harus mengijinkan semuanya. *Server* NFS berbasis TCP walaupun tersedia tetap tidak dapat digunakan secara luas.

IP di atas IP

Dalam beberapa keadaan, paket IP dienkapsulasi dengan paket IP lain untuk transmisi, yang menghasilkan apa yang disebut “IP di atas IP”. Secara umum, kegunaan IP di atas IP adalah membawa paket IP *multicast* (yaitu, paket-paket dengan alamat tujuan *multicast*) antara jaringan yang mendukung *multicast* pada jaringan lanjutan yang tidak melakukan *multicast* paket IP. Untuk melewati jaringan

lanjutan ini, *router multicast* khusus (atau *mrouter*) pada setiap jaringan *multicast* mengenkapsulasi paket IP *multicast* yang akan dikirim ke dalam paket IP *nonmulticast* yang dialamatkan ke *mrouter* lain. *Mrouter* lain, dalam menerima paket *multicast* yang dienkapsulasi, mengosongkan paket luar (*nonmulticast*) dan kemudian menangani paket dalam (*multicast*).

3.3 Protokol di Bawah IP

Secara teoritis ada kemungkinan untuk menapis informasi dari bawah IP level. Sebagai contoh, alamat perangkat-keras Ethernet. Namun, jarang digunakan karena pada kebanyakan kasus, semua paket dari luar berasal dari alamat perangkat-keras yang sama (alamat *router* yang menangani sambungan Internet). Banyak *router* yang mempunyai sambungan *multiple* dengan protokol level yang lebih rendah. Sebagai hasilnya, melakukan penapisan pada level yang lebih rendah memerlukan konfigurasi *interface* yang berbeda dengan macam aturan yang berbeda untuk protokol level yang lebih rendah yang berbeda. Aturan yang ditulis tidak cukup hanya satu untuk diaplikasikan pada semua *interface* pada *router* yang mempunyai dua sambungan Ethernet dan sebuah sambungan FDDI, karena *header* paket Ethernet dan FDDI sementara sama tetapi tidak identik. Dalam prakteknya, IP adalah protokol level terendah tempat orang-orang memilih untuk melakukan penapisan paket .

3.4 Protokol Lapisan Aplikasi

Pada sebagian besar kasus, terdapat protokol di atas TCP atau UDP yang khusus untuk aplikasi. Protokol ini berbeda secara luas dalam spesifikasinya, dan perbedaannya bisa ratusan bahkan ribuan (sebagian besar merupakan aplikasi berbasis jaringan). Beberapa aplikasi penapisan paket terbaru menyediakan kemampuan menapis pada protokol lapisan aplikasi untuk aplikasi tertentu yang dikenal. Sebagai contoh, mereka mampu mengetahui informasi khusus dalam transaksi FTP untuk melakukan *setup* penapis dinamis, atau mereka mampu membandingkan informasi dalam paket ke aplikasi yang diduga sedang dijalankan,

untuk memastikan paket-paket yang dialamatkan ke *port* DNS benar-benar merupakan paket DNS.

IP Versi 6

Versi IP terakhir menurut pembahasan ini dikenal sebagai IP versi 4, tetapi pembahasan ini tidak berbicara tentang IP dengan kualifikasi tertentu. Namun terdapat versi IP terbaru saat ini, yang dikenal sebagai IP versi 6 (disingkat IPv6).

IPv6 didasarkan pada konsep *header* gabungan. Yang melakukan enkripsi dan *authentication*. *Field* protokol selanjutnya setelah *header* Ipv6 menentukan enkripsi atau *authentication header*. Pada giilirannya, *field* protokol selanjutnya akan mengindikasi IPv6 atau salah satu dari protokol *transport* yang digunakan, seperti TCP atau UDP.

IP di atas IP gabungan dapat dilakukan tanpa enskripsi atau *authentication*, sehingga dapat digunakan sebagai bentuk *routing* sumber. Cara yang lebih efisien adalah menggunakan *routing header* sumber, yang akan lebih berguna daripada menghubungi Ipv4 option, dan kemungkinan lebih banyak digunakan, khususnya untuk *mobile* IP.

Beberapa implikasi *firewall* telah terlihat. Penapisan paket harus mengikuti mata rantai *header* secara penuh, memahami dan memproses satu sama lain secara bergiliran. *Stance* yang teliti akan memerintahkan paket dengan *header* yang tidak dikenal untuk dikembalikan, baik *inbound* maupun *outbound*. Disamping itu, dengan kemudahan dan meratanya *routing* sumber berarti bahwa *cryptography authentication* sangat diperlukan. *Authentication* harus dibuat standar dengan ciri-ciri yang diperintahkan. Paket yang dienkripsi merupakan paket yang kabur, dan dengan demikian tidak dapat diuji; tentu saja hal ini benar untuk saat ini, tetapi tidak banyak enkriptor yang digunakan sekarang ini. Disamping itu perhatikan bahwa enkripsi dapat dilakukan *host-to-host*, *host-to-gateway*, atau *gateway-to-gateway*, namun masih dengan adanya komplikasi analisis.

Penapisan berbasis alamat juga terpengaruh dalam beberapa tingkatan oleh karena mekanisme *auto*-konfigurasi baru. Merupakan sesuatu yang penting bahwa *host* yang alamatnya disebutkan dalam penapis menerima alamat yang sama setiap waktu. Sementara ini merupakan tujuan mekanisme standar, juga diperlukan ketelitian mengenai skema yang memadai, *dial-up server*, dan sebagainya. Disamping itu, bit alamat dengan urutan yang tinggi dapat diubah, untuk mengakomodasikan kombinasi alamat berbasis *provider* dan mempermudah diantara pembawa.

IPv6 menggabungkan aliran-aliran. Aliran merupakan *virtual circuit* yang penting pada level IP, mereka dimaksudkan untuk digunakan sebagai sesuatu seperti video, *intermediate-hop ATM circuit selection*, dan lain-lain. Namun mereka juga dapat digunakan untuk *firewall*, yang memberikan *authentication* yang cukup memadai, yaitu persoalan mengenai jawaban UDP yang mungkin hilang jika permintaan mempunyai aliran *id* yang direferensikan oleh respons. Protokol *setup* aliran biasa tidak akan bekerja karena terlalu mahal. Tetapi *firewall traversal header* dapat melakukan pekerjaan ini.

Seperti yang dijelaskan di atas, IPv6 mempunyai pengaruh utama terhadap *firewall*, khususnya yang berhubungan dengan penapisan paket.

3.5 Protokol Non-IP

Protokol-protokol lain pada level yang sama seperti IPv6, misalnya *AppleTalk* dan IPX, menyediakan bentuk informasi yang sama sebagai IP, meskipun *header* dan operasi untuk protokol ini serta karakteristik penapisan paketnya sangat radikal. Banyak implementasi penapisan paket hanya menunjang penapisan IP, dan biasanya tidak untuk paket non-IP. Namun ada juga beberapa paket yang menyediakan penapisan paket yang terbatas untuk protokol non-IP, tetapi dukungan ini biasanya kurang fleksibel dan kurang mampu dibandingkan dengan kemampuan penapisan *router* IP.

Pada saat ini, penapisan paket sebagai suatu peralatan tidak sepopuler dan tidak lebih baik perkembangannya dibandingkan dengan protokol IP, mungkin hal ini

terjadi karena protokol-protokol tersebut jarang digunakan untuk melakukan komunikasi di luar perusahaan tunggal melalui Internet. (Internet menurut definisi merupakan jaringan dari jaringan-jaringan IP). Sebagian besar protokol non-IP merupakan persoalan bagi *firewall* yang bersifat internal bagi suatu perusahaan, dan untuk aplikasi ini, perlu dipilih satu paket khusus yang menunjang penapisan non-IP.

Pada Internet, protokol non-IP ditangani dengan mengenkapsulasinya dalam protokol IP. Pada sebagian besar kasus, perlu adanya pembatasan mengenai ijin dan penolakan protokol yang dienkapsulasi dalam keseluruhan mereka; semua sambungan *Appletalk* dalam UDP dapat diterima atau ditolak sama sekali. Beberapa paket yang menunjang protokol non-IP dapat mengetahui sambungan tersebut ketika dienkapsulasi dan pada saat *field* di dalamnya ditapis.

4.4 Tanggapan *Router* terhadap Penapisan Paket

Ketika *router* penapisan paket telah menyelesaikan pemeriksaan paket khusus, ada dua pilihan yang dapat ia lakukan terhadap paket, seperti berikut.

- Mengedarkan paket. Normalnya, jika paket berhasil melewati kriteria dalam konfigurasi penapisan paket, maka *router* akan meneruskan paket ke tujuannya, seperti yang akan dilakukan *router* normal (bukan *router* penapisan paket).
- Mendrop paket. Tindakan lain yang jelas harus dilaksanakan adalah mendrop paket jika ia gagal melewati kriteria dalam konfigurasi penapisan paket.

5 Aksi *Logging*

Tanpa mengabaikan apakah paket akan diteruskan atau akan didrop (diijinkan atau ditolak dalam beberapa implementasi penapisan paket), *router* dapat diminta untuk melakukan *logging* terhadap aksi yang telah dilakukan. Hal ini secara khusus akan benar apabila paket didrop, karena ia mendapatkan kesulitan dengan aturan-aturan penapisan paket yang ada. Dalam kasus ini, sesuatu yang sedang diusahakan yang berupa hal yang tidak diijinkan dapat diketahui.

Tidak semua paket yang diijinkan akan di-*logging*, tetapi *user* dapat memintanya. Misalnya untuk sambungan awal paket TCP, dengan demikian jalur sambungan TCP yang keluar masuk dapat dilindungi. Tidak semua penapisan paket akan melakukan *logging* terhadap paket-paket yang diijinkan.

Implementasi penapisan paket yang berbeda menunjang bentuk *logging* yang berbeda. Sebagian akan melakukan *logging* terhadap informasi khusus mengenai paket, dan yang lain akan diteruskan atau paket-paket yang didrop akan di-*logging* secara keseluruhan. Penapisan paket perlu dikonfigurasikan untuk melakukan *logging* terhadap *host* di mana saja melalui layanan *syslog*. Jangan menginginkan salinan *logging* ada pada penapisan paket hanya pada saat ia dikompromikan. Banyak penapisan paket juga terjadi pada *router* yang diberikan, yang jarang mempunyai jumlah kapasitas penyimpanan yang luas untuk melakukan *logging*.

5.1 Mengembalikan Kode ICMP *Error*

Jika paket akan didrop, maka *router* harus memutuskan boleh atau tidak boleh mengirimkan kembali kode ICMP *error* yang mengindikasi apa yang telah terjadi. Pengiriman kembali kode ICMP *error* mengakibatkan adanya peringatan bagi mesin pengiriman agar tidak mengulang kembali pengiriman paket tersebut, dengan demikian dapat menghemat lalulintas jaringan dan kadangkala berguna bagi *user* pada posisi yang jauh. (Jika kode ICMP *error* dikirim kembali, maka usaha sambungan *user* akan mengalami kegagalan dengan segera; dengan kata lain ia akan mengambil waktu *timeout* sebentar selama beberapa menit).

Terdapat dua susunan kode ICMP yang relevan, yaitu:

- kode umum “tujuan tidak dapat dicapai” --- kode khusus “*host* tidak dapat dicapai” dan kode “jaringan tidak dapat dicapai”, dan
- kode umum “tujuan administratif tidak dapat dicapai” --- kode khusus “*host* secara administratif tidak dapat dicapai” dan kode “jaringan administratif tidak dapat dicapai”.

