Chương trình KC-01:
Nghiên cứu khoa học phát triển công nghệ thông tin và truyền thông

Đề tài KC-01-01:
Nghiên cứu một số vấn đề bảo mật và an toàn thông tin cho các mạng dùng giao thức liên mạng máy tính IP

Báo cáo kết quả nghiên cứu

PHẦN MỀM BẢO MẬT MẠNG DÙNG GIAO THỨC IP

Quyển 4B: "Phần mềm bảo mật trên môi trường Solaris"

HÀ NỘI-2002

Báo cáo kết quả nghiên cứu

PHẦN MỀM BẢO MẬT MẠNG DÙNG GIAO THỨC IP

Quyển 4B: "Phần mềm bảo mật trên môi trường Solaris"

Chủ trì nhóm thực hiện:
TS. Đặng Vũ Sơn

MỤC LỤC

2.1.3.4 Các tiên trình điều khiển (Driver)

MỞ ĐẦU

TCP/IP là bộ giao thức truyền thông được cài đặt trong hầu hết các hệ điều hành mạng và là giao thức chuẩn của Internet. Để bảo vệ thông tin trên mạng dùng giao thức TCP/IP, chúng ta có thể can thiệp mật mã vào một trong 4 tầng là tầng ứng dụng, tầng vận tải, tầng Internet và tầng truy nhập mạng. Việc can thiệp mật mã vào mỗi tầng sẽ có những ưu nhược điểm riêng. Tuy nhiên với sự phát triển mạnh mẽ của Internet và các mạng công cộng, việc can thiệp mật mã vào tầng IP cho phép chúng ta tạo ra các mạng riêng ảo kết nối các mạng nội bộ thông qua kênh công khai. Hơn nữa, giải pháp này đã cho phép bảo vệ được dữ liệu của tất cả các ứng dụng dùng giao thức TCP/IP bao gồm cả ảnh động và âm thanh mà không phải can thiệp vào cấu trúc của ứng dụng. Chính điều này đã giải quyết được một khó khăn trong thực tế hiện nay ở Việt nam là có nhiều ứng dụng có thông tin cần được bảo vệ nhưng chúng ta lại không thể can thiệp mật mã vào cấu trúc của nó và các ứng dụng thời gian thực.

Báo cáo gồm 4 chương, giới thiệu giải pháp nhúng kỹ thuật mật mã vào nhân hệ điều hành Solaris và kết quả khảo sát các chức năng của bộ phần mềm bảo vệ gói IP trên Solaris là sản phẩm của đề tài "***Nghiên cứu bảo vệ dữ liệu ở tầng IP cho các mạng máy tính sử dụng hệ điều hành UNIX***" của Ban Cơ yếu chính phủ.

Chương 1: **Khái quát chung về giải pháp bảo vệ gói IP bằng kỹ thuật mật mã**

Phân tích khả năng bảo vệ thông tin khi can thiệp mật mã vào mỗi tầng của giao thức TCP/IP, đánh giá ưu nhược điểm của giải pháp can thiệp mật mã vào tầng IP. Phân tích cơ chế truyền dữ liệu của giao thức TCP/IP, các dịch vụ bảo vệ gói IP bằng kỹ thuật mật mã. Từ đó đưa ra mô hình chức năng của hệ thống bảo vệ gói IP bằng kỹ thuật mật mã.

Chương 2 : Cơ chế quản lý dữ liệu của giao thức TCP/IP trên Solaris

Trình bầy những vấn đề cơ bản nhất của cơ chế quản lý các thành phần của gói IP trên Solaris , trong đó có cấu trúc STREAMS TCP/IP.

Chương 3: Giải pháp nhúng kỹ thuật mật mã vào nhân hệ điều hành Solaris

Trình bầy giải pháp xây dựng tầng IPF ngay dưới tầng IP trong cấu trúc Streams TCP/IP của Solaris.

Chương 4: Khảo sát khả năng chống lại các phần mềm hacker và tốc độ truyền dữ liệu của hệ thống bảo vệ gói IP bằng kỹ thuật mật mã trên Solaris

Giới thiệu kết quả khảo sát một số đặc trưng của các bộ phần mềm bảo vệ gói IP bằng kỹ thuật mật mã trên nền hệ điều hành Solaris (IPSEC_SUN) và hệ điều hành LINUX (FreeS/WAN), bao gồm khả năng ngăn chặn các tấn công của một số phần mềm Hacker, tốc độ truyền dữ liệu của hệ thống trong các trường hợp không chạy và chạy phần mềm IPSEC_SUN và FreeS/WAN. Khả năng mã hoá dữ liệu gói Multicast phục vụ cho việc bảo vệ dữ liệu hội nghị truyền hình cũng được giới thiệu trong chương này.

CHƯƠNG I

KHÁI QUÁT CHUNG VỀ GIẢI PHÁP BẢO VỆ GÓI IP BẰNG KỸ THUẬT MẬT MÃ

Xây dựng các hệ thống bảo mật thông tin trên mạng máy tính đòi hỏi một giải pháp tổng thể bao gồm nhiều cơ chế an toàn như điều khiển truy nhập, mã hoá dữ liệu, chữ ký số, xác thực, bảo vệ vật lý,... Kỹ thuật mật mã đóng một vai trò rất quan trọng trong việc bảo vệ thông tin trên mạng, nó cho phép chúng ta cài đặt hầu hết các dịch vụ an toàn, bao gồm các dịch vụ bí mật, xác thực, toàn vẹn, không chối bỏ. Ngoài ra nó góp phần quan trọng trong việc cài đặt dịch vụ điều khiển truy nhập.

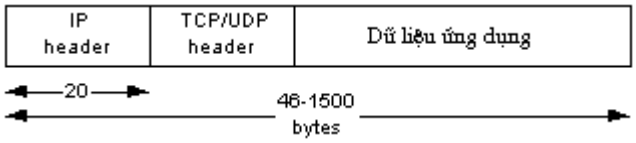
1.1 CAN THIỆP MẬT MÃ VÀO HỆ THỐNG MẠNG DÙNG GIAO THỨC TCP/IP

1.1.1 Can thiệp mật mã vào các tầng trong giao thức TCP/IP

Cấu trúc giao thức TCP/IP cho phép chúng ta can thiệp mật mã vào một tầng bất kỳ tuỳ theo chính sách an toàn được đưa ra. Dưới đây là một số căn cứ khi lựa chọn tầng để can thiệp mật mã.

- **Lựa chọn thành phần của gói IP để bảo vệ**

Một gói IP chứa dữ liệu bao gồm 3 thành phần là dữ liệu ứng dụng (data), header tầng vận tải (TCP/UDP header), header tầng Internet (IP header) như trong hình 1.1 dưới đây.



Hình 1.1: Các thành phần của gói IP

Khi dữ liệu được chuyển từ tầng vận tải xuống các tầng dưới, tại mỗi tầng header điều khiển được gắn vào phía bên trái của dữ liệu do tầng trên chuyển xuống. Mỗi header có cấu trúc riêng và có vai trò trong việc chuyển dữ liệu từ ứng dụng của máy nguồn tới ứng dụng của máy đích. Bảng 1.1 chỉ ra khả năng bảo vệ các thành phần của gói IP khi can thiệp mật mã vào các tầng trong giao thức TCP/IP.

Bảng 1.1: Bảo vệ các thành phần của gói IP trong giao thức TCP/IP

	Data	**TCP/UDP header**	**IP header**
Tầng ứng dụng	x		
Tầng vận tải	x	x	
Tầng Internet	x	x	x
Tầng truy nhập mạng	x	x	x

Nhìn vào bảng 1.1 chúng ta thấy rằng để bảo vệ dữ liệu ứng dụng chúng ta có thể can thiệp mật mã vào một trong bốn tầng. Nhưng để bảo vệ header tầng vận tải chúng ta chỉ có thể can thiệp vào một trong ba tầng dưới. Tầng ứng dụng không quan tâm đến header của tầng vận tải được nối vào đầu khối dữ liệu do nó chuyển xuống. Cuối cùng để bảo vệ IP header chúng ta chỉ có thể lựa chọn hai tầng dưới. Như vậy là để bảo vệ toàn bộ gói IP chúng ta phải can thiệp mật mã vào một trong hai tầng dưới cùng là tầng Internet và tầng truy nhập mạng. Tuy nhiên ta cần chú ý là tại tầng truy nhập mạng chúng ta có thể bảo vệ toàn bộ frame chứa gói IP bao gồm cả địa chỉ vật lý của giao diện mạng.

- **Lựa chọn dịch vụ cần được cài đặt**

Chuẩn ISO 7498-2 đã chỉ ra các dịch vụ an toàn được cài đặt tại các tầng trong mô hình OSI. Đối chiếu mô hình TCP/IP với mô hình OSI, chúng ta có các dịch vụ an toàn được cài đặt bằng kỹ thuật mật mã tại các tầng của giao thức TCP/IP như trong bảng 1.2. Ta có một số nhận xét sau:

- Tại tầng ứng dụng chúng ta có thể cài đặt tất cả các dịch vụ an toàn.
- Tầng vận tải có hai giao thức là TCP và UDP, trong đó TCP là giao thức kết nối và UDP là giao thức không kết nối. Tại tầng vận tải ta có thể cài đặt cả dịch vụ an toàn kết nối và không kết nối.
- Giao thức IP là giao thức không kết nối, các dịch vụ an toàn cài tại tầng IP là các dịch vụ không kết nối.
- Tại tầng truy nhập mạng ta chỉ có thể cài đặt một số dịch vụ bí mật.

Bảng 1.2: Các dịch vụ an toàn được cài đặt tại các tầng của giao thức TCP/IP

Dịch vụ an toàn	Tầng ứng dụng	Tầng vận tải	Tầng Internet	Tầng truy nhập mạng
Bí mật kết nối	x	x		x
Bí mật không kết nối	x	x (UDP)	x	x
Bí mật trường lựa chọn	x			
Bí mật luồng giao dịch	x		x	x
Xác thực thực thể	x	x (TCP)		
Xác thực nguồn gốc dữ liệu	x	x (UDP)	x	
Toàn vẹn kết nối khôi phục	x	x (TCP)		
Toàn vẹn không kết nối	x	x (UDP)	x	
Toàn vẹn trường lựa chọn	x			
Không chối bỏ	x			

1.1.1.1 Can thiệp mật mã vào tầng ứng dụng

Quyết định nhúng mật mã vào tầng ứng dụng được đưa ra khi:

Dịch vụ an toàn cần được tích hợp vào một ứng dụng cụ thể như thư điện tử, truyền tệp, dịch vụ WEB ...

Khi can thiệp mật mã vào tầng ứng dụng chúng ta phải quan tâm đến các đòi hỏi gắn liền với ứng dụng. Chẳng hạn trong ứng dụng truyền tệp, ta cần quan tâm đến cơ chế điều khiển truy nhập file như đọc và cập nhật các danh sách điều khiển truy nhập của file. Trong các trường hợp khác chúng ta phải truy nhập được vào các trường lựa chọn của dữ liệu nhằm cài đặt các dịch vụ bí mật trường lựa chọn, toàn vẹn trường lựa chọn... Chẳng hạn chúng ta cần bảo vệ trường số định danh cá nhân (PIN) trong giao dịch tài chính, các trường của một cơ sở dữ liệu ...

Dữ liệu cần bảo vệ phải đi qua các bộ chuyển tiếp ứng dụng.

Thư điện tử là một ví dụ điển hình về ứng dụng trong đó dữ liệu cần bảo vệ phải đi qua các bộ chuyển tiếp ứng dụng. Trường nội dung thư được bảo vệ bằng kỹ thuật mật mã, các trường khác như trường địa chỉ, trường vết được để nguyên

vì các hệ thống trung gian cần biết các thông tin này. Trong tất cả các trường hợp như vậy, dịch vụ an toàn phải được cài đặt tại mức ứng dụng.

Ưu điểm của việc can thiệp mật mã vào tầng ứng dụng là :

Có thể cài đặt được tất cả các dịch vụ an toàn.

Có thể quyết định cách thức bảo vệ cho từng loại dữ liệu của ứng dụng.

Không đòi hỏi sự can thiệp vào các tầng thấp hơn khi cài đặt dịch vụ an toàn.

Hạn chế của việc can thiệp mật mã vào tầng ứng dụng là :

Phải can thiệp mật mã vào cấu trúc của mỗi ứng dụng cần bảo vệ.

Phải can thiệp mật mã vào tất cả các các máy đang chạy ứng dụng cần bảo vệ.

Ảnh hưởng đến tính toàn vẹn của ứng dụng, khó khăn trong việc di chuyển ứng dụng sang hệ thống khác.

Người dùng phải thay đổi thói quen làm việc với ứng dụng.

1.1.1.2 Can thiệp mật mã vào tầng vận tải

Những yếu tố dẫn đến quyết định lựa chọn tầng vận tải là:

Khả năng tạo ra các dịch vụ bảo vệ trong suốt với ứng dụng.

Khả năng tạo ra dịch vụ bảo vệ luồng dữ liệu có hiệu năng cao nhờ khả năng thao tác trên các khối dữ liệu lớn và xử lý dữ liệu của nhiều ứng dụng theo một cách thức chung.

Việc quản lý chính sách an toàn cho toàn hệ thống đầu cuối do một người quản trị phụ trách, không phân biệt từng ứng dụng riêng rẽ.

Cần bảo vệ cả header tầng vận tải của gói IP.

Để cài đặt dịch vụ an toàn vào tầng vận tải, cần có các yêu cầu sau:

Thừa nhận rằng hệ thống đầu cuối (máy tính hiện thời) là tin cậy nhưng tất cả mạng truyền thông là không tin cậy.