Para perancang ICMP menghendaki pasangan pertama dari kode ICMP *error* saat *router* dapat mengembalikan, "*host* tidak dapat dicapai" atau "jaringan tidak dapat dicapai", untuk mengindikasi permasalahan jaringan yang serius, yaitu *host* tujuan telah jatuh atau sesuatu yang ada di jalan *host* telah jatuh. Kode *error* ini mendahului *firewall* dan penapisan paket. Permasalahan yang muncul dalam mengembalikan salah satu dari kode *error* tersebut adalah bahwa terdapat beberapa *host* (khususnya jika mereka menjalankan versi UNIX lama) mengambil kode tersebut secara mentah. Jika mesin tersebut menerima kembali "*host* tidak dapat dicapai" untuk *host* tertentu, maka mereka akan berasumsi bahwa *host* itu sepenuhnya tidak dapat dicapai dan akan menutup semua sambungan yang terbuka, meskipun ada sambungan lain yang sebenarnya diijinkan oleh penapisan paket.

Susunan kedua dari kode *error* ICMP tempat *router* dapat mengembalikan "*host* administratif tidak dapat dicapai" dan "jaringan administratif tidak dapat dicapai", telah ditambahkan ke daftar resmi tipe pesan ICMP beberapa tahun yang lalu, khusus untuk memberikan sistem penapisan paket sesuatu yang dapat dikembalikan ketika mereka mendrop paket. Banyak sistem yang sama sekali belum mengenal kode ini, meskipun mereka tidak menyebabkan persoalan bagi sistem. Sistem dapat mengabaikan kode ICMP *error* yang tidak mereka pahami, hal ini hampir sama dengan mengembalikan kode *no error* ke beberapa sistem. Sebaliknya, banyak peralatan-peralatan mengikuti standar yang sangat lemah, dan ini merupakan jenis kondisi pembatasan yang sebagian diantaranya tidak ditangani dengan baik. Fakta yang menunjukkan bahwa sistem memerlukan standar dalam mengabaikan kode *error* yang tidak dikenal tidak menjamin bahwa suatu sistem tidak akan mempunyai raksi yang fatal terhadap beberapa kode *error*.

Terdapat beberapa hal yang perlu dipertimbangkan pada saat mengambil keputusan apakah sistem penapisan paket akan mengembalikan kode ICMP *error* atau tidak, antara lain sebagai berikut.

- Pesan mana yang harus dikirimkan?
- Dapatkah diusahakan *overhead* mengenai kode *error* yang dihasilkan dan yang dikembalikan?
- Apakah dengan mengembalikan kode tersebut memungkinkan penyerang untuk mendapatkan informasi yang lebih banyak mengenai penapisan paket?
- Kode *error* mana yang dapat menghasilkan pengertian mengenai situs?

Pengembalian kode "*host* tidak dapat dicapai" dan "jaringan tidak dapat dicapai", secara teknis merupakan hal yang tidak benar (ingat bahwa *host* dapat atau tidak dapat dicapai menurut kebijakan penapisan paket tergantung pada *host* apa yang berusaha mengakses layanan apa). Disamping itu, kode *error* dapat menyebabkan banyak sistem bereaksi secara berlebihan (mematikan semua sambungan yang ada pada *host* atau jaringan).

Pengembalian kode "*host* administratif tidak dapat dicapai" atau kode "jaringan administratif tidak dapat dicapai" mengiklankan fakta bahwa terdapat sistem penapisan paket pada situs yang bisa memutuskan boleh atau tidak boleh keinginan untuk tidak mengerjakannya. Kode-kode ini juga dapat menyebabkan reaksi yang berlebihan dalam implementasi IP yang salah.

Disamping itu juga terdapat pertimbangan lain. Membuat dan mengembalikan kode ICMP *error* mengambil sejumlah usaha kecil tertentu pada bagian *router* penapisan paket. Penyerang diperkirakan dapat meningkatkan penolakan serangan layanan dengan membanjiri *router* dengan paket-paket yang akan ditolak oleh *router*, dan ia akan berusaha menghasilkan paket ICMP *error*. Persoalannya bukan jaringan *bandwith*, tetapi mengenai muatan CPU pada *router*. (Sementara ia sibuk membuat paket-paket ICMP, ia tidak akan mampu melakukan sesuatu yang lain secepatnya, seperti membuat keputusan penapisan). Sebaliknya, jika kode ICMP *error* tidak dikembalikan berarti akan menyebabkan sejumlah kecil lalulintas jaringan yang saling berdesakan, karena sistem pengiriman berusaha dan berusaha lagi untuk mengirim paket yang didrop. Lalulintas ini tidak akan bertambah banyak, karena

jumlah paket yang diblok dengan sistem penapisan paket menjadi satu pecahan dari total jumlah paket yang diproses. (Jika bukan berupa pecahan kecil, bisa terjadi persoalan serius, karena orang-orang akan mengupayakan banyak hal yang tidak diijinkan).

Jika *router* mengembalikan kode ICMP *error* untuk setiap paket yang menyimpang dari kebijakan penapisan, berarti memberikan jalan bagi penyerang untuk membuktikan sistem penapisan. Dengan mengamati paket-paket yang menimbulkan respons ICMP *error*, penyerang dapat menemukan tipe paket apa yang menyimpang dan yang tidak menyimpang dari kebijakan (dan dengan demikian ia akan mengetahui tipe paket apa yang diijinkan dan yang tidak diijinkan untuk masuk ke dalam jaringan). Informasi ini akan memudahkan pekerjaan penyerang. Penyerang mengetahui ke mana arah paket-paket yang tidak menerima ICMP *error*, dan ia dapat memusatkan perhatian pada protokol tersebut, kondisi ini merupakan kondisi yang mudah diserang.

Setelah melihat penjelasan di atas, tampaknya yang paling aman untuk dikerjakan adalah mendrop paket tanpa mengembalikan kode ICMP *error* apapun. Jika *router* menawarkan fleksibilitas yang cukup, maka ia dapat membuat konfigurasi agar dapat mengembalikan kode ICMP *error* ke sistem internal (yang tampaknya akan mengetahui dengan segera bahwa ada sesuatu yang sedang mengalami kegagalan, daripada menunggu *timeout*), tetapi bukan ke sistem eksternal (yang dapat memberikan informasi kepada penyerang untuk memeriksa konfigurasi penapisan *firewall*). Sekalipun *router* tampaknya tidak menawarkan fleksibilitas semacam itu, namun masih ada kemungkinan untuk menyelesaikan hasil yang sama dengan menentukan aturan-aturan penapisan paket yang mengijinkan paket-paket ICMP *inbound* yang relevan dan tidak mengijinkan paket-paket ICMP *outbound* yang relevan.

5.2 Aturan Penapisan Paket

Terdapat bermacam-macam aturan yang dapat ditentukan untuk *router* penapisan paket dalam mengontrol paket-paket apa saja yang dapat dan tidak dapat mengalir keluar masuk jaringan. Terdapat beberapa hal yang perlu diketahui mengenai aturan-aturan ini.

Untuk menghindari kekacauan, contoh aturan-aturan lebih baik ditetapkan berdasarkan deskripsi yang abstrak sebisa mungkin daripada berdasarkan alamat nyata. Selain menggunakan sumber dan alamat tujuan yang nyata (misalnya, 172.16.51.50), juga menggunakan "Internal" atau "Eksternal" untuk mengidentifikasi jaringan mana yang sedang dibicarakan. Sistem penapisan paket yang aktual biasanya menuntut penentuan *range* alamat secara eksplisit dan dengan sintaksis yang bervariasi dari *router* ke *router*.

Pada semua contoh penapisan paket terdapat asumsi bahwa pada setiap paket, *router* berjalan dengan aturan-aturan sampai ia menemukan aturan yang cocok, dan kemudian ia mengambil suatu tindakan yang ditetapkan oleh aturan tersebut. *Default deny* diasumsikan sebagai yang implisit jika tidak ada aturan-aturan yang diaplikasikan, meskipun menentukan *default* yang eksplisit juga merupakan gagasan yang baik (yang biasa umum dilakukan).

Sintaksis yang digunakan dalam contoh penapisan menentukan jumlah bit yang penting untuk dibandingkan dengan alamat lain setelah karakter *slash* (/). Jadi, 10.0.0.0/8 mencocokkan alamat yang dimulai dengan 10; ekuivalen dengan 10.0.0.0 dengan *net-mask* UNIX 255.0.0.0 atau 10.0.0.0 dengan *Cisco wildcard mask* 0.255.255.255 atau (jika ada *filename*) 10.*.*.*.

Meskipun sudah dijelaskan secara khusus dalam contoh, namun tidak dapat dijelaskan secara tepat apa yang harus ditetapkan untuk produk penapisan paket khusus. Mekanisme yang tepat untuk menetapkan *router* penapisan paket bervariasi dari satu produk dengan produk yang lain. Sebagian produk (seperti paket *screend*) mengijinkan penentuan aturan-aturan tunggal yang diaplikasikan pada semua paket yang ditempuh oleh sistem. Sebagian yang lain (seperti *Telebit NetBlazer*)

mengijinkan penentuan aturan-aturan *interface* khusus. Sebagian lagi yang lain (seperti produk *Livingston* dan *Cisco*), mengijinkan penentuan sejumlah aturan dan selanjutnya diaplikasikan dengan nama pada *interface* khusus (sehingga aturan-aturan yang dibagi dengan jumlah *interface* yang berbeda dapat ditentukan, dan dapat meletakkan aturan-aturan yang khusus pada *interface* tertentu ke dalam aturan-aturan yang berbeda).

Contoh sederhana di bawah ini mengilustrasikan perbedaan itu. Dipilih tiga sistem tersebut karena mereka menunjukkan suatu langkah yang berbeda dalam menentukan penapis, bukan karena kecenderungan khusus apapun. Secara umum, sistem lain sama dengan sistem ini. Sebagai contoh, produk *Cisco* sama dengan produk *Livingston* dalam hal menetapkan aturan kemudian mengaplikasikannya pada paket yang berjalan ke arah khusus melalui *interface* tertentu. Sintaksis mereka sebenarnya berbeda, seperti cara mereka menentukan *host* alamat dan *bitmask*.

Seandainya semua lalulintas IP antara *host* eksternal yang dipercaya (*host* 172.16.51.50) dan *host* pada jaringan internal (kelas jaringan C 192.168.10.0) diijinkan, maka kasus ditunjukkan dalam Tabel 1.

Tabel 1 Aturan yang diaplikasikan apabila semua lalulintas IP antara *host* eksternal dan *host* pada jaringan internal diijinkan

Aturan	Jurusan	Alamat Sumber	Alamat Tujuan	Set ACK	Aksi
A	*Inbound*	*Trusted external host*	Internal	Apa saja	Mengijinkan
B	*Outbound*	Internal	*Trusted external host*	Apa saja	Mengijinkan
C	Salah satu	Apa saja	Apa saja	Apa saja	Menolak

Dengan *screend*, yang harus ditentukan adalah:

```
between host 172.16.51.50 and net 192.168.10 accept ;
between host any and host any reject ;
```

Dengan *Telebit Netblazer*, yang harus ditentukan adalah aturan *interface* yang akan diaplikasikan dan kapan aturan akan menggunakan paket *incoming* dan *outgoing* pada *interface*. Jika *interface* eksternal dinamakan “syn0”, aturan harus:

```
permit 172.16.51.50/32  192.168.10/24 syn0 in
deny 0.0.0.0/0 0.0.0.0/0 syn0 in

permit 192.168.10/24  172.16.51.50/32 syn0 out
deny 0.0.0.0/0  0.0.0.0/0 syn0 out
```

Pada *Livingston PortMaster* atau IRX, aturan harus ditentukan dalam satu set dan kemudian menggunakan set yang relevan untuk tujuan yang benar pada *interface* yang benar. Jika *interface* eksternal dinamakan "s1", aturan yang dibuat harus berupa sesuatu yang seperti:

```
add filter s1.in
set filter s1.in 1 permit 172.16.51.50/32  192.168.10.0/24
set filter s1. In 2 deny 0.0.0.0/0  0.0.0.0/0
set s1 ifilter s1.in

add filter s1.out
set filter s1.out 1 permit 192.168.10.0/24  172.16.51.50/32
set filter s1.out 2 deny 0.0.0.0/0  0.0.0.0/0
set s1 ofilter s1.out
```

Pada *router Cisco*, aturan juga harus ditentukan dalam satu set, dan gunakan set yang relevan untuk tujuan yang benar pada *interface* yang benar. Jika *interface* eksternal dinamakan "serial 1", maka aturan-aturan akan tampak seperti:

```
access-list 101 permit ip 172.16.51.50  0.0.0.0  192.168.10.0  0.0.0.255
access-list 101 deny ip 0.0.0.0  255.255.255.255   0.0.0.0  255.255.255.255
interface serial 0
access-group 101 in

access-list 102 permit ip 192.168.10.0  0.0.0.255  172.16.51.50  0.0.0.0
access-list 102 deny ip 0.0.0.0  255.255.255.255   0.0.0.0  255.255.255.255
interface serial 0
access-group 101 out
```

Untuk informasi terperinci mengenai sintaksis untuk paket atau produk khusus, lihatlah dokumentasi pada paket atau produk tersebut. Dalam memahami sintaksis pada sistem khusus yang digunakan, terdapat banyak kesulitan yang akan dihadapi dalam penerjemahan dari tabel ke dalam sintaksis sistem.