Các đòi hỏi an toàn được đưa ra bởi người có thẩm quyền của hệ thống đầu cuối và áp dụng cho tất cả các giao dịch không quan tâm đến ứng dụng cụ thể.

Ưu điểm của việc can thiệp mật mã vào tầng vận tải:

Bảo vệ được dữ liệu của tất cả các ứng dụng dùng giao thức TCP/IP.

Không phải can thiệp và sửa đổi các ứng dụng hiện có.

Trong suốt với người dùng.

Cho phép bảo vệ dữ liệu của từng kết nối dùng giao thức TCP.

Hạn chế của việc can thiệp mật mã vào tầng vận tải là :

Không cài đặt được tất cả các dịch vụ an toàn như chỉ ra trong bảng 1.2.

Chỉ có một cơ chế bảo vệ chung cho dữ liệu của tất cả các ứng dụng.

Phải cài đặt dịch vụ an toàn tại tất cả các máy có thông tin cần bảo vệ.

1.1.1.3 Can thiệp mật mã vào tầng Internet:

Những yếu tố dẫn đến quyết định lựa chọn tầng Internet là :

Khả năng tạo ra các dịch vụ bảo vệ trong suốt với ứng dụng.

Khả năng tạo ra dịch vụ bảo vệ luồng dữ liệu có hiệu năng cao nhờ khả năng thao tác trên các khối dữ liệu lớn và xử lý dữ liệu của nhiều ứng dụng theo một cách thức chung.

Việc quản lý chính sách an toàn cho toàn hệ thống đầu cuối do một người quản trị phụ trách, không phân biệt từng ứng dụng riêng rẽ.

Hạn chế số nút cần cài đặt dịch vụ an toàn. Nếu mạng nội bộ được coi là an toàn thì mỗi mạng nội bộ chỉ cần lựa chọn một thiết bị làm Gateway ngầm định và gói IP được bảo vệ tại Gateway này. Tất cả các gói IP được sinh bởi các ứng dụng trong mạng con sẽ được can thiệp mật mã tại các Gateway trước khi ra kênh công khai.

Quan tâm đến thông tin định tuyến của gói IP, chẳng hạn cần tích hợp với ứng dụng Firewall lọc gói.

Cần bảo vệ các thành phần bất kỳ của gói IP kể cả IP header.

1.1.1.4 Can thiệp mật mã vào tầng truy nhập mạng

Tầng truy nhập mạng trong giao thức TCP/IP về cơ bản thực hiện chức năng của hai tầng liên kết dữ liệu (data link) và tầng vật lý trong mô hình OSI. Việc can thiệp mật mã vào tầng này cho phép bảo vệ dữ liệu của các frame trong đó có frame chứa gói IP. Nó không quan tâm đến cấu trúc gói IP cũng như các gói thông tin khác được chứa trong frame. Chính vì vậy tại các nút trung gian trên đường truyền, frame phải được giải mã để các Router biết thông tin định tuyến gói IP, sau đó frame lại được mã trở lại để chuyển đi. Người ta thường tạo ra các thiết bị phần cứng để bảo vệ dữ liệu mức truy nhập mạng.

Ưu điểm của việc can thiệp mật mã vào tầng truy nhập mạng:

Cho phép bảo vệ toàn bộ gói IP và các gói điều khiển khác như các gói ARP và RARP.
Cho phép tạo ra các thiết bị cầu (Bridge) có chức năng bảo vệ thông tin bằng kỹ thuật mật mã.
Cho phép tạo ra các thiết bị mã luồng tốc độ cao.

Hạn chế của việc can thiệp mật mã vào tầng truy nhập mạng:

Chỉ có thể cài đặt mật mã dùng phương thức theo tuyến truyền (Link to Link). Mỗi thiết bị chuyển mạch gói trung gian (Router, Gateway) phải tiến hành giải mã toàn bộ dữ liệu mã để thu được các thông tin định tuyến sau đó lại mã dữ liệu để truyền đi theo tuyến mới.
Chỉ cài đặt được một số dịch vụ an toàn.

1.1.2 Ý nghĩa của việc can thiệp mật mã vào tầng IP

Như ở phần trên đã trình bầy, việc can thiệp mật mã vào mỗi tầng trong cấu trúc giao thức TCP/IP đều có những ưu điểm và hạn chế riêng. Trong phần này chúng tôi phân tích ưu điểm và hạn chế của việc can thiệp mật mã vào tầng IP.

1.1.2.1 Bảo vệ được dữ liệu của tất cả các ứng dụng dùng giao thức TCP/IP

Chúng ta biết rằng TCP/IP là một giao thức chuẩn của Internet và được hầu hết các ứng dụng trên mạng hỗ trợ. Các ứng dụng dùng giao thức TCP/IP như thư điện tử, cơ sở dữ liệu, dịch vụ WEB, truyền tệp, truyền ảnh động và âm thanh. Dữ liệu của các ứng dụng này được chứa trong các gói IP và được bảo vệ nhờ các dịch vụ an toàn tại tầng Internet. Mỗi mạng con chỉ cần có một thiết bị bảo vệ gói IP và là một Gateway để cho phép tất cả các gói IP phải chuyển qua nó trước khi ra kênh công khai.

1.1.2.2 Không phải can thiệp và sửa đổi các ứng dụng hiện có

Do cài đặt tại tầng Internet nên các dịch vụ an toàn không quan tâm đến ứng dụng tạo ra dữ liệu chứa trong gói IP. Tất cả các gói IP của các ứng dụng khác nhau đều được xử lý theo một cách thức chung. Chính vì vậy, chúng ta không phải can thiệp và sửa đổi cấu trúc của ứng dụng cần bảo vệ.

1.1.2.3 Trong suốt với người dùng

Toàn bộ các thao tác bảo vệ gói IP được thực hiện tại tầng Internet một cách trong suốt với ứng dụng và người dùng. Người dùng không cần phải can thiệp vào quá trình thực hiện các dịch vụ an toàn và cũng không cần phải thay đổi các thao tác làm việc với ứng dụng.

1.1.2.4 Tăng cường các khả năng của Firewall

Chúng ta biết rằng có một số tấn công đối với Firewall lọc gói như soi gói, giả địa chỉ nguồn, chiếm kết nối... Những tấn công này không thể phát hiện ra nếu chỉ dùng các luật lọc gói. Khi cài đặt các dịch vụ an toàn bằng kỹ thuật mật mã tại tầng Internet, các tấn công trên sẽ bị ngăn chặn. Ngược lại thiết bị bảo vệ gói IP bằng kỹ thuật mật mã không thể ngăn chặn các gói bằng cách dựa vào các luật lọc gói. Nếu kết hợp chức năng lọc gói và chức năng bảo vệ gói IP bằng kỹ thuật mật mã chúng ta có thể tạo ra các Firewall có độ an toàn cao.

1.1.2.5 Giảm số đầu mối cần can thiệp dịch vụ an toàn

Khi cài đặt dịch vụ an toàn tại mức ứng dụng và mức vận tải, chúng ta chỉ có thể tạo ra một kênh an toàn giữa hai hệ thống đầu cuối, nghĩa là giữa hai máy tính có thông tin cần bảo vệ. Khi số hệ thống đầu cuối (chủ yếu là các máy PC) tại các mạng nội bộ tăng lên, số đầu mối cần can thiệp dịch vụ an toàn cũng sẽ tăng theo. Điều này dẫn đến những khó khăn về chi phí, quản trị hệ thống, huấn luyện đào tạo... Do can thiệp tại tầng Internet, nên chúng ta có thể tạo ra các Gateway có khả năng chặn bắt và bảo vệ gói IP của tất cả các máy trong một mạng nội bộ và dịch vụ an toàn chỉ cần cài đặt tại Gateway này.

1.1.2.6 Cho phép bảo vệ dữ liệu của một số ứng dụng giao tiếp thời gian thực

Việc can thiệp mật mã vào tầng ứng dụng không phải lúc nào cũng thực hiện được. Nhất là người can thiệp mật mã không phải là nhà sản xuất để tạo ra ứng dụng. Hơn nữa các ứng dụng thời gian thực đòi hỏi các luồng dữ liệu phải được chuyển gần như tức thời, không cho phép bị trễ lâu. Với các ứng dụng như vậy, việc cài đặt các dịch vụ an toàn tại tầng Internet và tầng truy nhập mạng là phù hợp. Tuy nhiên việc can thiệp vào tầng truy nhập mạng có hạn chế là phải dùng phương thức mã theo tuyến, tại các nút trung gian dữ liệu phải giải mã để tìm ra thông tin định tuyến sau đó được mã trở lại để đi tiếp. Khi cài đặt tại tầng Internet chúng ta dùng phương thức mã từ mút tới mút, trong đó header truyền thông được để ở dạng rõ và gói IP không cần phải giải mã và mã hoá tại các nút

truyền thông trung gian. Một ví dụ về ứng dụng thời gian thực là phần mềm trong thiết bị hội thảo trên mạng.

Hạn chế của việc bảo vệ dữ liệu tại tầng IP:

Không có khả năng cài đặt một số dịch vụ an toàn đặc biệt là dịch vụ xác thực thực thể và không chối bỏ.

Chỉ có một cơ chế bảo vệ chung cho tất cả các dữ liệu của tất cả các ứng dụng.

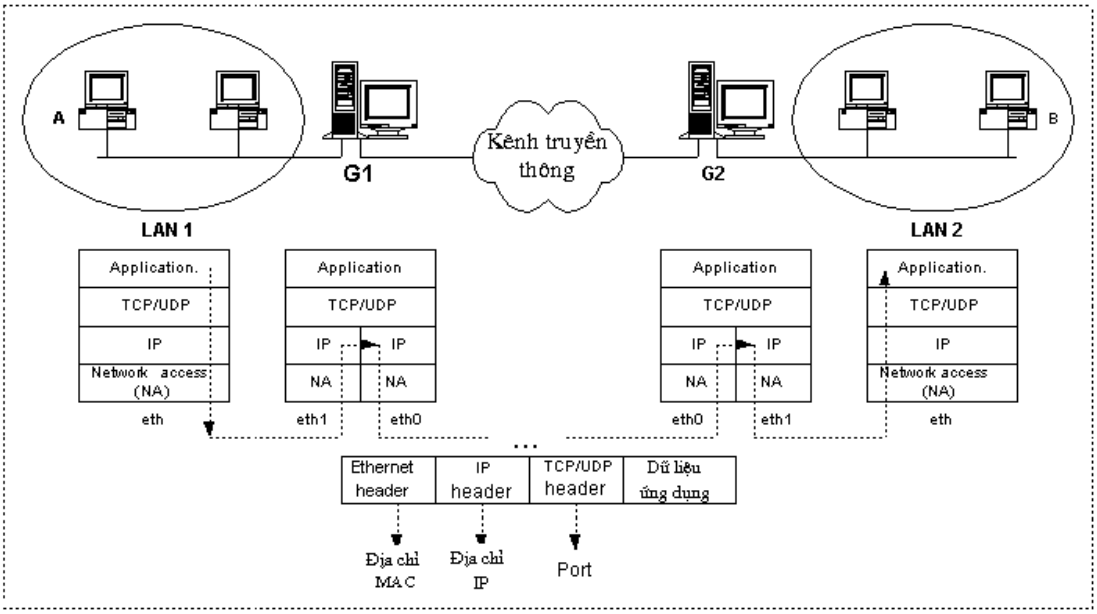
Người quản trị mạng phải có kiến thức tốt về công nghệ mạng và quản trị mạng.

Mạng nội bộ đứng sau Gateway bảo vệ gói IP phải an toàn.

1.2 GIAO THỨC AN TOÀN TẦNG INTERNET

1.2.1 Quá trình truyền dữ liệu của giao thức TCP/IP

Phần này mô tả quá trình truyền dữ liệu giữa hai máy tính dùng giao thức TCP/IP trong mô hình LAN to LAN với hai Gateway.



Hình 1.2: Quá trình truyền dữ liệu của giao thức TCP/IP

Hình 1.2 mô tả một mạng Ethernet bao gồm hai mạng nội bộ LAN 1 và LAN 2 được kết nối với nhau qua hai Gateway tương ứng là G1 và G2. Mỗi Gateway có hai giao diện mạng, trong đó giao diện eth1 có cùng lớp địa chỉ IP với mạng LAN bên trong và giao diện eth0 có cùng lớp địa chỉ IP với giao diện

ngoài của Gateway kia. Hai Gateway kết nối với nhau qua kênh truyền công khai. Quá trình truyền dữ liệu giữa máy A của LAN 1 với máy B của LAN 2 diễn ra như sau:

Tại tầng ứng dụng của máy A, toàn bộ dữ liệu cần truyền tới máy B (file, thư điện tử, trang WEB,...) được chia thành các khối nhỏ gọi là " dữ liệu ứng dụng". Mỗi khối " dữ liệu ứng dụng" này sẽ được chứa trong các gói IP khác nhau và truyền đến B theo các tuyến đường khác nhau. Đầu tiên, khối "dữ liệu ứng dụng" được chuyển xuống tầng vận tải, tại đây một header được gắn vào phía trước khối dữ liệu tuỳ thuộc vào giao thức tầng vận tải được dùng là TCP hay UDP. Toàn bộ khối thu được gọi là phân đoạn tầng vận tải. Tiếp đó, phân đoạn tầng vận tải được chuyển xuống tầng IP. Tại đây một IP header được gắn vào phía trước header của tầng vận tải để thu được gói IP và chuyển xuống tầng truy nhập mạng. Nhờ giao thức ARP cho phép phân giải địa chỉ IP thành địa chỉ MAC, tầng truy nhập mạng xác định được địa chỉ MAC của G1. Một Ethernet header được gắn vào phía trước IP header (trong đó chứa các địa chỉ MAC của máy A và Gateway G1) để tạo thành một Ethernet frame. Tiếp theo, Ethernet frame được phát theo phương thức quảng bá CSMA/CD trên mạng LAN 1. Vì địa chỉ MAC đích trùng với địa chỉ MAC của G1, nên G1 nhận Ethernet frame trên tại giao diện trong eth1 của nó.

Tại tầng truy nhập mạng của giao diện eth1 của G1, Ethernet header được loại bỏ và gói IP được chuyển lên tầng IP. Căn cứ vào địa chỉ IP đích, chức năng định tuyến của G1 chuyển tiếp gói IP sang tầng IP của giao diện ngoài eth0 và xuống tầng truy nhập mạng. Tại đây một Ethernet header mới lại được thêm vào để bắt đầu qúa trình chuyển gói IP tới giao diện eth0 của Gateway G2 thông qua rất nhiều Router trung gian trên kênh truyền công khai.

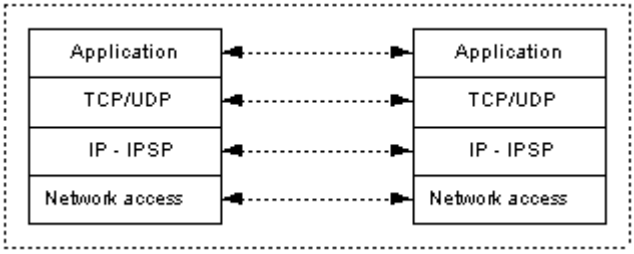
Tại tầng truy nhập mạng của giao diện eth0 của G2, Ethernet header được loại bỏ và gói IP được chuyển lên tầng IP. Căn cứ vào địa chỉ IP đích, chức năng định tuyến của G2 chuyển gói IP sang tầng IP của giao diện eth1 và xuống tầng truy nhập mạng. Nhờ giao thức ARP, tầng truy nhập mạng của G2 xác định được địa chỉ MAC của máy B và một Ethernet header được gắn vào phía trước IP header để tạo một frame. Trong Ethernet header chứa địa chỉ MAC của giao diện trong của G2 và địa chỉ MAC của máy B. Sau đó G2 phát quảng bá Ethernet frame để máy B nhận được.

Tại tầng truy nhập mạng của máy B, Ethernet header được loại bỏ và gói IP được chuyển lên tầng IP. Tiếp theo IP header được loại bỏ và phân đoạn tầng vận

tải được chuyển lên tầng vận tải. Tầng vận tải căn cứ vào giá trị cổng ứng dụng trong TCP/UDP header để ghép toàn bộ các khối "dữ liệu ứng dụng" thành dữ liệu ban đầu (file, thư điện tử, ...).

1.2.2 Cấu trúc TCP/IP với giao thức an toàn tầng Internet

Trong mô hình OSI, giao thức an toàn tầng mạng NLSP (Network Layer Security Protocol) tương ứng với giao thức an toàn tầng IP trong kiến trúc giao thức TCP/IP. Phần này trình bầy mô hình TCP/IP với giao thức an toàn IPSP (IP Security Protocol). Chúng ta có hình 1.3 là mô hình truyền thông tổng quát giữa hai hệ thống TCP/IP khi có giao thức IPSP.

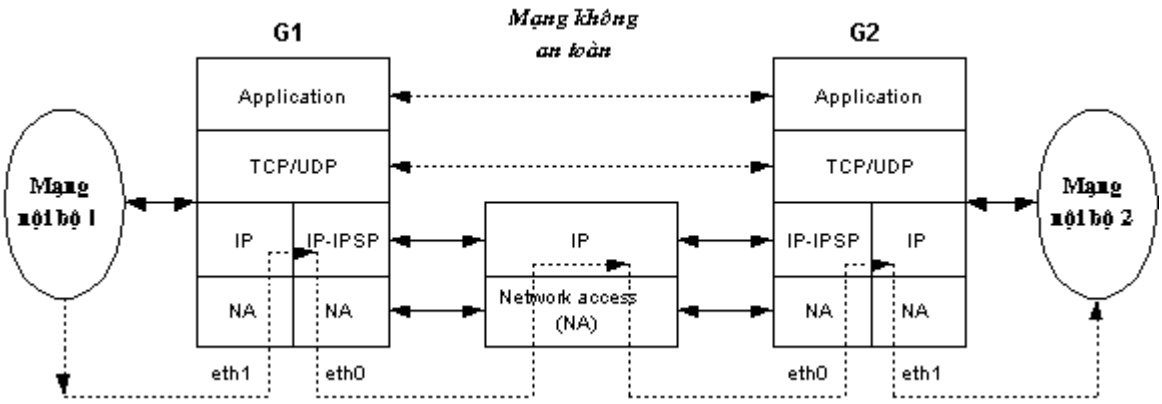


Hình 1.3: Mô hình TCP/IP với giao thức an toàn tầng IP

Cũng như các giao thức khác, giao thức IPSP trong cấu trúc TCP/IP phải có tập các quy tắc, quy ước trong việc bảo vệ gói IP bằng kỹ thuật mật mã như các thoả thuận về dịch vụ an toàn, cơ chế an toàn, các thuật toán mã hoá, các thuật toán xác thực, khoá mật mã. Tại tầng IP, giao thức IPSP sẽ xử lý các gói IP và chuyển chúng xuống tầng truy nhập mạng để ra mạng không an toàn bên ngoài. Giao thức an toàn tầng IP được cài đặt cho từng cặp các máy tính thường là các Gateway trên mạng. Để các Router trung gian có thể hiểu và xử lý các gói IP trên đường tới đích, các thông tin trong IP header phải để ở dạng rõ và không bị mã hoá. Dịch vụ bảo vệ gói IP có thể được cài đặt giữa hai máy PC, giữa một máy PC và một Gateway hoặc giữa hai Gateway.

Việc cài đặt dịch vụ bảo vệ gói IP trên hai Gateway cho phép chúng ta bảo vệ được các giao dịch giữa hai máy bất kỳ của hai mạng bên trong của hai Gateway.

Hình 1.4 là mô hình truyền thông an toàn dùng hai Gateway bảo vệ gói IP.



Hình 1.4: Mô hình truyền thông an toàn dùng hai Gateway bảo vệ gói IP

Trong mô hình trên, mỗi mạng nội bộ có một Gateway bảo vệ gói IP bao gồm hai giao diện mạng, giao diện trong eth1 có cùng lớp địa chỉ IP với mạng nội bộ và giao diện ngoài eth0 có chức năng bảo vệ gói IP và truyền thông với Gateway tương ứng.

Trong mô hình này đòi hỏi giao dịch giữa các máy của mạng nội bộ và Gateway tương ứng là an toàn. Đây là mô hình được sử dụng nhiều trong thực tế. Các dịch vụ an toàn không phải cài đặt tại các máy trong hai mạng nội bộ mà chỉ cần cài đặt tại hai Gateway. Dữ liệu của các ứng dụng dùng giao thức TCP/IP trên hai mạng nội bộ sẽ được bảo vệ khi truyền trên kênh không an toàn.

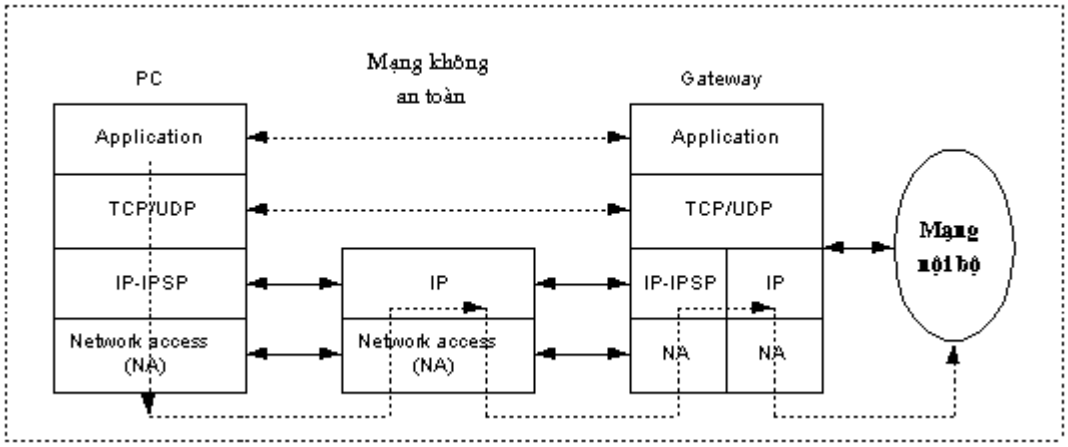
Nguyên tắc hoạt động của hệ thống như sau:

Giả sử chúng ta cần bảo vệ các gói IP được truyền từ mạng nội bộ 1 đến mạng nội bộ 2.

- Các gói IP được sinh bởi các ứng dụng trên các máy của mạng nội bộ 1 sẽ được chuyển đến giao diện trong eth1 của Gateway G1.
- Gói IP được chuyển lên tầng IP của giao diện trong, thông tin trong IP header được dùng để định tuyến gói IP và gói IP được chuyển sang tầng IP với giao thức IPSP (IP-IPSP) của giao diện ngoài eth0 của Gateway G1. Tại đây, gói IP được xử lý bởi các dịch vụ an toàn dùng kỹ thuật mật mã sau đó được chuyển xuống tầng truy nhập mạng eth0 để ra mạng không an toàn.

o Qua các Router trung gian, các gói IP đến được giao diện ngoài eth0 của Gateway G2 và được chuyển lên tầng IP với giao thức IPSP (IP-IPSP). Tại đây gói IP được xử lý bởi các dịch vụ an toàn và được chuyển đến tầng IP của giao diện trong eth1. Sau đó được chuyển xuống tầng truy nhập mạng để đi vào mạng bên trong an toàn.

Hình 1.5 là mô hình truyền thông an toàn giữa một máy PC và một mạng nội bộ thông qua một Gateway.



Hình 1.5: Mô hình truyền thông an toàn giữa một PC và mạng nội bộ

Trong mô hình này, PC chỉ có một giao diện mạng có chức năng bảo vệ gói IP. Thông qua Gateway, máy PC sẽ giao dịch an toàn với các máy của mạng nội bộ an toàn đứng sau Gateway. Khi kết nối PC với một Gateway bảo vệ gói IP đôi khi cần phải xác thực người sử dụng. Để giải quyết vấn đề này cần bổ xung chức năng xác thực người dùng tại tầng ứng dụng cho Gateway an toàn. Khi đó Gateway có thêm chức năng uỷ quyền ứng dụng. Khi một người dùng muốn kết nối với một máy chủ ở xa trong một mạng tin cậy sau Gateway, họ phải được xác thực tại Gateway trước khi được phép kết nối với máy nội bộ.

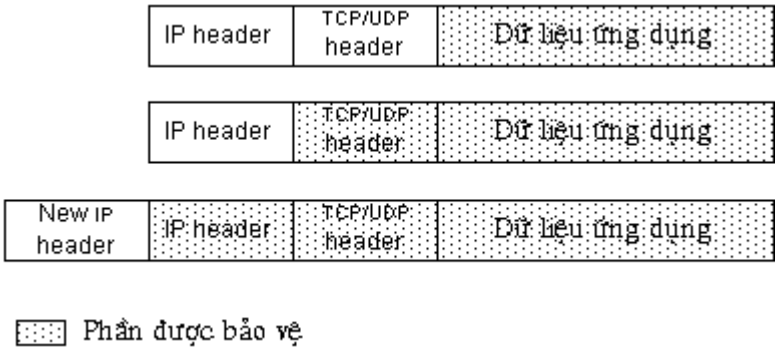
Trong các mô hình trên, Gateway có thể được coi như một Firewall với các chức năng lọc và bảo vệ gói IP tại tầng IP và chức năng uỷ quyền tại tầng ứng dụng. Việc kết hợp chức năng bảo vệ gói IP bằng kỹ thuật mật mã và các chức năng điều khiển truy nhập sẽ tạo ra các Firewall có độ an toàn cao.

1.3 CÁC DỊCH VỤ BẢO VỆ GÓI IP BẰNG KỸ THUẬT MẬT MÃ

Kỹ thuật mật mã đóng một vai trò rất quan trọng trong bảo vệ thông tin. Có thể nói rằng, trên kênh truyền không an toàn, mã hoá là kỹ thuật duy nhất có khả năng bảo vệ thông tin được bí mật, xác thực và toàn vẹn,... Phần này phân tích khả năng dùng kỹ thuật mật mã để bảo vệ các thành phần của gói IP.

1.3.1 Dịch vụ bí mật

Mã hóa là một trong các kỹ thuật rất quan trọng trong việc bảo vệ thông tin. Để cài đặt dịch vụ bí mật cho gói IP, chúng ta có thể lựa chọn để mã hoá các thành phần của gói IP như trong hình 1.6.