Berhati-hatilah dengan *default implisit*. Sistem penapisan yang berbeda mempunyai langkah-langkah *default* yang berbeda yang mereka lakukan jika paket tidak menyamai aturan-aturan penapisan khusus. Beberapa sistem menolak semua paket-paket tersebut. Sistem yang lain membuat *default* bertentangan dengan aturan

terakhir; yaitu jika aturan terakhir adalah "*permit*", maka *default* sistemnya adalah "*deny*", dan jika aturan terakhir adalah "*deny*", maka *default* sistemnya adalah "*permit*". Dalam beberapa kasus, meletakkan aturan *default* pada akhir daftar aturan-aturan penapisan paket merupakan ide yang cukup bagus, sehingga tidak perlu ada kekhawatiran mengenai *default* implisit sistem yang sedang digunakan.

6. Bentuk Penapisan Paket

Ada dua macam bentuk penapisan paket, yaitu penapisan paket menurut alamat dan penapisan paket menurut layanan.

6.1 Penapisan Menurut Alamat

Bentuk yang paling sederhana, meskipun bukan yang paling umum, untuk bentuk penapisan paket adalah penapisan menurut alamat. Penapisan ini membantu membatasi arus paket berbasis sumber dan/atau alamat tujuan paket, tanpa harus mempertimbangkan protokol apa yang digunakan. Sebagai contoh, penapisan ini dapat digunakan untuk mengijinkan *host* eksternal tertentu berbicara ke *host* internal tertentu, atau mencegah penyerang memasukkan paket-paket palsu ke dalam jaringan.

Sebagai contoh, seandainya paket-paket dengan alamat sumber palsu yang masuk akan diblok; maka aturan yang harus ditentukan seperti Tabel 2 berikut ini.

Tabel 2 Aturan penapisan menurut alamat

Aturan	Jurusan	Alamat Sumber	Alamat Tujuan	Aksi
A	*Inbound*	Internal	Apa saja	Menolak

Perhatikan bahwa jurusan relatif berhubungan dengan jaringan internal. Pada *router* antara jaringan internal dan Internet, aturan *inbound* dapat diaplikasikan ke dalam paket-paket yang baru masuk pada *interface* Internet atau ke paket-paket yang keluar dari *interface* Internet. Dengan cara ini, hasil yang sama dapat diicapai untuk

host yang diproteksi. Perbedaannya ada pada apa yang dilihat *router*. Jika paket-paket yang keluar disaring, maka *router* tidak akan melakukan proteksi dengan sendirinya.

Resiko Penapisan dengan Alamat Sumber

Sebenarnya tidak begitu aman mempercayai alamat sumber karena alamat sumber dapat dipalsukan. Kecuali jika digunakan beberapa macam *cryptographic autehentication* antara *user* dan *host* yang ingin diajak bicara, dan *user* tidak akan tahu jika ia benar-benar berbicara dengan *host* tersebut, atau beberapa mesin lain yang cenderung menjadi *host* tersebut. Penapis yang telah dibahas di atas bisa membantu apabila *host* eksternal mengklaim sebagai *host* internal, tetapi penapis itu tidak bisa melakukan apapun apabila *host* eksternal mengklaim sebagai *host* eksternal yang berbeda.

Dasar serangan penipuan alamat sumber (ditunjukkan pada Gambar 1) adalah penyerang mengirim paket-paket yang mengklaim berasal dari seseorang yang dipercayai dalam beberapa hal, dengan harapan untuk mendorong *user* mengambil langkah-langkah yang didasarkan pada kepercayaan tersebut, tanpa berharap untuk mendapatkan paket apapun kembali ke mereka. Jika penyerang tidak peduli dengan penerimaan paket-paket yang kembali ke mereka, maka dia tidak perlu berada di jalan antara *user* dan siapapun yang dia maksudkan; dia bisa berada di mana-mana.

Dalam kenyataannya, tanggapan *user* akan menuju siapa saja yang dimaksud oleh penyerang, bukan ke penyerang itu sendiri. Apabila penyerang dapat memprediksikan tanggapan tersebut, mereka tidak perlu melihatnya. Banyak (jika bukan sebagian besar) protokol yang dapat diprediksikan, cukup bagi penyerang untuk melakukan pekerjaannya dengan sukses.

Dalam beberapa keadaan, terutama keadaan yang meliputi sambungan TCP, mesin nyata (mesin incaran penyerang) akan bereaksi terhadap paket-paket *user* (paket yang berusaha melakukan pembicaraan yang tidak diketahuinya) dengan melakukan *reset* sambungan *bogus*; dan penyerang tidak menginginkan hal ini terjadi. Dia harus menjamin serangan sempurna sebelum mesin nyata dan

mendapatkan paket-paket yang dikirim *user*, atau sebelum *user* mendapat paket *reset* dari mesin nyata. Terdapat sejumlah cara untuk dapat menjamin hal ini, misalnya :

- melakukan serangan pada saat kondisi mesin nyata sedang turun,
- melanggar mesin nyata sehingga serangan dapat dilakukan,
- membanjiri mesin nyata ketika serangan dilakukan,
- mengacaukan *routing* antara mesin nyata dan target, dan
- menggunakan serangan dengan hanya paket respons pertama yang diperlukan, sehingga *reset* tidak menjadi masalah bagi mereka.

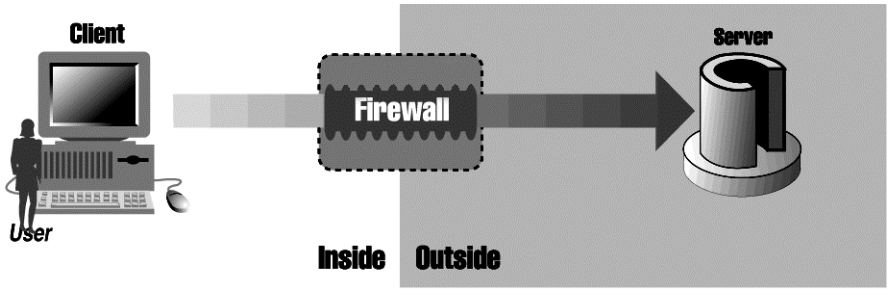
6.2 Penapisan Menurut Layanan

Pemblokan paket palsu yang baru masuk hanyalah merupakan kegunaan umum penapisan menurut alamat. Namun sebagian besar kegunaan penapisan paket meliputi penapisan menurut layanan, dan ini lebih komplikasi lagi.

Dari sudut pandang penapisan paket, seperti apa paket-paket yang bekerja sama dengan layanan khusus? Ambil contoh perincian Telnet. Telnet mengijinkan *user* untuk melakukan *log in* ke sistem lain, seolah-olah *user* mempunyai terminal dengan sambungan langsung ke sistem tersebut. Dalam pembahasan ini, yang dijadikan sebagai contoh adalah Telnet karena Telnet sangat umum, sangat sederhana, dan dari sudut pandang penapisan paket merupakan perwakilan dari beberapa protokol lain seperti SMTP dan NNTP. Yang perlu dilihat adalah *outbound* dan *inbound* layanan Telnet.

Outbound Layanan Telnet

Dalam *outbound* layanan Telnet, *client* lokal (seorang *user*) melakukan pembicaraan dengan *server* yang jauh. Di sini yang perlu ditangani adalah paket-paket yang masuk dan yang keluar. (Gambar 8 menunjukkan pandangan yang sederhana mengenai *outbound* Telnet).



Gambar 8 *Outbound* Telnet.

Paket-paket yang keluar untuk layanan *outbound* ini berisi *user keystroke* dan mempunyai karakteristik sebagai berikut.

- Alamat sumber IP paket yang keluar merupakan alamat IP *host* lokal.
- Alamat tujuan IP merupakan alamat IP *host* jarak-jauh.
- Telnet adalah layanan berbasis TCP, sehingga tipe paket IP adalah TCP.
- *Port* tujuan TCP adalah 23; yaitu bilangan *port* yang dikenal yang digunakan oleh *server* Telnet.
- Bilangan *port* sumber TCP (yang disebut "Y" dalam contoh ini) merupakan bilangan sembarang yang lebih besar dari 1023.
- Paket yang keluar pertama akan menciptakan sambungan yang tidak mempunyai susunan bit ACK; sedangkan sisa dari paket-paket keluar yang lain mempunyai susunan bit ACK.

Paket-paket yang masuk untuk layanan *outbound* ini berisi data yang akan ditunjukkan pada layar *user* (misalnya "*login*:" *prompt*) dengan karakteristik berikut.

- Alamat sumber IP paket yang masuk merupakan alamat IP *host* jarak jauh.
- Alamat tujuan IP merupakan alamat IP *host* lokal.
- Tipe paket IP adalah TCP.
- *Port* sumber TCP adalah 23; yaitu *port* yang digunakan *server*. *Port* tujuan TCP sama dengan yang digunakan sebagai *port* sumber untuk paket-paket yang keluar, yaitu "Y".

- Semua paket yang masuk mempunyai susunan bit ACK (kecuali sambungan pertama, dalam contoh ini, paket pertama adalah paket yang keluar, bukan paket yang masuk).

Perhatikan persamaan antara *header field* dari paket yang keluar dan yang masuk pada Telnet, yaitu menggunakan alamat dan bilangan *port* yang sama; hanya ditukar antara sumber dan tujuan.

Port client, yaitu *port* sumber untuk paket yang keluar dan *port* tujuan untuk paket yang masuk, dibatasi tidak lebih dari 1023. Ini merupakan suatu pengesahan versi BSD sistem UNIX, dengan hampir semua kode jaringan UNIX dapat ditelusuri asal mulanya. *Port* cadangan BSD UNIX dari 0 sampai 1023 untuk pemakaian lokal hanya digunakan oleh *root*. *Port* biasanya hanya digunakan oleh *server*, bukan *client* (kecuali BSD "*router command*" seperti *rcp* dan *rlogin*). Sistem operasi lain, bahkan yang tidak mempunyai konsep yang analog untuk *user root* yang istimewa, misalnya sistem *Macintosh* dan MS-DOS, juga mengikuti perjanjian ini. Ketika program *client* memerlukan bilangan *port* untuk pemakaian mereka, maka bilangan *port* lama akan melakukan hal yang sama, dan program menentukan *port* diatas 1023.

Inbound LayananTelnet

Dalam *inbound* layanan Telnet, *client* yang jauh (seorang *user* yang jauh) berkomunikasi dengan *server* Telnet lokal. Sama dengan *outbound* layanan Telnet, yang perlu ditangani adalah paket-paket yang masuk dan yang keluar. Paket yang masuk bagi *inbound* layanan Telnet memuat *user keystroke* dengan karakteristik berikut.

- Alamat sumber IP dari paket-paket tersebut merupakan alamat *host* yang jauh.
- Alamat tujuan IP merupakan alamat *host* lokal.
- Tipe paket IP adalah TCP.
- *Port* sumber TCP merupakan beberapa bilangan *port* acak yang lebih besar dari 1023 (yang akan disebut "Z" dalam contoh ini).