Hình 1.6: Lựa chọn các thành phần của gói IP để mã hoá

+ Mã dữ liệu ứng dụng

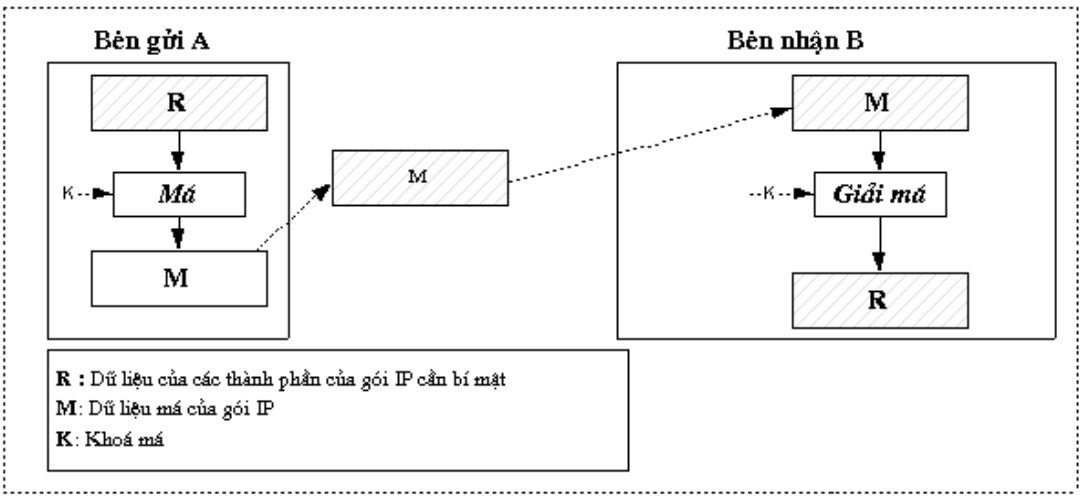
Mã hoá dữ liệu ứng dụng là đòi hỏi đương nhiên khi cài đặt dịch vụ bí mật. Nó cho phép giữ bí mật dữ liệu của tất cả các ứng dụng dùng giao thức TCP/IP.

+ Mã phân đoạn tầng vận tải

Việc mã hoá phân đoạn tầng vận tải bao gồm dữ liệu ứng dụng và TCP/UDP header sẽ cho phép chúng ta ngăn chặn một số tấn công đối với gói IP như chặn và lọc các gói IP của từng ứng dụng, tạo ra các gói thiết lập kết nối giả trong giao thức TCP/IP. Hơn nữa việc mã hoá header tầng vận tải sẽ cho phép chúng ta lợi dụng chức năng tổng kiểm tra (checsum) của header tầng vận tải để xác thực gói IP.

+ Mã hoá toàn bộ gói IP

Việc mã hoá toàn bộ gói IP cho phép giữ bí mật tất cả các thành phần của gói IP, khi IP header được mã hoá sẽ cho phép chúng ta chống lại một số tấn công như giả địa chỉ nguồn, phân tích luồng giao dịch, chặn bắt các gói IP ...



Hình 1.7: Mô tả quá trình cài đặt dịch vụ bí mật cho gói IP

Hình 1.7 mô tả quá trình cài đặt dịch vụ bí mật cho gói IP bằng kỹ thuật mật mã. Bên gửi A mã hoá các thành phần cần giữ bí mật của gói IP bởi một thuật toán mã khối với khoá K để được phần dữ liệu mã M. Sau đó A gửi cho B gói IP với các thành phần đã được mã.

Bên nhận B giải mã các thành phần được mã của gói IP để thu được gói IP ban đầu.

Những điều cần lưu ý khi mã hoá toàn bộ gói IP là:

- Phải bổ xung một IP header mới để chứa các thông tin định tuyến cho gói IP tới đích. Việc mã hoá cả IP header được sử dụng trong trường hợp các dịch vụ an toàn được cài đặt tại hai Gateway của hai mạng nội bộ an toàn. Khi đó IP header mới sẽ chứa địa chỉ IP của Gateway nguồn và Gateway đích. Các địa chỉ IP của các máy thuộc hai mạng nội bộ được giữ bí mật.

- Khi mã hoá các thành phần của gói IP bằng kỹ thuật mã khối, chúng ta có 4 chế độ làm việc. Khi dùng hai chế độ ECB và CBC (mã từng khối 64 bit), nói chung chúng ta phải bổ xung phần dữ liệu đệm padding vào cuối gói IP để tổng độ dài gói (tính theo bit) là bội của 64. Nếu dùng hai chế độ CFB và OFB, mỗi lần mã một khối 8 bit (1 byte) thì chúng ta không phải bổ xung phần padding vì độ dài các thành phần của gói IP là số chẵn các byte.

- Khi sử dụng các chế độ CBC, CFB và OFB hai hệ thống phải thống nhất với nhau về giá trị véc tơ khởi tạo 64 bit.

1.3.2 Dịch vụ xác thực và toàn vẹn

Ngoài dịch vụ bí mật, kỹ thuật mật mã còn cho phép chúng ta cài đặt dịch vụ xác thực và dịch vụ toàn vẹn. Nguyên tắc của việc cài đặt dịch vụ xác thực dùng kỹ thuật mật mã là chỉ người nào có khoá mật mã bí mật nào đó mới có thể tạo ra bằng chứng xác thực trên thông báo được gửi đi và thông báo không bị sửa đổi trên đường truyền.

Từ khái niệm trên, chúng ta thấy rằng nếu một gói IP được coi là đã được xác thực trong khi truyền từ nguồn tới đích thì cũng có thể khẳng định rằng gói IP đó đã được toàn vẹn. Chính vì vậy khi cài đặt dịch vụ an toàn, người ta thường kết hợp dịch vụ xác thực với dịch vụ toàn vẹn. Theo William Stallings, "xác thực thông báo là một thủ tục để xác nhận rằng các thông báo đến từ nguồn đã được chỉ ra và không bị thay đổi". Cũng như khi mã hoá, chúng ta có thể lựa chọn các thành phần của gói IP để cài đặt dịch vụ xác thực và toàn vẹn. Các thành phần có thể lựa chọn là dữ liệu ứng dụng, phân đoạn tầng vận tải hoặc toàn bộ gói IP.

Do cấu trúc của gói IP nên chúng ta có lưu ý là:

- Một số trường sẽ thay đổi trong quá trình gói tin được truyền trên mạng như trường Time To Live (TTL), trường Checksum trong IP header. Khi cài đặt dịch vụ xác thực và toàn vẹn những trường này sẽ không được tính và thường được gán giá trị 0.

- Dữ liệu dùng để kiểm tra tính xác thực phải được chuyển tới máy đích, chính vì vậy cần phải chèn phần dữ liệu này vào gói IP được chuyển đi.

Để đảm bảo tính xác thực và toàn vẹn các thành phần của gói IP, người ta có thể dùng các hàm xác thực. Các hàm xác thực có chức năng tạo ra các bằng chứng xác thực (authenticator) để xác thực thông báo. Các hàm xác thực được chia thành ba lớp như sau:

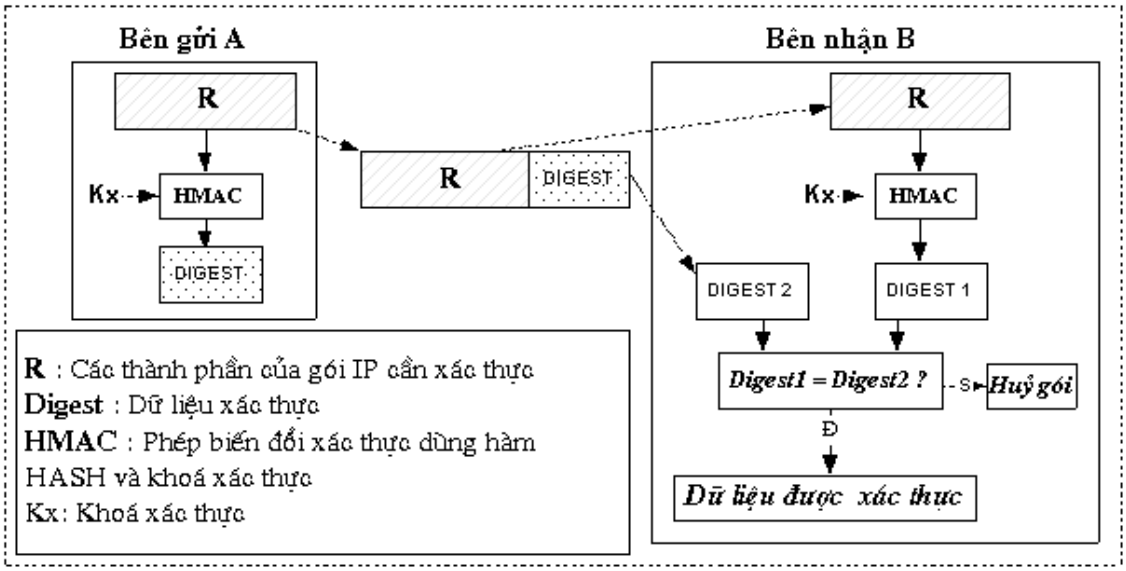
- Lập mã thông báo (Message encryption): Bản mã của thông báo là bằng chứng xác thực.

- Tổng kiểm tra mật mã (Cryptographic checksum) hay mã xác thực thông báo (MAC - Message authentication code): Một hàm công khai của thông báo và một khoá bí mật tạo thành một giá trị độ dài cố định để làm bằng chứng xác thực.

- Hàm băm (Hash funtion): Một hàm công khai ánh xạ một thông báo có độ dài bất kỳ thành một giá trị băm để làm bằng chứng xác thực.

Hàm băm được dùng để tạo ra bằng chứng xác thực. Trong trường hợp này giá trị của hàm băm sẽ được mã hoá bởi khoá bí mật của người gửi. Một cách dùng khác đối với hàm băm là nó được kết hợp với một phép biến đổi mật mã để tạo thành một giá trị làm bằng chứng xác thực. Phép biến đổi mật mã thường được dùng với hàm băm là HMAC (Hashed Message Authentication Code). Đây là một biến dạng của mã xác thực thông báo. Với đầu vào là một thông báo và một khoá, HMAC sẽ sử dụng hàm băm để có một giá trị đầu ra làm bằng chứng xác thực.

Do đặc thù của gói IP, đòi hỏi thời gian xử lý nhanh, nên chúng ta không thể dùng mật mã khoá công khai để cài đặt các dịch vụ xác thực và toàn vẹn mà chỉ có thể dùng mật mã khoá bí mật. Theo chuẩn IPSEC, trong việc cài đặt dịch vụ xác thực và toàn vẹn gói IP, phép biến đổi xác thực HMAC với hàm băm và khoá xác thực được sử dụng rộng rãi.



Hình 1.8 : Sơ đồ cài đặt dịch vụ xác thực cho gói IP

Hình 1.8 mô tả quá trình cài đặt dịch vụ xác thực gói IP dùng phép biến đổi HMAC với hàm băm và khoá xác thực. Hoạt động của sơ đồ trên như sau:

Tại bên gửi A:

- Dữ liệu R cần được xác thực của gói IP (dữ liệu ứng dụng, phân đoạn tầng vận tải hoặc toàn bộ gói IP trừ một số trường thay đổi) được lấy làm đầu vào của thuật toán xác thực HMAC với một hàm băm và khoá xác thực để thu được một dữ liệu xác thực digest với kích thước nhỏ. Digest được dùng làm bằng chứng xác thực (authenticator).

- Digest được gắn với dữ liệu R (phía trước hoặc phía sau) để truyền đến nơi nhận

Tại bên nhận B:

- Dữ liệu R được xử lý bởi thuật toán xác thực HMAC để thu được một giá trị digest1. Đây là giá trị digest thực sự của thông báo.

- Nếu digest1 trùng với giá trị digest đi sau R do A gửi đến, thì B khẳng định rằng dữ liệu R của gói IP đã được xác thực. Nếu hai giá trị trên không trùng nhau thì gói bị huỷ.

1.3.3 Kết hợp dịch vụ bí mật với dịch vụ xác thực, toàn vẹn

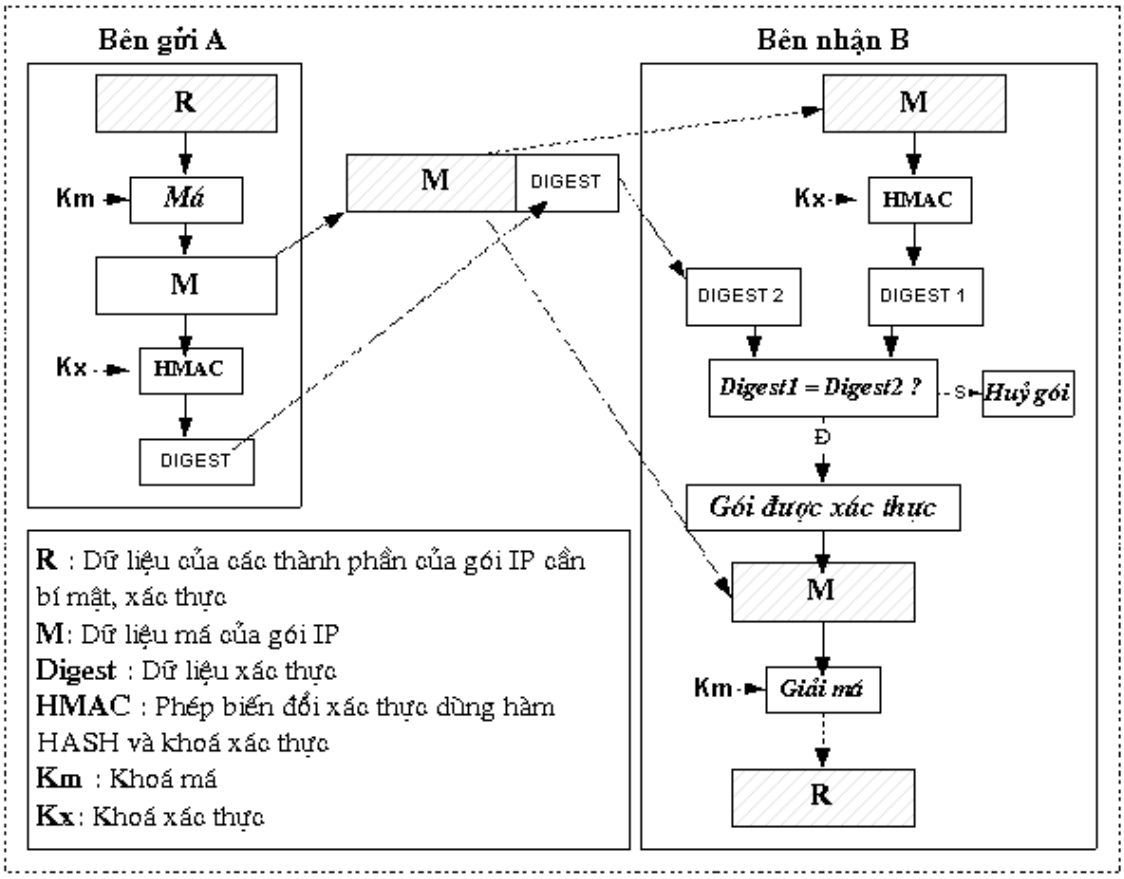
Hình 1.9 là mô hình tổng quát của việc cài đồng thời dịch vụ bí mật và xác thực, toàn vẹn các thành phần của gói IP. Hoạt động của mô hình như sau:

Tại bên gửi A:

- Dữ liệu R của các thành phần cần cài đặt dịch vụ an toàn của gói IP được mã hoá bởi kỹ thuật mã khối với khoá phiên Km để được bản mã M.

- Bản mã M được lấy làm đầu vào của thuật toán xác thực HMAC với một hàm băm và khoá xác thực Kx để thu được một giá trị digest làm bằng chứng xác thực.

- Dữ liệu xác thực digest được gắn với dữ liệu mã M (phía trước hoặc phía sau) để gửi cho bên nhận B.



Hình 1.9: Sơ đồ cài đặt dịch vụ bí mật, xác thực cho gói IP

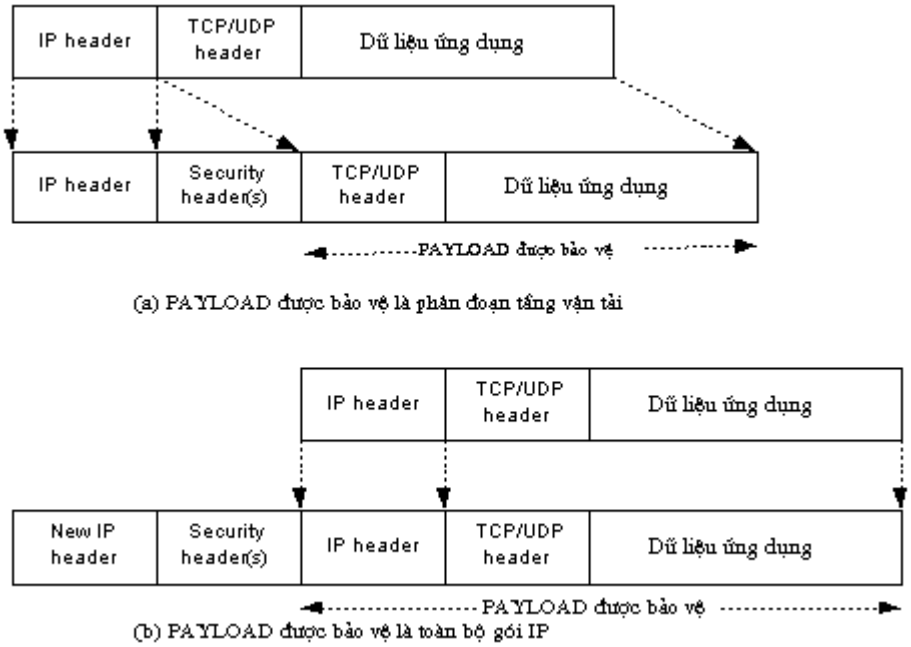
Bên nhận B:

- Đầu tiên dữ liệu mã M được lấy làm đầu vào của thuật toán xác thực HMAC với hàm băm và khoá xác thực giống như thuật toán mà bên gửi A đã sử dụng. Kết quả ra là digest1 được so sánh với giá trị digest đi kèm với dữ liệu mã M do A gửi tới. Nếu hai giá trị trùng nhau thì B khẳng định rằng dữ liệu mã M được xác thực, từ đó R cũng được xác thực. Nếu hai giá trị trên không giống nhau thì B khẳng định rằng hoặc là R bị mạo danh hoặc bị thay đổi trên đường truyền.

- Nếu R được xác thực, B sẽ dịch M bởi thuật toán mật mã và khoá do A và B đã thoả thuận để thu được bản rõ R.

1.3.4 Kỹ thuật đóng gói trong việc bảo vệ gói IP

Một gói IP bao gồm IP header và một phần tải (payload) theo sau. Payload có thể là một phân đoạn tầng vận tải hoặc các thông báo ICMP, IGMP. IP Header cung cấp các thông tin như địa chỉ nguồn, địa chỉ đích giúp cho việc giao nhận và định tuyến dữ liệu qua liên mạng để tới đích. Kỹ thuật đóng gói (Encapsulation techniques) cho phép thu được IP payload và bảo vệ nó bằng các dịch vụ an toàn như bí mật, xác thực, toàn vẹn.



Hình 1.10: Kỹ thuật đóng gói trong bảo vệ gói IP

Hình 1.10 mô tả kỹ thuật đóng gói trong việc bảo vệ gói IP. Như ở phần trên đã trình bầy, các thành phần của gói IP cần can thiệp mật mã có thể là một số thành phần nào đó hoặc toàn bộ gói IP. Chính vì vậy IP payload trong một gói IP đã được cài đặt dịch vụ an toàn có thể là một gói IP khác khi ta muốn bảo vệ toàn bộ gói IP hoặc có thể chỉ là phân đoạn tầng vận tải của chính gói IP đó. Trong tất cả các trường hợp đó, payload cần bảo vệ được gắn một hoặc nhiều header an toàn vào phía trước. Các header an toàn chứa các thông tin về các liên kết an toàn SA đã được thiết lập giữa hai hệ thống, các giá trị được sinh bởi mã xác thực thông báo, hàm băm để làm bằng chứng xác thực.

Trong hình 1.10a, payload được bảo vệ là phân đoạn tầng vận tải hoặc thông báo ICMP. Trong hình 1.10b, payload được bảo vệ là toàn bộ gói IP, trong trường hợp này phải bổ xung một IP header mới vào phía trước gói IP ban đầu. Giải pháp này được dùng với mô hình hai mạng LAN kết nối với nhau qua hai Gateway và dịch vụ an toàn được cài đặt tại hai Gateway. Kỹ thuật tóm lược được dùng khi can thiệp mật mã để bảo vệ IP payload khi nó được truyền qua một mạng không an toàn.

1.4 MÔ HÌNH CHỨC NĂNG CỦA HỆ THỐNG BẢO VỆ GÓI IP DÙNG KỸ THUẬT MẬT MÃ

1.4.1 Liên kết an toàn trong hệ thống bảo vệ gói IP

1.4.1.1 Khái niệm về liên kết an toàn

Trong môi trường Internet, các giao dịch giữa các máy được bảo vệ bởi các dịch vụ an toàn với các thuật toán mã hoá, thuật toán xác thực và khoá khác nhau. Chính vì vậy giữa chúng phải có các thoả thuận từ trước về cách thức bảo vệ gói IP và các thông tin bí mật chia sẻ. Trong các thuộc tính cần thoả thuận giữa hai hệ thống, có những thuộc tính có thể thoả thuận từ trước và công khai như thuật toán mã, thuật toán xác thực. Những thoả thuận này được áp dụng cho các gói IP được trao đổi giữa hai hệ thống. Nhưng có một thuộc tính thường không thoả thuận từ trước mà phải được trao đổi định kỳ theo một cách thức an toàn đó là khoá mật mã. Kỹ thuật mật mã dùng để bảo vệ gói IP thường là kỹ thuật mã khối. Khoá phiên cho kỹ thuật mã khối cần được trao đổi định kỳ. Việc thoả thuận từ trước các thuộc tính trên không làm ảnh hưởng đến độ an toàn của các dịch vụ được cài đặt. Ví dụ dưới đây chỉ ra một thoả thuận được dùng để giữ bí mật gói IP:

- Dùng kỹ thuật mã khối IDEA ở chế độ CFB với một véc tơ khởi tạo cho trước để mã hoá phân đoạn tầng vận tải (bao gồm header tầng vận tải và dữ liệu ứng dụng)

- Ngầm định khoá phiên ban đầu là một xâu 128 bít cho trước. Định kỳ khoá sẽ được thay đổi bằng một giao thức quản lý khoá.

Trong trường hợp này, tại máy nguồn khi một gói IP được chuyển tới, phân đoạn tầng vận tải sẽ được mã hoá bằng thuật toán IDEA ở chế độ CFB với véc tơ

khởi tạo và khoá phiên cho trước. Tiếp sau đó, định kỳ khoá phiên sẽ được thay đổi và các gói IP sẽ được bảo vệ bằng khoá mật mã mới. Như vậy vấn đề còn lại chỉ là việc thiết kế giao thức trao đổi khoá. Tuy nhiên trong một môi trường mở như Internet, người ta mong muốn một máy tính có thể thiết lập các giao dịch an toàn với nhiều máy khác nhau, với các dịch vụ an toàn khác nhau và các thuật toán mật mã khác nhau. Như vậy là để mỗi máy trong hệ thống an toàn xác định được giải pháp bảo vệ gói IP đến thì gói IP đó phải chứa những thông tin tham chiếu đến các thoả thuận về cài đặt dịch vụ an toàn. Các thông tin thoả thuận giữa hai máy trong hệ thống an toàn được gọi là liên kết an toàn. Từ đó chúng ta có khái niệm về liên kết an toàn SA (Security associate) :

Một liên kết an toàn giữa hai máy tính là một tập các thông tin có quan hệ với nhau được thống nhất giữa hai hoặc nhiều máy tính để thiết lập phiên truyền thông an toàn.

Để hai máy tính có thể truyền thông an toàn với nhau thì chúng phải có những thoả thuận với nhau về các liên kết an toàn SA như:

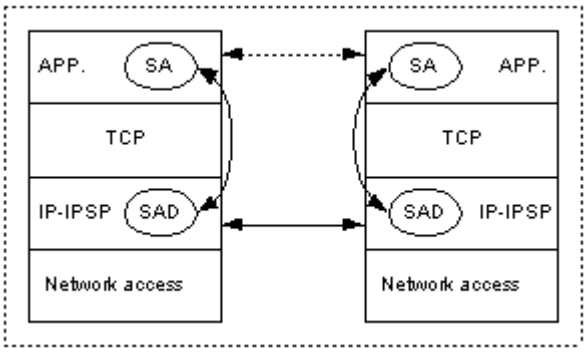
- Các dịch vụ và cơ chế an toàn được cài đặt.
- Các thuật toán mã hoá và thuật toán xác thực.
- Các khoá mật mã.

Liên kết an toàn được dùng để tạo ra một cách thức bảo vệ một dãy dữ liệu được truyền, có thể là tất cả dữ liệu được truyền trong một kết nối, hoặc tất cả các dữ liệu được truyền giữa hai hệ thống trong một khoảng thời gian nào đó... Các mục thông tin được chứa trong một liên kết an toàn được gọi là các thuộc tính (attributes) của liên kết an toàn đó. Các thuộc tính bao gồm hai loại là thuộc tính tĩnh (static) và thuộc tính động (dynamic). Các thuộc tính tĩnh có thể chứa cùng một giá trị trong suốt thời gian tồn tại của liên kết an toàn. Thuộc tính động có thể thay đổi giá trị, chẳng hạn như một chỉ số nguyên được tăng liên tiếp mỗi khi một gói số liệu được truyền nhằm chống lại việc dùng lại gói dữ liệu đã được bảo vệ hoặc khoá mật mã đã được sử dụng.

1.4.1.2 Mối quan hệ giữa liên kết an toàn và giao thức an toàn tầng IP

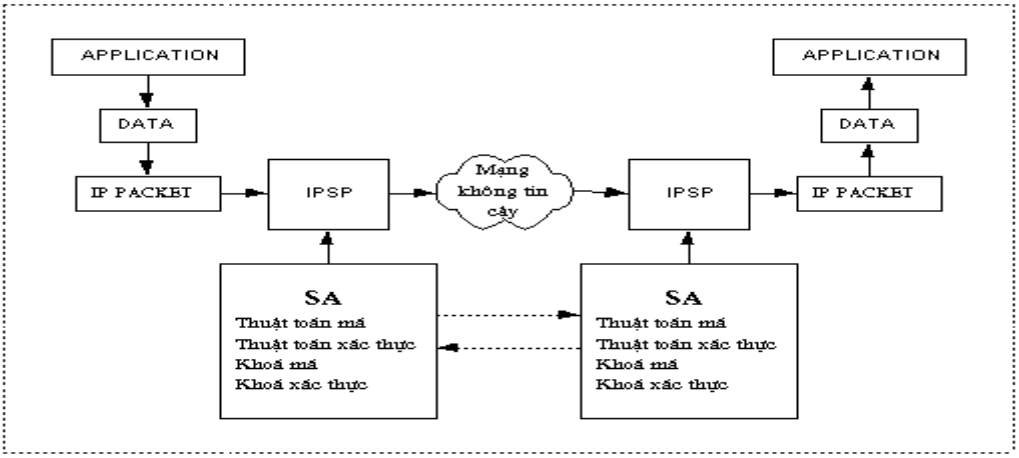
Một liên kết an toàn có một thời gian sống (lifetime) nhất định. Khi thiết lập một liên kết an toàn, các giá trị của các thuộc tính tĩnh và các giá trị khởi tạo của các thuộc tính động được xác định. Để thiết lập một liên kết an toàn giữa hai hệ thống, cần phải có một giao thức mật mã được thực hiện trực tiếp giữa hai hệ

thống đó hoặc thông qua một máy chủ an toàn tin cậy. Giao thức mật mã có thể được thực hiện ở một tầng khác trong cấu trúc hệ thống mở. Chẳng hạn để thiết lập một liên kết an toàn cho các dịch vụ an toàn ở tầng IP, chúng ta phải có một giao thức mật mã được thực hiện ở tầng ứng dụng. Một hệ thống bất kỳ có thể có nhiều liên kết an toàn được kích hoạt đồng thời. Cũng có thể có nhiều liên kết an toàn khác nhau giữa cùng một cặp hệ thống đầu cuối. Điều này xảy ra khi các các giao dịch khác nhau giữa hai hệ thống đầu cuối đòi hỏi các cách thức bảo vệ khác nhau. Ví dụ, các liên kết an toàn khác nhau có thể được dùng cho các giao dịch khác nhau, trong đó giao dịch thư tín điện tử thông thường chỉ cần giữ bí mật và các giao dịch kinh doanh chỉ đòi hỏi xác thực. Để tránh sự nhầm lẫn giữa các liên kết an toàn được sử dụng cho các giao dịch khác nhau giữa hai hệ thống, cần có các thuộc tính để xác định duy nhất một liên kết an toàn. Trong hệ thống bảo vệ gói IP, các thuộc tính để xác định một liên kết an toàn cho một gói là địa chỉ IP của máy đích và chỉ số tham số an toàn SPI (Security Paramaters Index). Nói chung việc thoả thuận các SA được thực hiện bởi các giao thức ở tầng ứng dụng, các thông tin về các SA sẽ được quản lý tại cơ sở dữ liệu ứng dụng SAD ở tầng IP.