- *Port* tujuan TCP adalah 23.
- TCP ACK bit tidak akan disusun pada setiap paket *inbound* pertama yang menciptakan sambungan, tetapi disusun pada semua paket *inbound* yang lain.

Paket yang keluar bagi *inbound* layanan Telnet berisi tanggapan *server* (data yang akan ditunjukkan ke *user*) dan mempunyai karakteristik sebagai berikut.

- Alamat sumber IP merupakan alamat *host* lokal.
- Alamat tujuan IP merupakan alamat *host* yang jauh.
- Tipe paket IP adalah TCP.
- *Port* sumber TCP adalah 23 (paket tersebut berasal dari *server* Telnet).
- *Port* tujuan TCP merupakan *port* sembarang "Z", sama dengan yang digunakan paket-paket *inbound* sebagai *port* sumber.
- TCP ACK bit akan disusun pada semua paket yang akan keluar.

Perhatikan persamaan antara *header* yang relevan dengan paket yang keluar masuk, yaitu alamat sumber dan tujuan ditukar dan *port* sumber dan tujuan ditukar.

Ringkasan Telnet

Tabel 3 menggambarkan berbagai tipe paket yang terlibat dalam *inbound* dan *outbound* layanan Telnet.

Tabel 3 Tipe paket yang terlibat dalam *inbound* dan *outbound* layanan Telnet

Jurusan Layanan	Jurusan Paket	Alamat Sumber	Alamat Tujuan	Tipe Paket	*Port* Sumber	*Port* Tujuan	Susunan ACK
Outbound	Keluar	Internal	Ekternal	TCP	Y	23	a
Outbound	Masuk	Eksternal	Internal	TCP	23	Y	*Yes*
Inbound	Masuk	Eksternal	Internal	TCP	Z	23	a
Inbound	Keluar	Internal	Eksternal	TCP	23	Z	*Yes*

a adalah TCP ACK bit yang akan disusun pada semua paket pertama, yang menciptakan sambungan.

Ingat bahwa Y dan Z keduanya merupakan bilangan *port* sembarang diatas 1023 (dari sudut pandang sistem penapisan paket).

Jika Telnet yang akan keluar diijinkan, dan tidak ada yang lain, maka penapisan paket akan di-*setup* seperti Tabel 4.

Tabel 4 *Setup* penapisan paket apabila Telnet yang akan keluar diijinkan

Aturan	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Sumber	*Port* Tujuan	Susunan ACK	Aksi
A	Keluar	Internal	Apa saja	TCP	>1023	23	Salahsatu	Mengijinkan
B	Masuk	Apa saja	Internal	TCP	23	>1023	*Yes*	Mengijinkan
C	Salah satu	Apa saja	Apa saja	Apa saja	Apa saja	Apa saja	Salahsatu	Menolak

- Aturan A mengijinkan paket keluar ke *server* Telnet yang jauh.
- Aturan B mengijinkan paket kembali masuk ke dalam. Bit ACK akan dibuktikan susunannya, oleh karena itu aturan B tidak dapat disalahgunakan oleh penyerang untuk mengijinkan sambungan TCP yang masuk dari *port* 23 atas tujuan penyerang ke *port* diatas 1023 pada tujuan *user*. Contoh, *server* X11 pada *port* 6000.
- Aturan C merupakan aturan *default*. Jika tidak ada satupun aturan yang ditambahkan sebelumnya, maka paket akan diblok. Ingat bahwa paket yang diblok akan di-*logging*, dan bahwa ia dapat atau tidak dapat menyebabkan pesan ICMP dikembalikan ke pemiliknya.

Resiko Penapisan Menurut *Port* Sumber

Membuat keputusan penapisan berdasarkan pada *port* sumber bukan tanpa resiko. Terdapat satu persoalan besar dalam tipe penapisan ini, kepercayaan terhadap *port* sumber hanya dapat seimbang dengan kepercayaan terhadap mesin sumber.

Andaikan *port* sumber yang bekerja sama dengan layanan khusus salah diasumsi. Siapapun yang berada dalam pengawasan mesin sumber, misalnya seseorang dengan akses *root* pada sistem UNIX (atau siapapun dengan jaringan PC) dapat menjalankan apa yang diinginkan *client* dan *server* pada *port* sumber, yang sebenarnya hanya diijinkan melalui sistem penapisan paket yang telah dikonfigurasikan. *User* tidak dapat mempercayai alamat sumber yang memberitahukan mengenai mesin sumber itu karena *user* tidak dijamin secara pasti

apakah membicarakan alamat tersebut ke mesin nyata, atau malah ke penyerang yang mengaku sebagai mesin tersebut.

Yang dapat dilakukan sehubungan dengan situasi ini adalah membatasi bilangan *port* lokal sebanyak mungkin, tanpa terkecuali beberapa *port* yang jauh yang diijinkan untuk mengaskes mereka. Jika sambungan *inbound* ke *port* 23 saja yang diijinkan, dan jika *port* 23 mempunyai *server* Telnet yang dapat dipercaya (*server* hanya akan melakukan sesuatu atas permintaan *client* Telnet yang boleh dilakukannya), berarti bukan merupakan satu persoalan, baik program yang berbicara dengannya merupakan *client* Telnet asli atau bukan. Yang harus dikonsentrasikan adalah membatasi sambungan *inbound* hanya ke *port* tempat *server* yang dipercaya dijalankan, dan pastikan bahwa *server* tersebut benar-benar dapat dipercaya.

Karena banyak layanan yang menggunakan *port* sembarang diatas 1023 untuk *client*, dan karena sebagian layanan menggunakan *port* di atas 1023 untuk *server*, seringkali menerima paket *inbound* untuk *port* yang mempunyai *server* yang tidak dapat dipercaya. Dalam TCP, paket *inbound* dapat diterima tanpa menerima sambungan *inbound* menurut susunan bit ACK. Dengan UDP, tidak ada pilihan lain karena tidak ada ekuivalen pada bit ACK. Untungnya, sangat sedikit protokol yang digunakan untuk melintasi Internet berbasis UDP.

4.6 Memilih *Router* Penapisan Paket

Tersedia sejumlah *router* penapisan paket, ada yang bagus dan ada yang tidak. Hampir setiap *router* yang diberikan menunjang penapisan paket dengan beberapa bentuk. Selain itu paket penapisan paket tersedia untuk berbagai macam tujuan-umum UNIX dan PC *platform* yang biasanya digunakan sebagai *router*.

Cara memilih *router* penapisan paket yang terbaik bagi situs adalah memilih kemampuan yang paling penting yang seharusnya dimiliki *router* penapisan perusahaan. Yang harus ditentukan adalah kemampuan mana yang penting bagi perusahaan dan sistem penapisan yang dipilih minimal menawarkan kemampuan tersebut.

6.1 Memerlukan *Performance* Penapisan Paket Sesuai Kebutuhan

Banyak orang merasa khawatir mengenai *performance* penapisan paket. Padahal faktor *performance* berhubungan dengan kecepatan sambungan ke Internet, bukan kecepatan sistem penapisan paket. Pertanyaan yang benar mengenai sistem penapisan paket bukanlah "seberapa cepat sistem penapisan paket?", tetapi "apakah sistem penapisan paket tersebut cukup cepat sesuai dengan yang dibutuhkan?"

Sambungan Internet biasanya merupakan saluran 56-kb/s atau 1.544 Mb/s. Penapisan paket merupakan operasi per paket. Oleh karena itu, semakin kecil paket semakin banyak paket yang akan ditangani setiap detik, berarti semakin banyak keputusan penapisan yang harus dilakukan setiap detik. Paket IP terkecil (sebuah paket kosong yang hanya berisi IP *header* dan tidak ada data apapun) mempunyai panjang 20 byte (160 bit). Jadi kemampuan saluran 56 kb/s dapat membawa 350 paket per detik, dan kemampuan saluran 1.544 kb/s (misalnya, saluran T-1) dapat membawa hampir 9.650 paket per detik, seperti ditunjukkan dalam Tabel 5.

Tabel 5 Kemampun tipe sambungan dalam menangani penapisan paket/detik

Tipe Sambungan	Bit/Detik (kira-kira)	Paket/Detik (paket 20 byte)	Paket/Detik (paket 40 byte)
v.32bis modem	14.400	90	45
56-Kb/s *leased line*	56.000	350	175
T-1 *leased line*	1.544.000	9.650	4.825
Ethernet	3.000.000	18.750	9.375
Ethernet (teoritis)	10.000.000	62.500	31.250
T-3 *leased line*	45.000.000	281.250	140.625
FDDI	100.000.000	625.000	312.500

Dalam kenyataannya, paket IP kosong jarang ditemukan karena selalu terdapat sesuatu dalam data *segment* (misalnya, paket TCP, UDP, atau ICMP). Jenis paket yang melintasi *firewall* adalah paket TCP/IP karena sebagian besar layanan Internet berbasis TCP. Ukuran paket TCP/IP minimal (untuk sebuah paket yang hanya berisi IP *header* dan TCP *header* dan tidak ada data aktual) adalah 40 byte, yang memotong separuh nilai paket maksimum, sampai 175 paket per detik untuk

saluran 56 kb/s dan 4825 paket per detik untuk saluran 1.544 Mb/s. Kecepatan paket per detik ini merupakan kemampuan yang baik dari beberapa sistem penapisan paket, baik yang tersedia saat ini secara bebas maupun yang komersial di Internet.

Kecepatan seperti itu akan menjadi persoalan *firewall* yang bersifat internal bagi jaringan perusahaan. Beberapa *Firewall* perlu menjalankan kecepatan LAN, secara teoritis minimal 10 Mb/s dan ada kemungkinan lebih tinggi lagi. (*Firewall* dalam hal ini tidak praktis dalam jaringan gigabit-per-detik. Untungnya, dari perspektif *firewall*, jaringan ini jarang ditemukan).

Firewall dengan sambungan yang lebih dari dua memerlukan kecepatan yang lebih tinggi. Dengan dua sambungan, kecepatan maksimum yang diperlukan adalah kecepatan dari sambungan yang paling rendah. Dengan tiga sambungan, kecepatan yang diperlukan bisa meningkat. Sebagai contoh, jika sambungan Internet kedua diletakkan ke dalam *router* eksternal, berarti ada keperluan untuk menjalankan keduanya dalam kecepatan penuh, jika tidak, berati akan menjadi faktor pembatas. Jika dua jaringan internal diletakkan ke dalamnya, berarti perlu mencapai kecepatan LAN yang lebih tinggi untuk melakukan *route* antara kedua jaringan internal.

6.2 Dapat Berupa *Router* Tujuan Tunggal atau Komputer Tujuan Umum

Jangan mengharapkan suatu peralatan tunggal dapat berfungsi sebagai *router* penapisan paket dan juga dapat melakukan sesuatu yang bukan salah satu bagian dari *firewall*. (*User* bisa memiliki peralatan yang melakukan penapisan paket dan *proxying*, atau penapisan paket dan layanan *bastion host* yang dipilih, atau bahkan ketiganya). Yang harus diharapkan adalah menggunakan *router* penapisan paket. Namun ini tidak berarti bahwa *user* harus membeli *router* dengan tujuan tunggal. *User* dapat memilih menggunakan *router* dengan tujuan tunggal atau komputer dengan tujuan umum untuk *routing*. Apa untung ruginya masing-masing pilihan ini?

Jika terdapat sejumlah besar jaringan atau multi protokol, maka diperlukan *router* dengan tujuan tunggal. Paket *routing* untuk komputer dengan tujuan umum biasanya tidak mempunyai kecepatan atau fleksibilitas seperti *router* dengan tujuan

tunggal, juga ada kemungkinan diperlukan suatu mesin besar yang sulit untuk mengakomodasikan *interface board* yang diperlukan.

Sebaliknya, jika penapisan dilakukan pada mata rantai Internet tunggal, maka tidak perlu banyak hal yang harus dilakukan jika dibandingkan dengan *route* paket IP antara dua Ethernet. Hal ini akan berjalan dengan baik apabila dilakukan dalam kemampuan komputer berbasis 486, dan mesin ini bisa lebih murah dibandingkan dengan *router* tujuan tunggal. Paket penapisan dan *routing* tersedia pada MS-DOS dan sebagian besar tersedia pada UNIX.

Apapun peralatan yang digunakan untuk *router* penapisan, *firewalling* seharusnya merupakan tindakan yang dilakukan semua *router*. Sebagai contoh, jika mungkin, jangan gunakan satu peralatan untuk dijadikan sebagai *router* penapisan dan *backbone router* yang terikat bersama multi-jaringan internal yang terpisah. Sebaliknya, gunakan satu peralatan untuk diikat bersama jaringan internal dan peralatan *router* penapisan yang terpisah. Semakin kompleks *router* penapisan dan konfigurasinya, semakin besar kemungkinan berbuat salah dalam konfigurasi, yang bisa mendatangkan implikasi keamanan yang serius. Penapisan juga mempunyai dampak kecepatan yang cukup penting pada *router* dan dapat melambatkan *router* pada suatu titik dimana *performance* untuk jaringan internal sulit dicapai.

Sebagaian paket *firewall* yang komersial mengkombinasikan penapisan paket dengan *proxying* dalam satu mesin yang berlaku sebagai *router* bertujuan tunggal. Sebagian yang lain mengkombinasikan penapisan paket dengan *proxying* atau dengan layanan *bastion host* dalam satu komputer tujuan umum berkekuatan tinggi. Hal ini baik meskipun diperlukan peningkatan kecepatan. Jangan berharap bisa menggunakan mesin kecil untuk malakukan pekerjaan ini. Tergantung pada mesin apa yang tersedia, ini mungkin merupakan tawaran yang baik (yaitu membeli mesin besar tunggal sebagai ganti mesin *multiple* ukuran medium), atau mungkin menjadi tawaran yang buruk (yaitu membeli mesin besar tunggal sebagai ganti penambahan mesin kecil pada konfigurasi yang ada).

6.3 Aturan Spesifikasi Harus Sederhana

Tentukan aturan-aturan penapisan paket sesederhana mungkin. Carilah ciri-ciri ini dalam peralatan yang dipilih. Dari sudut pandang konseptual, penapisan paket terlalu rumit untuk dimulai, dan menjadi semakin rumit dalam perincian berbagai macam protokolnya. Kompleksitas dan komplikasi yang telah ada dalam sistem jangan ditambah lagi dengan sistem penapisan paket.

Tentukan aturan-aturan pada level abstraksi yang tinggi. Hindari implementasi penapisan paket yang melayani paket-paket dengan *array* bit yang tidak terstruktur dan tentukan aturan-aturan mengenai *offset* dan *state* bit khusus dalam paket *header*.

6.4 Harus Memenuhi Aturan yang Berdasarkan Kriteria *Header* atau *Meta-Packet.*

Tentukan aturan-aturan yang didasarkan pada informasi *header* atau *meta-packet* apapun yang ada pada paket. Informasi *Header* meliputi:

- sumber IP dan alamat tujuan,
- IP *option*,
- protokol, seperti TCP, UDP, atau ICMP,
- *port* sumber dan tujuan TCP atau UDP,
- tipe pesan ICMP, dan
- informasi sambungan awal (bit ACK) untuk paket-paket TCP dan informasi yang sama mengenai protokol lain yang ditapis.

Informasi *meta-packet* meliputi informasi apapun mengenai paket yang diketahui *router*, tetapi bukan informasi dalam *header*, misalnya mengenai paket yang keluar-masuk berasal di suatu *router interface*. Tentukan aturan-aturan berdasarkan pada kombinasi kriteria *header* dan *meta-packet* tersebut.

Sehubungan dengan berbagai alasan, banyak produk penapisan yang tidak memperbolehkan *user* melihat *port* sumber TCP atau UDP pada saat membuat keputusan paket penapisan; mereka hanya memperbolehkan *user* melihat *port* tujuan TCP atau UDP. Hal ini menyebabkan *user* tidak dapat menentukan jenis penapis

yang tertentu. Beberapa manufaktur menghapus *port* sumber TCP/UDP pada kriteria penapisan paket yang dianggap tidak berguna bagi penapis tersebut, atau sesuatu yang berguna, tetapi terlalu berbahaya bagi pelanggan untuk dimengerti (karena seperti diketahui bahwa informasi *port* sumber tidak dapat dipercaya sepenuhnya).

Perhatikan contoh penapis yang didasarkan pada *port* sumber yang berguna, yaitu akses untuk *Archie server*. Akses *Archie* mempunyai beberapa masalah gangguan. Masalah terbesar adalah protokol *Archie* berbasis UDP, dan pada UDP *firewall* Internet biasanya diblok bersama (kecuali untuk *peepholes* kecil seperti layanan DNS, NTP, dan *Archie*). Alasan pemblokiran biasanya digunakan untuk mencegah orang luar menyerang layanan berbasis RPC seperti NIS/YP dan NFS. Karena layanan berbasis RPC tidak bekerja pada bilangan *port* yang tetap, maka akses ke *port* layanan berbasis RPC tidak dapat diblok secara sederhana (karena *user* tidak tahu siapa mereka). Tidak seperti TCP, UDP tidak mempunyai bit ACK, sehingga penapisan sambungan awal tidak dapat dilakukan. Sistem penapisan paket tidak dapat memberitahu dengan hanya memeriksa paket yang masuk, apakah paket merupakan tanggapan dari *server* eksternal ke *client* internal, atau perintah dari *client* eksternal (kemungkinan merupakan program serangan) ke *server* internal (seperti NIS/YP atau NFS). Dengan semua hambatan tersebut, bagaimana selanjutnya dalam menentukan aturan untuk *Archie*?

Salah satu dari karakteristik *Archie* adalah bahwa kurang lebih hanya ada 20 *Archie server* di dunia. Untuk alasan *performance*, *user* biasanya hanya menginginkan sedikit pembicaraan dari *server* yang dekat dengan situs dalam topologi jaringan (sebagai contoh, hanya terdapat lima *server* di Amerika Serikat). Jika ingin membuat asumsi tertentu mengenai sekuritas *Archie server*, akses *Archie* untuk *user* dapat mengijinkan paket keluar ke *server Archie*, dan mengijinkan paket kembali dari *server Archie* (dengan menggunakan penapisan *port* sumber).

Namun hal ini sangat riskan. Jika seseorang memalsukan paket dari *port* 1525 pada *server Archie* ke *server* NFS internal, mereka akan berhasil mendapatkan respons. Untuk melakukan hal itu, mereka menduga paket-paket tersebut akan

diijinkan, mendapat alamat *host* internal dan *port* saat *server* NFS dalam posisi hidup, paket dipalsukan, dan mereka akan menerima jawaban. Paket NFS biasanya berupa fragmen, dan biasanya dilewatkan melalui penapisan paket, ada kemungkinan mereka berhasil mendapatkan beberapa data sekalipun penapisan paket cukup lihai dalam memblok NFS dengan memeriksa informasi protokol level aplikasi. Jika yang dihadapi adalah jenis penyerang ini, maka tipe penapisan ini belum cukup aman.

Tabel 6 menunjukkan aturan sederhana yang mengijinkan *client* internal berinteraksi dengan *Archie server*, dan tidak mengijinkan sesuatu yang lain. Diasumsikan bahwa untuk setiap paket, aturan-aturan dievaluasi dengan urutan yang telah didaftar sampai keseimbangan ditemukan.

Tabel 6 Aturan yang mengijinkan *client* internal berinteraksi dengan Archie *server*

Aturan	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Sumber	*Port* Tujuan	Aksi
A	*Out*	Internal	*Archie Server*	UDP	>1023	1525	Mengijinkan
B	*In*	*Archie Server*	Internal	UDP	1525	>1023	Mengijinkan
C	Salah satu	Apa saja	Apa saja	Apa saja	Apa saja	Apa saja	Menolak

- Aturan A membantu *client* berbicara ke *Archie server*.
- Aturan B mengijinkan *Archie server* berbicara kembali ke *client*.
- Aturan C adalah aturan *default* yang memblok segala sesuatu yang lain.

Aturan ini hanya dapat dilakukan apabila sistem penapisan mengijinkan penapisan berdasarkan pada *port* sumber.

6.5 Harus Mengaplikasikan Aturan dalam *Order* Khusus

Penapisan paket harus mengaplikasikan aturan-aturan yang telah ditentukan baginya dalam *order* yang dapat diramalkan. Sejauh ini, *order* yang paling sederhana adalah *order*, dengan *user* sebagai orang yang membuat konfigurasi *router*, menentukan aturan-aturan. Untungnya, ada beberapa produk (sebagai ganti dalam

mengaplikasikan aturan dalam *order* yang telah ditetapkan) mencoba melakukan *reorder* dan menggabungkan aturan-aturan untuk mencapai efisiensi yang lebih besar dalam aplikasi aturan-aturan tersebut, tetapi hal ini menimbulkan beberapa masalah.

- *Reorder* aturan menyulitkan *user* dalam menunjukkan apa yang sedang terjadi, dan apa yang sedang dilakukan *router* dengan susunan instruksi penapisan khusus. Mengkonfigurasikan sistem penapisan paket sudah cukup banyak menghadapi kesulitan, tanpa ditambah lagi dengan penggunaan *vendor* yang mempunyai komplikasi dalam penggabungan dan *reordering* susunan aturan.
- Jika terdapat *quirk* dan *bug* dalam penggabungan atau *reordering* sesunan aturan (dan seringkali karena terdapat sesuatu yang sangat sulit diuji bagi *vendor*), ia tidak mungkin dapat menunjukkan apa yang sedang dilakukan sistem dengan penapis yang ada.
- Yang paling penting, *reordering* aturan dapat menghancurkan aturan yang sudah bekerja dengan baik jika ia belum di-*reorder*.

Perhatikan contoh ini, andaikan *user* berada dalam suatu perusahaan, bekerja atas dasar proyek khusus dengan universitas lokal. Jaringan kelas B perusahaan adalah 172.16 (misalnya, alamat IP 172.16.0.0 sampai 172.16.255.255). Universitas mempunyai jaringan kelas A sebanyak 10 (misalnya, alamat IP mereka adalah 10.0.0.0 sampai 10.255.255.255).

Untuk tujuan proyek ini, jaringan langsung dihubungkan dengan universitas menggunakan *router* penapisan paket. Semua akses Internet diijinkan melalui hubungan ini (akses Internet sebaiknya melewati *firewall* Internet). Proyek khusus dengan universitas menggunakan *subnet* 172.16 jaringan kelas B (yaitu alamat IP 172.16.6.0 sampai 172.16.6.255). Semua *subnet* di universitas dikehendaki untuk mampu mengakses *subnet* proyek ini. Terdapat seperdelapan bit *subnet* di universitas, *subnet* 10.1.99, yang mempunyai aktivitas perlawanan didalamnya; beri jaminan bahwa *subnet* ini hanya dapat mencapai *subnet* proyek saja.

Untuk memenuhi keperluan di atas, buat spesifikasi tiga aturan penapisan paket yang didaftar di bawah ini (dalam contoh ini, hanya menetapkan aturan untuk lalulintas yang masuk ke dalam situs; jangan lupa bahwa aturan untuk lalulintas yang keluar dari situs juga perlu ditetapkan).

- Aturan A mengijinkan universitas mencapai *subnet* proyek.
- Aturan B mengunci *subnet* yang berlawanan di universitas dari segala sesuatu yang ada pada jaringan.
- Aturan C tidak mengijinkan akses Internet pada jaringan.

Sekarang mari lihat apa yang terjadi dalam beberapa kasus yang berbeda, tergantung pada bagaimana aturan-aturan tersebut diaplikasikan secara tepat.

Jika aturan-aturan diaplikasikan dalam *order* ABC

Jika aturan-aturan diaplikasikasikan dalam *order* ABC, yaitu *order* yang sama yang ditetapkan oleh *user*, maka Tabel 7 menunjukkan apa yang akan terjadi pada berbagai macam paket contoh.