Hình 1.11: Liên kết an toàn trong cấu trúc TCP/IP

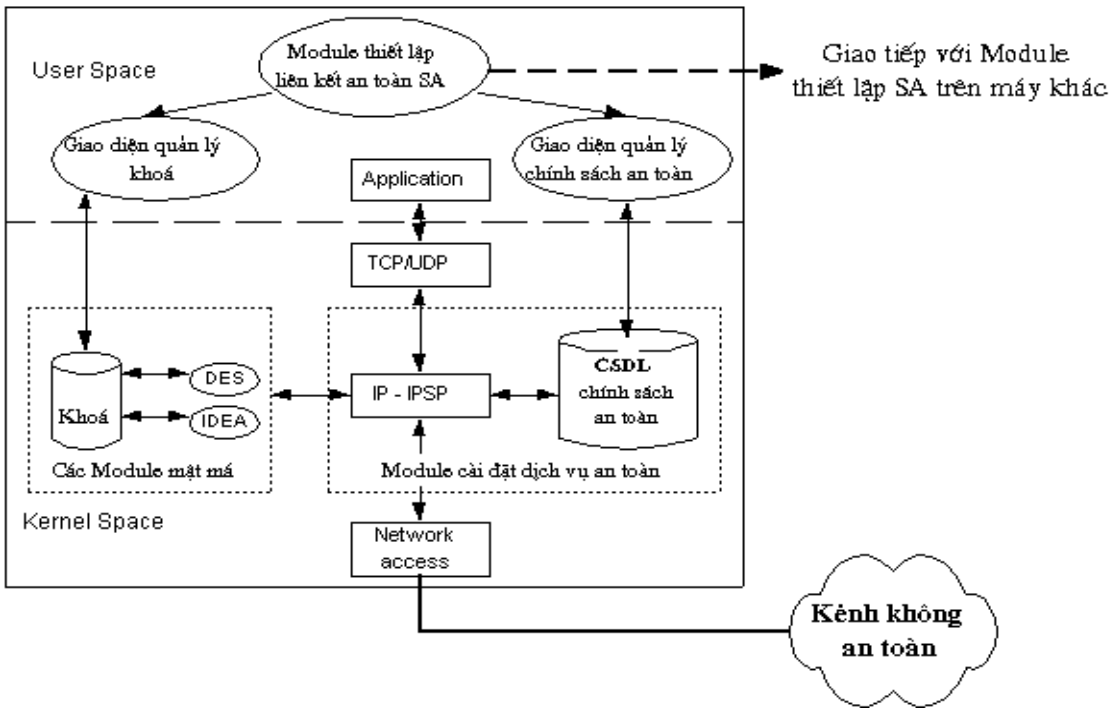
Hình 1.11 mô tả mối quan hệ giữa các giao thức thiết lập liên kết an toàn SA và giao thức an toàn ở tầng IP . Yêu cầu đối với giao thức thiết lập liên kết an toàn là phải tạo ra các SA tin cậy. Hai máy tính phải được xác thực lẫn nhau, các thông tin về SA phải được an toàn.



Hình 1.12: Thiết lập liên kết an toàn trong hệ thống bảo vệ gói IP

Hình 1.12 mô tả việc thiết lập các liên kết an toàn trong hệ thống bảo vệ gói IP. Gói IP được tạo từ các ứng dụng của máy nguồn, được cài đặt các dịch vụ an toàn bởi giao thức IPSP. Để hai máy có thể làm việc với nhau theo giao thức IPSP, giữa chúng phải thoả thuận các liên kết an toàn SA, gồm các thông tin như thuật toán mã, thuật toán xác thực, khoá mã, khoá xác thực, ...

1.4.2 Mô hình chức năng của hệ thống bảo vệ gói IP bằng kỹ thuật mật mã



Hình 1.13: Mô hình chức năng của hệ thống bảo vệ gói IP

Hình 1.13 là mô hình chức năng của hệ thống bảo vệ gói IP bằng kỹ thuật mật mã. Các thành phần của mô hình bao gồm:

- Module thiết lập liên kết an toàn: Có chức năng thoả thuận các liên kết an toàn SA giữa hai hệ thống bao gồm các thông tin như dịch vụ an toàn cần cài đặt, thuật toán mã hoá, thuật toán xác thực, các khoá mã và khoá xác thực... Việc quản lý khoá bao gồm các công việc như tính toán, lưu trữ, phân phối, thay đổi, huỷ khoá ... .Liên kết an toàn có thể được tạo bằng phương pháp thủ công, hoặc phương pháp tự động. Module này có hai giao diện chính là Giao diện quản lý khoá và Giao diện quản lý chính sách an toàn. Giao diện quản lý khoá thường là các file chứa các thông tin về khoá. Để tiến hành các thao tác mật mã được nhanh chóng, một bản sao của các khoá mật mã sẽ được lưu trong một vùng nhớ đệm của nhân hệ điều hành.

- Các module mật mã: Bao gồm các module mã hoá như IDEA, Blowfish và các hàm xác thực như SHA, MD5 ... được dùng để bảo vệ các thành phần của gói IP.

- Module cài đặt dịch vụ an toàn: Có chức năng chặn bắt, mã hoá, biến đổi xác thực và đóng gói lại gói IP dựa theo đặc trưng của các dịch vụ an toàn. Cách thức xử lý gói IP được điều khiển bởi người quản trị hệ thống thông qua "Giao diện quản lý chính sách an toàn". Một "Cơ sở dữ liệu chính sách an toàn" sẽ quyết định thao tác đối với gói IP như không mã, huỷ, hoặc gửi tới các module mật mã để cài đặt dịch vụ an toàn. Danh sách điều khiển truy nhập được quản lý bởi các công cụ mức người dùng.

Nguyên tắc hoạt động của mô hình như sau:

Đầu tiên, hai hệ thống cần thoả thuận với nhau để có các liên kết an toàn SA. Việc thoả thuận này có thể thực hiện bằng thủ công hoặc tự động. Trong phương pháp thủ công, hai bên thoả thuận trước với nhau về cách thức bảo vệ gói IP, các khoá mã và khoá xác thực sẽ được trao đổi với nhau theo một kênh an toàn. Với phương pháp tự động, hai hệ thống dùng một giao thức mật mã để thoả

thuận về các liên kết an toàn SA. Các liên kết an toàn cũng sẽ được thay đổi định kỳ một cách tự động.

Gói IP chứa dữ liệu ứng dụng được chuyển tới tầng IP có cài đặt giao thức IPSP. Tại đây, căn cứ vào cơ sở dữ liệu chính sách an toàn, hệ thống sẽ có những thao tác phù hợp đối với gói IP (chuyển đi không mã, huỷ, cài đặt dịch vụ an toàn,...). Những gói IP cần cài đặt dịch vụ an toàn sẽ được xử lý bởi "Module cài đặt dịch vụ an toàn" và " Các Module mật mã" bao gồm các thao tác như mã hoá, chèn header an toàn, chèn dữ liệu xác thực,... sau đó được chuyển xuống tầng truy nhập mạng để đi ra kênh công khai. Tại nơi nhận, gói IP lại được xử lý bằng kỹ thuật mật mã theo trình tự ngược lại.

1.4.3 Những yếu tố ảnh hưởng đến độ an toàn của hệ thống bảo vệ gói IP

Cũng như những hệ thống an toàn dùng kỹ thuật mật mã nói chung, độ an toàn của hệ thống bảo vệ gói IP phụ thuộc vào các yếu tố sau:

- Độ an toàn của các thuật toán mã hoá và xác thực dữ liệu.
- Độ an toàn các giao thức thiết lập liên kết an toàn.
- Độ an toàn của hệ điều hành mạng được cài đặt phần mềm bảo vệ gói IP.

1.4.3.1 Độ an toàn của các thuật toán mã hoá và xác thực dữ liệu

Chúng ta lấy cơ chế bảo vệ gói IP trong chuẩn IPSEC làm ví dụ minh hoạ. Hình 1.9 là sơ đồ bảo vệ gói IP bằng kỹ thuật mật mã trong chuẩn IPSEC. Đầu tiên A mã thông báo R bằng khoá Km để được bản mã M, sau đó tính mã xác thực $HMAC_{Kx}(M)$ với một hàm băm và khoá Kx trên bản mã M. Tại nơi nhận, đầu tiên B thẩm tra tính xác thực của M (và cũng là tính xác thực của R) bằng cách tính mã xác thực $HMAC_{Kx}(M)$ để so sánh với dữ liệu xác thực đi kèm với M, nếu M được xác thực thì B giải mã M để thu được R. Trong trường hợp M không được xác thực, gói IP sẽ bị huỷ.

a. Độ an toàn của thuật toán mã hoá

Trong hệ thống bảo vệ gói IP, thuật toán mã hoá được dùng để giữ bí mật các thành phần của gói IP như phân đoạn tầng vận tải hoặc toàn bộ gói IP. Chúng thường là các thuật toán mã khối như DES, IDEA, Blowfish,... Trong giao thức TCP/IP, dữ liệu cần trao đổi của các ứng dụng được chia thành các khối nhỏ trong các gói IP và được chuyển từ máy nguồn tới máy đích. Trong quá trình đó,

kẻ tấn công dễ dàng thu được các gói IP và khôi phục lại toàn bộ dữ liệu ban đầu như nội dung của thư điện tử, nội dung file dữ liệu,... Khi các thành phần gói IP được mã hoá, kẻ tấn công tìm cách thu được dữ liệu ứng dụng và thông tin điều khiển trong từng gói IP. Để làm được điều này, kẻ tấn công cố gắng xác định được khoá đang được sử dụng để tìm ra dữ liệu rõ từ dữ liệu mã hoặc thu được dữ liệu rõ từ dữ liệu mã. Độ an toàn của thuật toán mã hoá đặc trưng cho mức độ khó khăn của kẻ tấn công trong việc tìm ra khoá hoặc dữ liệu rõ từ dữ liệu mã. Để khôi phục toàn bộ dữ liệu ứng dụng từ các gói IP, kẻ tấn công phải thu được các khối dữ liệu ở dạng rõ và cả giá trị của cổng ứng dụng trong header tầng vận tải. Chính vì vậy, khi bảo vệ gói IP chúng ta thường mã hoá cả header tầng vận tải. Các kỹ thuật mã khối được dùng để mã hoá các thành phần của gói IP phải đảm bảo rằng kẻ tấn công không thể tìm ra được khoá hoặc dữ liệu rõ từ dữ liệu mã.

b. Độ an toàn của thuật toán xác thực

Để xác thực các thành phần của gói IP trong chuẩn IPSEC, thuật toán HMAC phải đảm bảo rằng:

- Với giá trị m bất kỳ của dữ liệu xác thực không thể tìm ra x để $HMAC_K(x) = m$.
- Với dữ liệu x bất kỳ, không thể tìm ra y khác x để $HMAC_K(y) = HMAC_K(x)$.
- Không thể tìm ra cặp (x,y) thoả mãn $HMAC_K(x) = HMAC_K(y)$.

2.4.3.2 Độ an toàn của giao thức thiết lập liên kết an toàn

Các giao thức thiết lập liên kết an toàn nhằm thoả thuận các tham số an toàn như các thuật toán mã hoá, thuật toán xác thực, các khoá phiên,... Các tham số trên phải được trao đổi một cách an toàn, nghĩa là chúng phải được xác thực, bí mật, toàn vẹn tuỳ theo đặc trưng thông tin. Trong nhiều trường hợp, giao thức phân phối khoá đóng vai trò là một giao thức thiết lập liên kết an toàn.

Các tấn công đối với một giao thức thiết lập liên kết an toàn là tấn công dùng lại (replay) , từ chối dịch vụ (Denial of service), người đàn ông ở giữa (Man in the middle), cướp kết nối (Connection hijacking)...

Để chống lại các tấn công trên, kỹ thuật mật mã đóng một vai trò rất quan trọng. Việc đánh giá độ an toàn của một giao thức phải dựa trên sự tổng hợp của các yếu tố như các tấn công, độ an toàn của kỹ thuật mật mã, tính đúng đắn của giao thức,...