Tabel 7 Paket contoh apabila aturan diaplikasikan dalam *order* ABC

Paket	Alamat Sumber	Alamat Tujuan	Aksi yang Diinginkan	Aksi Aktual (menurut Aturan)
1	10.1.99.1	172.16.1.1	Menolak	Menolak (B)
2	10.1.99.1	172.16.6.1	Mengijinkan	Mengijinkan(A)
3	10.1.1.1	172.16.1.1	Mengijinkan	Mengijinkan (A)
4	10.1.1.1	172.16.6.1	Menolak	Menolak (C)
5	192.168.3.4	172.16.1.1	Menolak	Menolak (C)
6	192.168.3.4	172.16.6.1	Menolak	Menolak (C)

- Paket 1 berasal dari mesin di universitas pada *subnet* lawan ke mesin sembarang pada jaringan *user* (bukan pada *subnet* proyek); yang diinginkan adalah “menolak”, dan ini dilakukan menurut aturan B.
- Paket 2 berasal dari mesin di universitas pada *subnet* lawan ke mesin pada *subnet* proyek; yang diinginkan adalah “mengijinkan”, dan ini dilakukan menurut aturan A.

- Paket 3 berasal dari sembarang mesin di universitas ke mesin pada *subnet* proyek; yang diinginkan adalah "mengijinkan", dan ini dilakukan menurut aturan A.
- Paket 4 berasal dari sembarang mesin di universitas ke salah satu mesin non-proyek; yang dinginkan adalah "menolak", dan ini dilakukan menurut aturan C.
- Paket 5 berasal dari sembarang mesin pada Internet ke salah satu mesin non-proyek; yang diinginkan adalah "menolak", dan ini dilakukan menurut aturan C.
- Paket 6 berasal dari sembarang mesin pada Internet ke salah satu mesin proyek; yang diinginkan adalah "menolak", dan ini dilakukan menurut aturan C.

Jadi, jika aturan-aturan diaplikasikan dalam *order* ABC, maka mereka melaksanakan apa yang *user* inginkan.

Jika aturan-aturan diaplikasikan dalam *order* BAC

Apa yang akan terjadi jika *router reorder* aturan-aturan menurut jumlah bit yang penting pada alamat sumber, sehingga aturan-aturan yang lebih khusus yang akan diaplikasikan terlebih dahulu? Dengan kata lain, aturan-aturan mengaplikasikan alamat sumber IP yang lebih khusus (misalnya aturan-aturan mengaplikasikan alamat sumber yang mempunyai *range* yang lebih kecil) akan diaplikasikan sebelum aturan-aturan mengaplikasikan alamat sumber IP yang kurang khusus? Dalam kasus ini, aturan-aturan akan diaplikasikan dalam *order* BAC.

Tabel 8 Aturan yang akan diaplikasikan dalam *order* BAC

Aturan	Alamat Sumber	Alamat Tujuan	Aksi
A	10.1.99.0/24	172.16.0.0/16	Menolak
B	10.0.0.0/8	172.16.6.0/24	Mengijinkan
C	Apa saja	Apa saja	Menolak

Terdapat enam paket contoh, dengan hasil baru apabila aturan-aturan diaplikasikan dalam *order* BAC; tabel berikut ini menunjukkan bagaimana aksinya berbeda dari kasus sebelumnya (aturan-aturan diaplikasikan dalam *order* yang ditetapkan oleh *user*).

Tabel 9 Paket contoh apabila aturan diaplikasikan dalam *order* BAC

Paket	Alamat Sumber	Alamat Tujuan	Aksi yang Diinginkan	Aksi Aktual (menurut Aturan)
1	10.1.99.1	172.16.1.1	Menolak	Menolak (B)
2	10.1.99.1	172.16.6.1	Mengijinkan	Menolak (B)
3	10.1.1.1	172.16.1.1	Mengijinkan	Mengijinkan (A)
4	10.1.1.1	172.16.6.1	Menolak	Menolak (C)
5	192.168.3.4	172.16.1.1	Menolak	Menolak (C)
6	192.168.3.4	172.16.6.1	Menolak	Menolak (C)

Jika aturan-aturan diaplikasikan dalam *order* BAC, maka paket 2, yang seharusnya diijinkan, ditolak dengan cara yang tidak tepat oleh aturan B. Sekarang, menolak sesuatu seharusnya diijinkan akan lebih aman dibandingkan dengan mengijinkan sesuatu yang seharusnya ditolak, tetapi akan lebih baik lagi jika sistem penapisan melakukan apa yang ingin dia lakukan.

User dapat membuat konstruksi contoh yang sama untuk sistem dengan aturan-aturan *reorder* berdasarkan pada jumlah bit yang penting dalam alamat tujuan, yang merupakan kriteria *reordering* lain yang sangat populer.

Aturan B sebenarnya tidak diperlukan

Perhatikan contoh ini baik-baik, yaitu mengenai *subnet* musuh yang menjadi alasan bagi aturan B, pada dasarnya terlalu berlebihan dan tidak diperlukan. Aturan B dimaksudkan untuk membatasi *subnet* musuh dalam mengakses *subnet* proyek. Sebenarnya aturan A sudah membatasi seluruh universitas, termasuk *subnet* musuh, dalam mengakses *subnet* proyek. Jika aturan B dihilangkan, kemudian aturan-aturan akan diaplikasikan dalam *order* AC tanpa peduli apakah sistem *reorder* berdasarkan pada jumlah bit penting alamat sumber IP atau tidak. Tabel 10 menunjukkan apa yang terjadi dalam kasus ini.

Tabel 10 Aturan B dihilangkan dalam *order* BAC

Aturan	Alamat Sumber	Alamat Tujuan	Aksi
A	10.0.0.0/8	172.16.6.0/24	Mengijinkan
C	Apa saja	Apa saja	Menolak

Tabel 11 Paket contoh apabila aturan diaplikasikan dalam *order* AC

Paket	Alamat Sumber	Alamat Tujuan	Aksi yang Diinginkan	Aksi Aktual (menurut Aturan)
1	10.1.99.1	172.16.1.1	Menolak	Menolak (C)
2	10.1.99.1	172.16.6.1	Mengijinkan	Mengijinkan (A)
3	10.1.1.1	172.16.1.1	Mengijinkan	Mengijinkan (A)
4	10.1.1.1	172.16.6.1	Menolak	Menolak (C)
5	192.168.3.4	172.16.1.1	Menolak	Menolak (C)
6	192.168.3.4	172.16.6.1	Menolak	Menolak (C)

Aturan penapisan paket bersifat *tricky*

Aturan penapisan yang benar harus bersifat *tricky*, yaitu rumit dan penuh akal. Contoh di atas mengambil situasi yang sangat sederhana, dan masih terdapat *error* dalam pengelolaan susunan aturan. Susunan aturan yang nyata lebih kompleks dan sering berisi puluhan atau ratusan aturan. Memperhatikan implikasi dan interaksi atas semua aturan ini merupakan hal yang mustahil, kecuali ditetapkan dalam *order* khusus. Jadi, panggilah "*help*" dari *router*, dalam bentuk susunan aturan *reordering*.

6.6 Aturan Harus Terpisah untuk Paket yang Masuk dan yang Keluar pada Basis Per-*Interface*

Untuk mendapatkan fleksibilitas, kapabilitas, dan *performance* yang maksimum, susunan aturan untuk paket yang keluar dan paket yang masuk pada setiap *interface* harus dipisah. Dalam bagian ini diberikan contoh yang menunjukkan persoalan yang dapat dijalankan dengan *router* tetapi tidak fleksibel.

Keterbatasan dalam pembagian beberapa sistem penapisan paket adalah yang membiarkan paket diuji hanya pada saat mereka meninggalkan sistem. Keterbatasan ini membawa tiga persoalan berikut.

- Sistem selalu meletakkan penapisnya sendiri di luar.
- Mendeteksi paket palsu merupakan pekerjaan yang sulit dan hampir mustahil.
- Sulit mengkonfigurasikan beberapa sistem jika mereka lebih dari dua *interface*.

Pada persoalan pertama; jika *router* hanya membolehkan *user* melihat paket yang keluar saja, sehingga paket yang berhubungan langsung dengan *router* itu

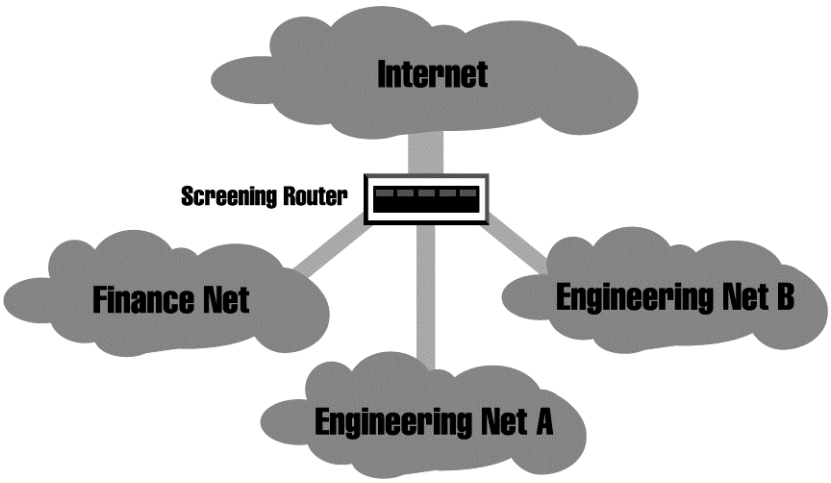
sendiri tidak pernah tunduk terhadap penapisan paket. Hasilnya adalah penapis tidak pernah melindungi *router* itu sendiri. Sebenarnya hal ini bukan merupakan persoalan yang serius, karena terdapat berbagai macam layanan pada *router* yang dapat diserang, terdapat pula cara lain yang bisa melindungi mereka. Telnet merupakan contoh layanan yang dapat diserang dengan cara ini, namun *user* dapat mengelilingi permasalahan *routing* ini dengan memutuskan *server* Telnet, atau dengan mengontrol dari mana ia akan menerima sambungan yang masuk. SNMP merupakan contoh lain yang umum tersedia dan merupakan layanan yang mudah diserang.

Pada persoalan kedua; jika *router* hanya dapat menapis paket yang akan keluar, mendeteksi paket palsu yang dimasukkan dari luar merupakan sesuatu hal yang sulit atau hampir mustahil (yaitu, paket yang berasal dari luar tetapi mengklaim bahwa ia mempunyai alamat sumber internal), seperti yang diilustrasikan pada Gambar 1. Mendeteksi pemalsuan lebih mudah dilakukan pada saat paket memasukkan *router* pada *inbound interface*. Mendeteksi pemalsuan pada *outbound interface* sangat komplikasi karena paket dihasilkan oleh *router* itu sendiri (yang mempunyai alamat sumber internal jika *router* itu sendiri mempunyai alamat internal) dan oleh paket internal yang sah yang keliru menghubungi *router* (paket yang seharusnya dikirimkan langsung dari sumber internal ke tujuan internal, tetapi salah dikirimkan ke *router* penapisan).

Pada persoalan ketiga berhubungan dengan penapisan pada *outbound*; penapisan paket sangat sulit dikonfigurasikan apabila ia mempunyai *interface* lebih dari dua. Jika ia hanya mempunyai dua *interface* saja, maka penapisan *outbound* pada setiap *interface* bukan merupakan persoalan besar. Karena hanya ada dua jalan melalui *router* (dari *interface* pertama ke *interface* kedua, dan sebaliknya). Paket yang menuju satu jalan dapat ditapis sebagai paket yang keluar pada satu *interface*, pada saat paket menuju ke jalan yang satunya lagi, ia dapat ditapis sebagai paket yang keluar pada *interface* yang satunya lagi. Misalnya, *router* mempunyai empat *interface*, yaitu satu untuk sambungan situs Internet, satu untuk jaringan keuangan, dan dua untuk jaringan teknik. Berarti harus ada penambahan kebijakan berikut ini.

- Dua jaringan teknik dapat berkomunikasi satu sama lain tanpa batasan.
- Dua jaringan teknik dan Internet dapat saling berkomunikasi dengan batasan tertentu.
- Dua jaringan teknik dan jaringan keuangan dapat saling berkomunikasi dengan batasan tertentu, dan batasan ini berbeda dengan batasan antara jaringan teknik dan Internet.
- Jaringan keuangan tidak berkomunikasi dengan Internet dalam keadaan apapun.

Gambar 9 mengilustrasikan keadaan ini.



Gambar 9 Batasan penapisan paket pada *interface* yang berbeda.

Terdapat 12 jalan melalui *router*, dari masing-masing empat *interface* ke masing-masing tiga *interface* lain (pada umumnya, terdapat N*N-1 jalan melalui sebuah N *interface router*). Dengan sistem penapisan *outbound*, penapisan berikut ini harus diciptakan pada setiap *interface*.

- Teknik A, meliputi penapis Internet, penapis keuangan, penapis teknik B.
- Teknik B, meliputi penapis Internet, penapis keuangan, penapis teknik A.
- Keuangan, meliputi penapis Internet, penapis teknik A, penapis teknik B.
- Internet, meliputi penapis teknik A, penapis teknik B, penapis keuangan.

Penggabungan penapis *mulltiple* dalam *interface* tunggal seperti ini dapat menjadi sangat rumit. Tergantung pada kompleksitas penapis dan fleksibilitas sistem penapisan, dan kemungkinan akan terdapat sejumlah situasi yang mustahil.

Permasalahan yang lebih rumit adalah terdapat beberapa *setup* yang memasukkan penapisan paket yang berlebihan di antara dua jaringan teknik (yang menghasilkan permasalahan *performance* yang cukup penting). Dengan *setup* ini, *router* harus menguji semua paket yang mengalir antara dua jaringan teknik tersebut, meskipun paket tidak pernah diputuskan untuk didrop.

Sekarang, mari lihat skenario yang sama, yaitu mengandaikan sistem penapisan paket mempunyai penapis *inbound* dan penapis *outbound*. Yang harus dilakukan dalam kasus ini adalah sebagai berikut.

- Semua penapis dihubungan ke Internet (baik yang diaplikasikan ke jaringan teknik maupun ke jaringan keuangan) pada *interface* Internet.
- Semua penapis dihubungkan ke jaringan keuangan (baik yang diaplikasikan ke jaringan teknik maupun ke Internet) pada *interface* keuangan.
- Tidak ada penapis sama sekali pada *interface* teknik (hal ini memaksimumkan *performance* bagi lalulintas antara jaringan teknik, karena ia tidak akan dilewatkan melalui penapis apapun).

6.7 Mampu Melakukan *Logging* Paket Penerimaan dan Pendropan

Pastikan bahwa *router* penapisan paket memberikan pilihan untuk melakukan *logging* semua paket yang didrop. Setiap paket yang diblok oleh aturan penapisan paket harus diketahui. Aturan ini mencerminkan kebijakan sekuritas dan *user* harus mengetahui apabila ada seseorang yang mencoba melanggarnya. Cara paling sederhana untuk mengetahui seseorang yang melanggar adalah melalui *logging*.

Logging harus dilakukan terhadap penerimaan paket tertentu saja. Misalnya, *lagging* terhadap awal setiap sambungan TCP. *Logging* semua penerimaan paket yang memuat terlalu banyak data dalam operasi normalnya, yang mungkin

menyebabkan *debugging* atau serangan. Meskipun *logging* sudah dilakukan terhadap paket tujuan, namun *logging* tidak akan bekerja apabila *host* tujuan dikompromikan, sehingga paket yang melalui penapisan paket tidak ditunjukkan apabila ia tidak mempunyai tujuan yang benar. Paket-paket ini cukup menarik karena mereka dapat diselidiki oleh penyerang. Tanpa informasi dari *router*, *user* tidak akan mendapat gambaran yang jelas mengenai aktivitas penyerang.

Logging harus fleksibel; penapisan paket harus memberikan kemampuan logging melalui *syslog* dan bisa disimpan ke dalam *file* lokal.

6.8 Mempunyai Kemampuan Pengesahan dan Pengujian yang Bagus

Salah satu bagian penting dalam menciptakan *firewall* adalah bisa menyakinkan diri sendiri (dan yang lain dalam perusahaan) bahwa pekerjaan sudah diselesaikan dengan baik, dan tidak melupakan sesuatu. Untuk itu, yang harus dikerjakan adalah melakukan pengujian dan pengesahan konfigurasi. Sebagian besar paket penapisan saat ini sedikit atau sama sekali tidak mempunyai kemampuan pengujian dan pengesahan.

Pengujian dan pengesahan dibagi ke dalam dua pertanyaan yang saling berhubungan:

- Sudahkah perusahaan memberitahukan secara khusus kepada *router* apa yang perusahaan ingin dia kerjakan?
- Apakah *router* melakukan apa yang perusahaan perintahkan untuk dikerjakannya?

Sayangnya, dengan banyaknya produk yang tersedia saat ini di pasaran, kedua pertanyaan tersebut di atas sulit dijawab.

7. Menggabungkan Semua dalam Dunia Nyata

Bagian ini menjelaskan penggabungan semua yang telah dibahas di atas dalam dunia nyata. Bagian ini dirancang untuk menunjukkan proses pengembangan seperangkat penapis. Bukan menunjukkan seperangkap penapis lengkap pada situs

pun. Setiap situs berbeda, dan *user* akan menjumpai kesulitan dengan penapisan paket tersebut apabila *user* tidak memahami perincian dan implikasi penggunaannya dalam lingkungan khusus. Diinginkan agar orang-orang memperhatikan dan memahami secara hati-hati apa yang mereka lakukan, tidak sembarangan menyalin sesuatu tanpa perhatian yang teliti. Dalam beberapa kasus, solusi lengkap bagi situs adalah pertimbangan mengenai penapisan paket, *proxying*, dan masalah konfigurasi.

Contoh yang sederhana, misalnya mengijinkan *inbound* dan *outbound* SMTP (sehingga *e-mail* dapat diterima dan dikirimkan) dan tidak untuk yang lain. Mulailah dengan aturan pada Tabel 12.

Tabel 12 Aturan dalam mengijinkan *inbound* dan *outbound* SMTP

Aturan	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Tujuan	Aksi
A	Masuk	Eksternal	Internal	TCP	25	Mengijinkan
B	Keluar	Internal	Eksternal	TCP	>1023	Mengijinkan
C	Keluar	Internal	Eksternal	TCP	25	Mengijinkan
D	Masuk	Eksternal	Internal	TCP	>1023	Mengijinkan
E	Sala satu	Apa saja	Apa saja	Apa saja	Apa saja	Menolak

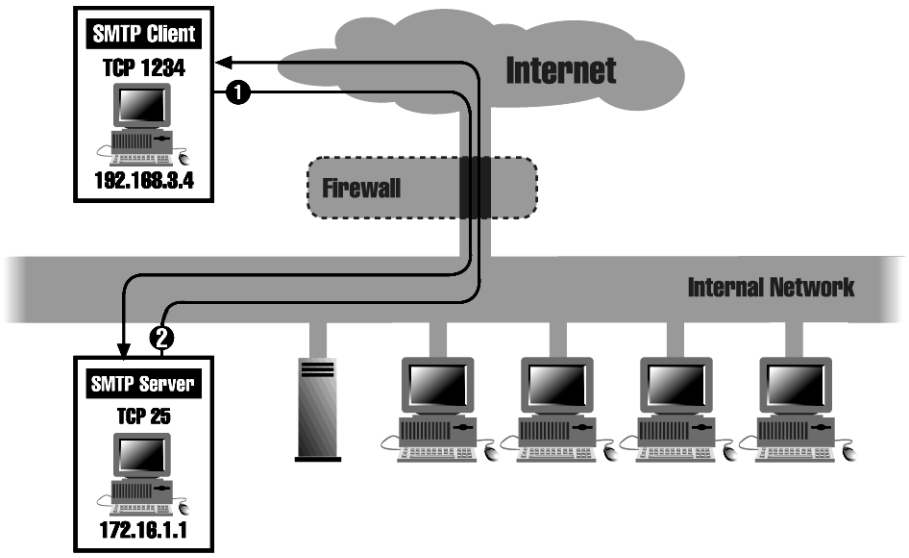
Diasumsikan dalam contoh ini bahwa untuk setiap paket, sistem penapisan melihat aturan-aturan secara berurutan. Langkah dimulai pada bagian atas sampai ia menemukan aturan yang cocok dengan paket, baru kemudian mengambil tindakan khusus.

- Aturan A dan B mengijinkan sambungan *inbound* SMTP (*e-mail* yang masuk).
- Aturan C dan D mengijinkan sambungan *inbound* SMTP (*e-mail* yang keluar).
- Aturan E adalah aturan *default* yang diaplikasikan jika kesemuanya gagal.

Sekarang perhatikan beberapa paket contoh pada Tabel 13 untuk melihat apa yang akan terjadi. Seandainya *host* mempunyai alamat IP 172.16.1.1, dan seseorang berusaha mengirim *e-mail* kepada perusahaan dari *host* yang jauh dengan alamat IP 192.168.3.4. Seandainya pengirim, yaitu *client* SMTP menggunakan *port* 1234 untuk berbicara kepada *server* SMTP perusahaan pada *port* 25. (*Server* SMTP selalu diasumsikan berada pada *port* 25). Dan Gambar 10 menunjukkan kasus ini.

Tabel 13 Paket contoh dengan *port* tujuan 25 dan 1234

Paket	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Tujuan	Aksi (Aturan)
1	Masuk	192.168.3.4	172.16.1.1	TCP	25	Mengijinkan (A)
2	Keluar	172.16.1.1	192.168.3.4	TCP	1234	Mengijinkan (B)



Gambar 10 Penapisan paket: *inbound* SMTP (paket contoh 1 dan 2).

Dalam kasus ini, aturan penapisan paket mengijinkan *e-mail* yang masuk:

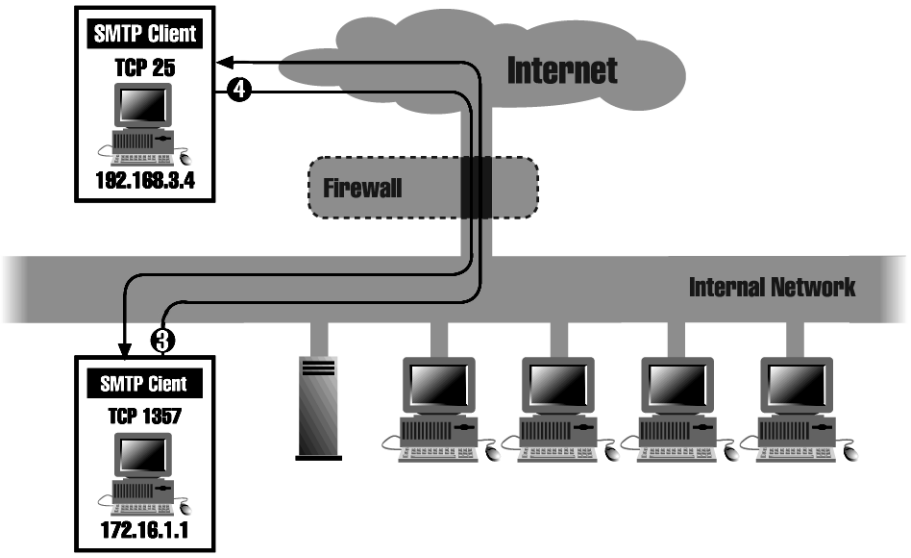
- Aturan A mengijinkan paket yang masuk dari *client* SMTP pengirim ke *server* SMTP perusahaan.
- Aturan B mengijinkan respons *server* perusahaan kembali ke *client* pengirim (ditunjukkan oleh paket nomor dua diatas).

Bagaimana dengan *e-mail* yang keluar dari perusahaan ke mereka? Seandainya *client* SMTP perusahaan menggunakan *port* 1357 untuk berbicara dengan *server* SMTP mereka.