Chẳng hạn, trong một giao thức phân phối khoá phiên, các giá trị nonce đã được sử dụng để chống lại kiểu tấn công dùng lại. Tuy nhiên sự có mặt của tham số nonce chỉ có ý nghĩa khi chúng được bảo vệ bằng các kỹ thuật mật mã.

Một giao thức thiết lập liên kết an toàn phải có khả năng xác thực lẫn nhau giữa hai máy đang trao đổi thông tin. Để giải quyết vấn đề này, cơ chế chữ ký số thường được sử dụng. Trong IKE, chữ ký số RSA đã được sử dụng để xác thực hai máy thực hiện giao thức thoả thuận khoá Diffie-Hellman nhằm chống lại tấn công "người đàn ông ở giữa ".

2.4.3.3 An toàn hệ thống

An toàn hệ thống nhằm ngăn chặn các truy nhập trái phép và các tấn công phá hoại vào hệ thống. Trong hệ thống IPSEC có một số tham số mật mã cần phải bảo vệ như cơ sở dữ liệu khoá công khai, khoá bí mật, khoá phiên,...

Để dùng chữ ký số RSA, cơ sở dữ liệu khoá công khai và khoá bí mật được lưu giữ trên ổ cứng của nút bảo vệ gói IP. Việc bảo vệ cơ sở dữ liệu khoá công khai nhằm chống lại các sửa đổi trái phép các khoá mã công khai, sao chép khoá dịch bí mật.

Trong hệ thống này cơ sở dữ liệu khoá công khai được bảo vệ thông qua cơ chế an toàn của hệ điều hành mạng. Chỉ người quản trị mới có quyền truy nhập vào cơ sở dữ liệu khoá. Tuy nhiên nếu kẻ tấn công biết mật khẩu của người quản trị mạng và tiếp cận trực tiếp với nút bảo vệ gói IP thì họ có thể truy nhập đến cơ sở dữ liệu khoá công khai. Chính vì vậy, các nút bảo vệ gói IP cần được bảo vệ bằng các phương pháp vật lý, chỉ những người có trách nhiệm mới được vào phòng đặt thiết bị bảo vệ gói IP. Ngoài ra, để ngăn chặn các tấn công bất hợp pháp từ xa vào các máy chủ dựa vào các lỗ hổng an toàn cần tham khảo các tài liệu về các biện pháp nhằm nâng cao độ an toàn cho các hệ điều hành mạng.

CHƯƠNG II
CƠ CHẾ QUẢN LÝ DỮ LIỆU CỦA GIAO THỨC TCP/IP TRÊN SOLARIS

2.1 GIỚI THIỆU VỀ LUỒNG (STREAMS) TRONG SOLARIS

2.1.1 Khái niệm về luồng

Luồng (STREAMS) là phần bổ xung mới đây tới kiến trúc của nhân (kernel) UNIX. Chúng được phát triển bởi Dennis Ritchie tại phòng thí nghiệm BELL vào giữa những năm 1980 và lần đầu tiên xuất hiện trong phiên bản thương mại của UNIX với phiên bản 3. STREAMS được thiết kế để giải quyết một vài hạn chế của mô hình SOCKET, đặc biệt trong lĩnh vực mạng và truyền thông. Cốt lõi của mô hình STREAMS là nó được cài đặt giống như chồng giao thức. Một chồng STREAMS hay còn gọi là một *luồng* (stream) bao gồm một trình điều khiển luồng ở đáy để điều khiển giao diện với phần cứng, không có hoặc có một số mô đun (module) tương ứng các mức giao thức khác nhau và một đầu luồng (stream head) điều khiển giao diện giữa luồng và tiến trình người dùng (user process). Để tiện theo dõi chúng ta phân biệt hai từ "STREAMS" và "stream". Từ "STREAMS" chỉ đến mô hình hay cơ chế STREAMS nói chung, còn từ "stream" chỉ đến một luồng thông tin cụ thể trong một hệ thống dùng cơ chế STREAMS. Dữ liệu từ tiến trình của người dùng đi xuôi (flows downstream) từ đầu luồng qua các mô đun giao thức được xếp chồng để tới trình điều khiển. Cũng như vậy, dữ liệu nhận đi ngược (flows upstream) từ trình điều khiển đi qua các các mô đun của luồng để tới đầu luồng và từ đó tới tiến trình người dùng. Khi dữ liệu đi qua các mô đun, những thao tác xử lý giao thức cần thiết được tiến hành. Cùng với dữ liệu các thông tin điều khiển cũng có thể đi ngược hoặc xuôi trong một luồng. STREAMS là một cách thức mềm dẻo và là một tập các công cụ để phát triển các dịch vụ truyền thông hệ thống UNIX. Nó hỗ trợ việc cài đặt các các bộ giao thức truyền thông và các trình điều khiển thiết bị đơn lẻ. STREAMS định nghĩa một giao diện chuẩn vào ra ký tự (input/output character) trong nhân (kernel) và giữa nhân với phần còn lại của hệ thống UNIX. Cơ chế liên kết (associated mechanism) là đơn giản và có tính mở. STREAMS chứa một tập các lời gọi hệ thống (system calls), tài nguyên nhân (kernel resources) và trình nhân (kernel routines). Giao diện và cơ chế chuẩn của STREAMS cho phép phát triển các ứng dụng theo mô đun và có thể chuyển từ hệ thống này sang hệ thống khác, dễ tích hợp các dịch vụ hiệu năng cao và các thành phần của chúng. Giao diện người dùng STREAMS luôn

hướng đến việc tương thích với các hàm vào ra ký tự mức người dùng như open(), close(),read(),write(),ioctl().

Một luồng (stream) là một đường dẫn truyền dữ liệu và xử lý hai chiều giữa một trình điều khiển STREAMS (STREAMS driver) trong nhân và một tiến trình trong không gian người dùng. Trong nhân, một luồng được cấu trúc bởi việc liên kết một đầu luồng, một trình điều khiển (driver) , không có hoặc có một số mô đun giữa đầu luồng và trình điều khiển. Đầu luồng là phần của luồng gần nhất với tiến trình người dùng. Tất cả các lời gọi hệ thống được tạo bởi một tiến trình người dùng ở phía trên một luồng được xử lý bởi đầu luồng.

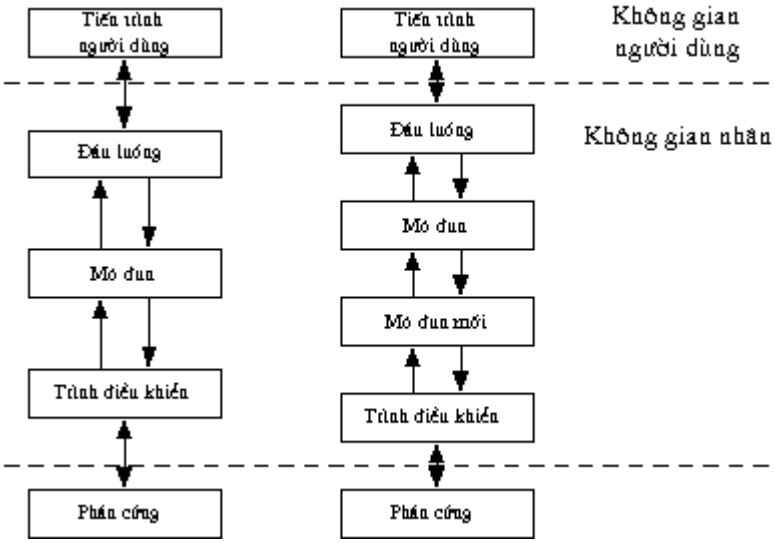
Một trình điều khiển STREAMS (STREAMS driver) là một trình điều khiển thiết bị cung cấp các dịch vụ của thiết bị vào ra, hoặc một trình điều khiển phần mềm, được coi như một trình điều khiển thiết bị ảo (pseudo-device driver). Trình điều khiển truyền dữ liệu giữa nhân và thiết bị, chuyển đổi giữa những cấu trúc dữ liệu được dùng bởi cơ chế STREAMS (STREAMS mechanism) và những cấu trúc dữ liệu mà thiết bị có thể hiểu được.

Một mô đun STREAMS (STREAMS module) đại diện cho những chức năng xử lý được tiến hành trên dữ liệu chuyển qua luồng . Một mô đun là một tập các trình (routine) mức nhân và các cấu trúc dữ liệu được dùng để xử lý dữ liệu, thông tin trạng thái và thông tin điều khiển. Việc xử lý dữ liệu có thể là thay đổi cách biểu diễn dữ liệu, thêm hoặc xoá thông tin đầu và thông tin cuối của dữ liệu, nén và giải nén dữ liệu,..

Thông tin điều khiển và thông tin trạng thái bao gồm các thông tin điều khiển vào ra, các tín hiệu. Mỗi mô đun đều liên kết chặt chẽ với các thành phần khác trong stream.

Một mô đun truyền thông với các thành phần ngay sát nó (mô đun khác, đầu luồng, trình điều khiển) bằng việc chuyển các thông báo. Mô đun không là một thành phần bắt buộc của STREAMS. Một hoặc nhiều mô đun có thể được chèn vào giữa đầu luồng và trình điều khiển. Các mô đun STREAMS được kết nối động vào một luồng bằng một tiến trình người dùng và không đòi hỏi phải lập trình nhân (kernel programming). STREAMS dùng các cấu trúc hàng đợi (queue) để giữ thông tin về một mô đun được chèn vào hoặc một trình điều khiển STREAMS được mở. Một hàng đợi là một cấu trúc dữ liệu chứa thông tin trạng thái, một con trỏ chỉ tới các thủ tục để xử lý các thông báo và các con trỏ để quản lý luồng. Các hàng đợi luôn luôn được cấp phát theo từng cặp, một hàng đợi đọc

và một hàng đợi viết. Có một cặp hàng đợi cho mỗi mô đun và trình điều khiển, một cặp cho đầu luồng. Các cặp hàng đợi được cấp bất cứ khi nào luồng được mở hoặc mô đun được thêm vào luồng.



Hình 2.1 : Mô hình STREAMS

Dữ liệu được chuyển giữa một trình điều khiển và đầu luồng và giữa các mô đun trong dạng các thông báo (message). Một thông báo là một tập các cấu trúc dữ liệu được dùng để chuyển các dữ liệu, thông tin trạng thái và thông tin điều khiển giữa các tiến trình người dùng, các mô đun và các trình điều khiển. Thông báo được chuyển từ đầu luồng tới trình điều khiển hoặc từ một tiến trình đến thiết bị được gọi là "dòng xuống" (downstream). Tương tự các thông báo được chuyển theo hướng ngược lại từ thiết bị đến tiến trình hoặc từ trình điều khiển đến đầu luồng được gọi là "dòng lên" (upstream).

Một thông báo STREAMS bao gồm một hoặc nhiều khối thông báo (block). Mỗi khối là một bộ ba bao gồm một "đầu" (header), một khối dữ liệu (data block), và một vùng nhớ đệm dữ liệu (data buffer). Đầu luồng truyền dữ liệu giữa không gian dữ liệu (data space) của một tiến trình người dùng và không gian dữ liệu nhân STREAMS.Dữ liệu được gửi tới một trình điều khiển từ một tiến trình người dùng được đóng gói thành các thông báo STREAMS và được chuyển xuống. Khi thông báo chứa dữ liệu được chuyển lên đầu luồng, thông báo được xử lý bởi đầu luồng và dữ liệu được sao chép tới vùng nhớ đệm người dùng

(user buffer). Trong một luồng, các thông báo được phân biệt bởi chỉ số kiểu. Có những kiểu thông báo được gửi lên đầu luồng để yêu cầu đầu luồng tiến hành các thao tác đặc biệt như gửi một tín hiệu tới một tiến trình người dùng. Các kiểu thông báo khác chỉ có vai trò chuyển thông tin trong một luồng và không được thao tác trực tiếp bởi tiến trình người dùng.

2.1.2 Các thao tác trên luồng

Một trình điều khiển STREAMS tương tự như một trình điều khiển vào ra ký tự truyền thống, nó có một hoặc nhiều đầu mối (notes) được kết hợp với nó trong hệ thống file và được truy nhập dùng lời gọi hệ thống open(). Mỗi đầu vào hệ thống file tương ứng với một thiết bị minor riêng rẽ của trình điều khiển. Việc mở các thiết bị minor của các trình điều khiển tạo ra các luồng riêng rẽ được kết nối giữa tiến trình người dùng và trình điều khiển. Một mô tả file (file description) được trả lại bởi lời gọi *open* được dùng để truy nhập tới luồng.

Khi thiết bị được mở, một tiến trình người dùng có thể gửi dữ liệu tới thiết bị dùng lời gọi hệ thống write() và nhận dữ liệu từ các thiết bị dùng lời gọi hệ thống read(). Việc truy nhập luồng dùng các lời gọi hệ thống read và write tương thích với cơ chế vào ra ký tự truyền thống. Lời gọi hệ thống close() đóng một thiết bị và huỷ một luồng được liên kết bởi lần gọi open cuối cùng. Điều khiển luồng (flow control) là một cơ chế STREAMS điều khiển tốc độ các thông báo truyền qua các mô đun, trình điều khiển, đầu luồng và các tiến trình. Điều khiển luồng có tác động riêng rẽ đến luồng. Nó hạn chế số ký tự có thể xếp hàng để xử lý tại một hàng đợi bất kỳ trong luồng . Cơ chế này hạn chế các vùng nhớ đệm và việc xử lý tại các hàng đợi và trong một luồng bất kỳ. Điều khiển luồng không tác động đến các thông báo có quyền ưu tiên cao.