Tabel 14 Paket contoh dengan *port* tujuan 25 dan 1357

Paket	Jurusan	Alamat Sumber	Alamat tujuan	Protokol	*Port* Tujuan	Aksi (Aturan)
3	Keluar	172.16.1.1	192.168.3.4	TCP	25	Mengijinkan (C)
4	Masuk	192.168.3.4	172.16.1.1	TCP	1357	Mengijinkan (D)

Gambar 11 menunjukkan kasus ini.



Gambar 11 Penapisan paket: *outbound* SMTP (paket contoh 3 dan 4).

Dalam kasus ini, aturan penapisan paket mengijinkan *e-mail* yang keluar.

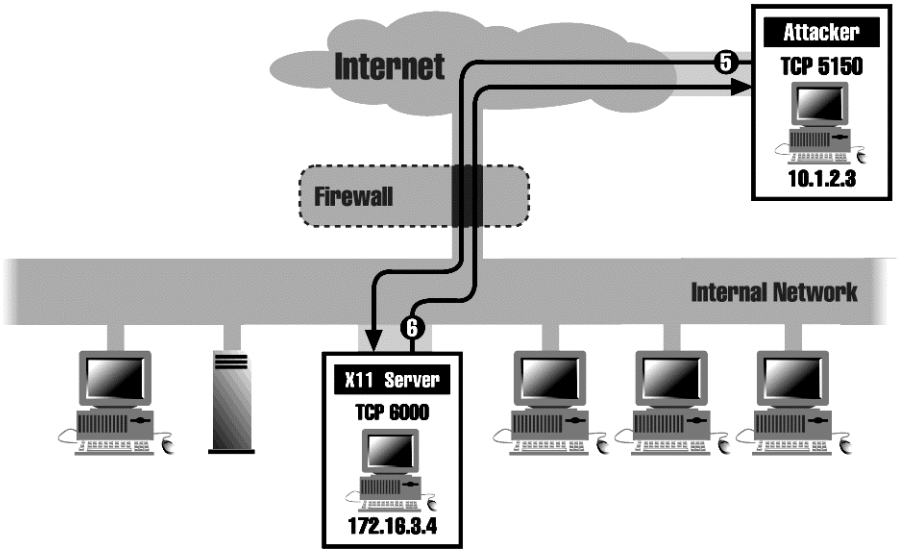
- Aturan C mengijinkan paket yang keluar dari *client* SMTP perusahaan ke *server* SMTP mereka (ditunjukkan oleh paket nomor 3 di atas).
- Aturan D mengijinkan respons *server* mereka kembali pada *client* perusahaan (ditunjukkan oleh paket nomor 4).

Apakah yang terjadi jika seseorang di dunia luar (misalnya seorang yang berada pada kedudukan *host* 10.1.2.3) berusaha membuka sambungan *port* 5150 dalam tujuannya ke *server* XII pada *port* 6000 pada salah satu sistem internal perusahaan (misalnya 172.16.3.4) dalam *order* untuk melakukan suatu serangan.

Tabel 15 Paket contoh dengan *port* tujuan 6000 dan 5150

Paket	Jurusan	Alamat Sumber	Alamat tujuan	Protokol	*Port* Tujuan	Aksi (Aturan)
5	Masuk	10.1.2.3	172.16.3.4	TCP	6000	Mengijinkan (D)
6	Keluar	172.16.3.4	10.1.2.3	TCP	5150	Mengijinkan (B)

Gambar 12 menunjukkan kasus ini.



Gambar 12 Penapisan paket: *inbound* SMTP (paket contoh 5 dan 6).

Sesungguhnya, aturan tersebut mengijinkan sambungan apapun untuk dilaksanakan selama akhir sambungan menggunakan *port* diatas 1023. Mengapa?

- Aturan A dan B bersama-sama melakukan apa yang diinginkan untuk mengijinkan sambungan *inbound* SMTP.
- Aturan C dan D bersama-sama melakukan apa yang diinginkan untuk mengijinkan sambungan *outbound* SMTP.
- Tetapi aturan B dan D bersama-sama mengakhiri untuk mengijinkan semua sambungan yang kedua tujuannya menggunakan *port* diatas 1023, dan ini sudah barang tentu bukan apa yang dimaksudkan.

Terdapat kemungkinan tanggapan *server* yang rentan terhadap serangan pada *port* diatas 1023 dalam situs perusahaan. Contoh XII (*port* 6000), *OpenWindows* (*port* 2000), *database* (*Sybase*, *Oracle*, *Informix*, dan *database* lain pada umumnya menggunakan *port* situs diatas 1023), dan seterusnya. Inilah mengapa perlu dipertimbangkan aturan secara keseluruhan sebagai ganti asumsi bahwa jika setiap aturan atau kelompok aturan sudah beres, maka keseluruhan ketentuannya juga beres.

Apa yang bisa dilakukan? Disini ada lima aturan dasar yang sama dengan *port* sumber yang bisa ditambahkan sebagai kriteria, seperti ditunjukkan oleh Tabel 16, dan Tabel 17 menunjukkan paket contohnya.

Tabel 16 Aturan dasar untuk *port* diatas 1023

Aturan	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Sumber	*Port* Tujuan	Aksi
A	Masuk	Eksternal	Internal	TCP	>1023	25	Mengijinkan
B	Keluar	Internal	Eksternal	TCP	25	>1023	Mengijinkan
C	Keluar	Internal	Eksternal	TCP	>1023	25	Mengijinkan
D	Masuk	Eksternal	Internal	TCP	25	>1023	Mengijinkan
E	Salah satu	Apa saja	Apa saja	Apa saja	Apa saja	Apa saja	Menolak

Tabel 17 Paket contoh dengan aturan dasar *port* diatas 1023

Paket	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Sumber	*Port* tujuan	Aksi (Aturan)
1	Masuk	192.168.3.4	172.16.1.1	TCP	1234	25	Mengijinkan (A)
2	Keluar	172.16.1.1	192.168.3.4	TCP	25	1234	Mengijinkan (B)
3	Keluar	172.16.1.1	192.168.3.4	TCP	1357	25	Mengijinkan (C)
4	Masuk	192.168.3.4	172.16.1.1	TCP	25	1357	Mengijinkan (D)
5	Masuk	10.1.2.3	172.16.3.4	TCP	5150	6000	Menolak (E)
6	Keluar	172.16.3.4	10.1.2.3	TCP	6000	5150	Menolak (E)

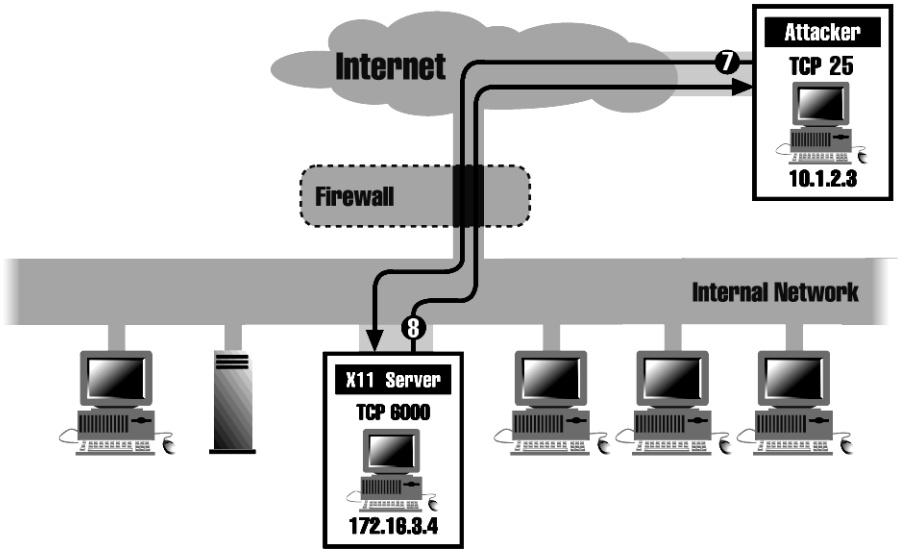
Sebagaimana dapat dilihat, ketika *port* sumber dipandang sebagai kriteria, persoalan paket (nomor 5 dan 6, yang ditunjukkan oleh serangan pada salah satu *server* X11) tidak ditemukan lagi dengan paket yang diijinkan (aturan A sampai D). Masalah paket diakhiri dengan ditolak oleh aturan *default*.

Apa yang harus dilakukan untuk menangani penyerang yang lebih cermat? Bagaimana jika penyerang menggunakan *port* 25 sebagai *port client* bagi tujuannya

(dia dapat melakukan hal ini dengan mematikan *server* SMTP pada mesin yang dia kontrol dan menggunakan *port*-nya atau dengan menyerang dari mesin yang tidak pernah menggunakan *server* SMTP pada tempat yang pertama, seperti PC), dan kemudian ia berusaha membuka sambungan pada *server* X11 perusahaan. Paket-paket dapat dilihat pada Tabel 18, dan Gambar 13 menunjukkan kasus ini.

Tabel 18 Paket contoh dengan *port* 25 dan 6000

Paket	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Sumber	*Port* tujuan	Aksi (Aturan)
7	Masuk	10.1.2.3	172.16.3.4	TCP	25	6000	Mengijinkan (D)
8	Keluar	172.16.3.4	10.1.2.3	TCP	6000	25	Mengijinkan (C)



Gambar 13 Penapisan paket: *inbound* SMTP (paket contoh 7 dan 8).

Seperti yang dapat dilihat, paket-paket akan diijinkan dan bisa menimbulkan serangan (sekuritas X11 menjadi selemah mungkin). Solusinya adalah menggunakan bit ACK sebagai kriteria tambahan dalam penapisan.

Tabel 19 Aturan dengan menggunakan bit ACK sebagai kriteria tambahan dalam panapisan

Aturan	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Sumber	*Port* Tujuan	Susunan ACK	Aksi
A	Masuk	Eksternal	Internal	TCP	>1023	25	Apa saja	Mengijinkan
B	Keluar	Internal	Eksternal	TCP	25	>1023	*Yes*	Mengijinkan
C	Keluar	Internal	Eksternal	TCP	>1023	25	Apa saja	Mengijinkan
D	Masuk	Eksternal	Internal	TCP	25	>1023	*Yes*	Mengijinkan
E	Salah satu	Apa saja	Apa saja	Apa saja	Apa saja	Apa saja	Apa saja	Menolak

Sekarang, paket 7 (penyerang berusaha membuka sambungan ke *client* X11 perusahaan) akan digagalkan seperti paket contoh pada Tabel 20.

Tabel 20 Paket contoh saat penyerang berusaha membuka sambungan ke *client* X11

Paket	Jurusan	Alamat Sumber	Alamat Tujuan	Protokol	*Port* Sumber	*Port* Tujuan	Susunan ACK	Aksi
7	Masuk	10.1.2.3	172.16.3.4	TCP	25	6000	*No*	Menolak (E)

Satu-satunya perbedaan dalam perangkat aturan ini adalah dalam aturan B dan aturan D. Tentu saja, aturan D adalah yang paling penting, karena ia mengendalikan sambungan yang baru masuk pada situs perusahaan. Aturan B mengaplikasikan pada sambungan yang keluar dari situs perusahaan.

Aturan D menerima paket yang masuk dari sesuatu yang diduga *server* SMTP (karena paket berasal dari *port* 25) hanya jika paket-paket memiliki susunan bit ACK; hanya jika paket merupakan bagian suatu sambungan yang dimulai dari dalam (dari *client* perusahaan ke *server*-nya).

Jika seseorang berusaha membuka sambungan TCP dari luar, paket pertama yang dia kirim tidak akan mempunyai susunan bit ACK; yang seharusnya ada dalam sambungan awal TCP. Jika *user* memblok paket yang paling awal, berarti *user* akan memblok sambungan TCP secara keseluruhan. Tanpa informasi tertentu dalam *header* paket pertama, misalnya secara khusus mengenai jumlah rangkaian TCP, maka sambungan tidak dapat diciptakan.