2.1.3 Các thành phần của luồng

2.1.3.1 Các hàng đợi (queue)

Một hàng đợi là một giao diện giữa một trình điều khiển STREAMS hoặc mô đun với phần còn lại của luồng. Mỗi thể hiện (instance) của một trình điều khiển được mở hoặc một mô đun được thêm vào có một cặp hàng đợi được cấp phát, một hàng đợi đọc (đón dữ liệu từ dưới lên) và một hàng đợi viết (đón dữ liệu từ trên xuống). Các hàng đợi luôn luôn được cấp phát như một cặp liền kề, tương tự như một mảng của các cấu trúc. Hàng đợi có địa chỉ thấp là hàng đợi đọc, hàng đợi có địa chỉ cao là hàng đợi viết.

Một thủ tục service của hàng đợi được cần đến để xử lý các thông báo trên hàng đợi. Nó chuyển các thông báo khỏi hàng đợi, xử lý chúng, và gọi thủ tục put của mô đun tiếp theo trong luồng để chuyển thông báo đã được xử lý tới hàng đợi tiếp theo.

Một thủ tục put của hàng đợi được cần đến bởi thủ tục put hoặc service để thêm một thông báo tới hàng đợi hiện tại. Nếu một mô đun không cần xử lý thông báo, thủ tục put của nó có thể gọi thủ tục put của hàng đợi liền kề.

Mỗi hàng đợi có một con trỏ chỉ tới trình (routine) open và close.

Trình open của trình điều khiển được gọi khi trình điều khiển lần đầu tiên được mở và mỗi lần luồng mở. Trình open của mô đun được gọi khi mô đun lần đầu tiên được thêm vào luồng và mọi lần mở của luồng. Trình close của mô đun được gọi khi mô đun bị xoá khỏi luồng. Trình close của trình điều khiển được gọi khi luồng bị tháo dỡ (dismantled).

2.1.3.2 Các thông báo (message)

Tất cả các đầu vào và đầu ra của STREAMS được dựa trên các thông báo. *Các đối tượng được chuyển giữa các mô đun STREAMS là các con trỏ chỉ tới các thông báo*. Tất cả các thông báo STREAMS dùng hai cấu trúc thông báo (msgb và datab) để chỉ tới dữ liệu thông báo (message data). Các cấu trúc dữ liệu này mô tả kiểu của thông báo và chứa các con trỏ chỉ tới dữ liệu của thông báo, cũng như các thông tin khác. Các thông báo được gửi qua luồng bằng việc gọi thủ tục put của mỗi mô đun hoặc trình điều khiển trong luồng.

a.Các kiểu thông báo

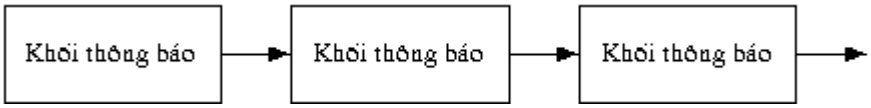
Tất cả các thông báo STREAMS được gán các kiểu thông báo xác định cách thức sử dụng chúng bởi mô đun và trình điều khiển và việc điều khiển chúng bởi đầu luồng. Một trình điều khiển hoặc một mô đun có thể gán hầu hết các kiểu tới các thông báo mà chúng sinh ra và một mô đun có thể thay đổi kiểu của các thông báo trong quá trình xử lý chúng. Đầu luồng sẽ chuyển đổi các lời gọi hệ thống nhất định tới các kiểu thông báo riêng và gửi chúng xuống, đồng thời đầu luồng đáp ứng các lời gọi khác bằng việc sao chép nội dung của các kiểu thông báo được gửi lên.

Hầu hết các kiểu thông báo được trao đổi ở phạm vi bên trong STREAMS và chỉ có thể chuyển từ thành phần này tới thành phần khác của STREAMS. Một vài kiểu thông báo, ví dụ như M_DATA, M_PROTO, M_PCPROTO có thể

chuyển giữa một đầu luồng và các tiến trình người dùng. Các thông báo M_DATA chứa dữ liệu trong một luồng và giữa một đầu luồng và một tiến trình người dùng. Các thông báo M_PROTO hoặc M_PCPROTO chứa dữ liệu và các thông tin điều khiển.

Như chỉ ra trong hình 2.2, một thông báo STREAMS chứa một hoặc nhiều khối thông báo (message block) liên kết với nhau và được gắn với khối thông báo đầu tiên của cùng thông báo.

Các thông báo có thể tồn tại đơn lẻ (stand-alone) khi thông báo được xử lý bởi một thủ tục.

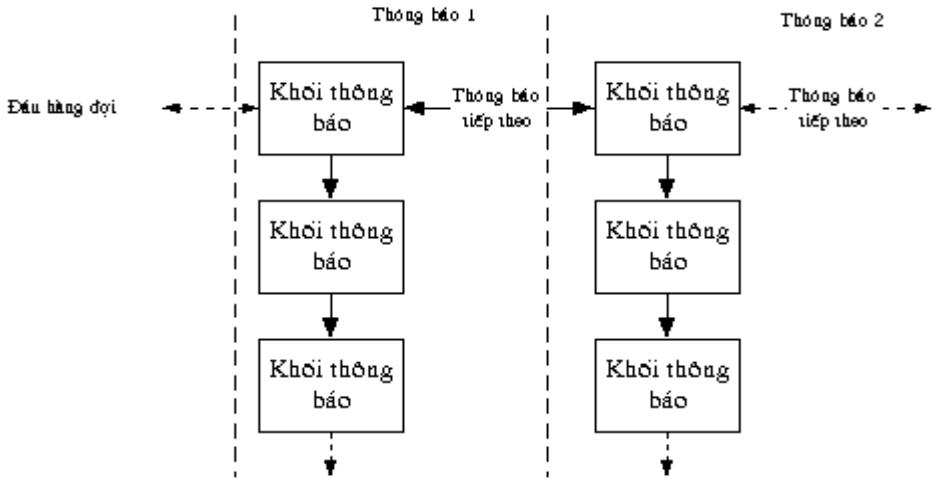


Hình 2.2: Một thông báo của luồng

Một cách luân phiên, một thông báo có thể đợi để được xử lý trên một danh sách liên kết của các thông báo được gọi là hàng đợi thông báo. Trong hình 2.3 thông báo 2 được liên kết với thông báo 1.

Khi một thông báo được xếp hàng trong một hàng đợi, khối đầu tiên của thông báo chứa các liên kết tới các thông báo trước và sau ở trên cùng một hàng đợi thông báo và một liên kết tới khối thông báo thứ hai của thông báo (nếu nó tồn tại). Đầu (head) và đuôi (tail) của hàng đợi thông báo được chứa trong hàng đợi. Các trình STREAMS (STREAMS routine) cho phép các nhà phát triển điều khiển các thông báo và hàng đợi thông báo.

b.Quyền ưu tiên xếp hàng thông báo:(Message queueing Priority)

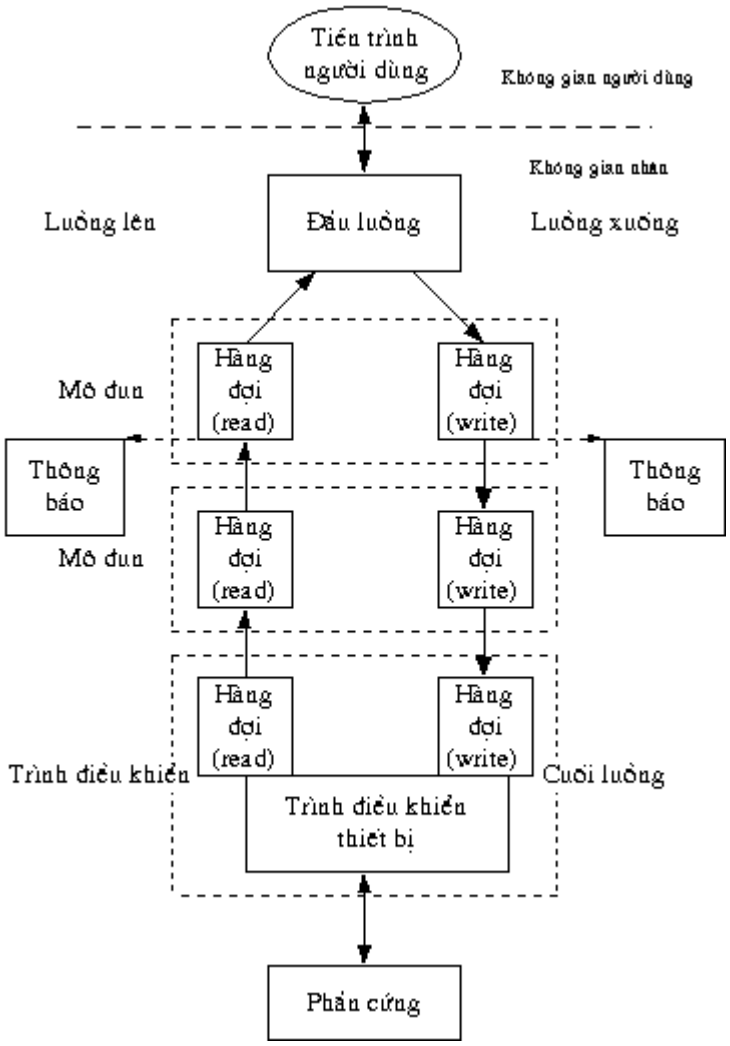


Hình 2.3: Các thông báo trên một hàng đợi thông báo

Trong những trường hợp nhất định, các thông báo chứa thông tin khẩn (urgent information) phải chuyển nhanh qua luồng. Để đáp ứng đòi hỏi này, STREAMS cung cấp nhiều lớp quyền ưu tiên . Tất cả các thông báo có một trường quyền ưu tiên được gán . Các thông báo bình thường (Normal message) có quyền ưu tiên là 0. Các thông báo ưu tiên có quyền lớn hơn 0. Các thông báo ưu tiên cao có quyền ưu tiên cao do kiểu thông báo của chúng. Theo quy ước, STREAMS ngăn chặn việc các thông báo ưu tiên cao bị khoá bởi điều khiển luồng và dùng thủ tục service để xử lý chúng trước tất cả các thông báo bình thường trên hàng đợi. Điều này giúp cho các thông báo ưu tiên cao đi qua các mô đun trong thời gian trễ tối thiểu.

Các thông báo bình thường (không ưu tiên) được đặt tại cuối hàng đợi sau tất cả các thông báo khác trong hàng đợi. Các thông báo ưu tiên có thể là các thông báo ưu tiên cao hoặc các thông báo băng ưu tiên (priority band). Các thông báo ưu tiên cao được đặt tại đầu của hàng đợi nhưng sau tất cả các thông báo ưu tiên cao khác đang tồn tại trong hàng đợi. Các thông báo băng ưu tiên cho phép cung cấp các dữ liệu khẩn được đặt trong hàng đợi sau các thông báo ưu tiên cao và trước các thông báo thông thường. Quyền ưu tiên thông báo được xác định bởi kiểu thông báo. Các kiểu thông báo ưu tiên cao không thể thay đổi tới các kiểu thông báo bình thường.

2.1.3.3 Các mô đun



Hình 2.4: Các cặp hàng đợi thông báo

Một mô đun tiến hành các phép biến đổi trung gian trên các thông báo chuyển giữa đầu luồng và trình điều khiển. Có thể không có hoặc có nhiều mô đun trong một luồng .

Mỗi mô đun được tạo từ một cặp các cấu trúc hàng đợi. Một hàng đợi tiến hành các thao tác với các thông báo từ trên xuống mô đun (Hàng đợi viết), hàng đợi khác tiến hành các thao tác với các thông báo từ dưới lên (Hàng đợi đọc). Mỗi hàng đợi trong một mô đun có các chức năng phân biệt bao gồm các thủ tục xử lý và dữ liệu không quan hệ với nhau. Các hàng đợi thao tác độc lập và hàng đợi này không biết thông báo được chuyển qua hàng đợi kia trừ khi được lập trình nhờ nhà phát triển. Mỗi hàng đợi có thể truy nhập trực tiếp tới hàng đợi liền

kề theo hướng của luồng thông báo. Tuy nhiên trong một mô đun, một hàng đợi có thể xác định được hàng đợi còn lại và truy nhập tới các thông báo và dữ liệu của hàng đợi đó. Mỗi hàng đợi trong mô đun có các con trỏ chỉ tới các thông báo, các thủ tục xử lý và dữ liệu :

Các thông báo: Các thông báo được kết nối động tới hàng đợi trên một danh sách liên kết khi chúng được chuyển qua mô đun.

Các thủ tục xử lý: Một thủ tục put xử lý các thông báo và phải là một bộ phận của mỗi hàng đọi. Một thủ tục Service cũng có thể là một bộ phận của mỗi hàng đợi. Các thủ tục có thể gửi thông báo lên hoặc xuống.

Dữ liệu: Ta có thể dùng một trường riêng trong hàng đợi để chỉ đến cấu trúc dữ liệu riêng.

2.1.3.4 Các trình điều khiển (driver)

Các trình điều khiển thiết bị STREAMS là một phần khởi tạo của STREAMS. Chúng được cấu trúc tương tự mô đun STREAMS. Có ba sự khác nhau cơ bản giữa các mô đun và trình điều khiển :

Một trình điều khiển phải điều khiển được các ngắt từ thiết bị, một trình điều khiển có thể có nhiều luồng kết nối với nó, một trình điều khiển được khởi tạo (initialized) và ngừng khởi tạo (deinitialized) thông qua open và close.

Một mô đun có thể khởi tạo bởi I_PUSH ioctl (hoặc open) và dừng khởi tạo qua I_POP ioctl (hoặc close).

Các trình điều khiển và các mô đun có thể chuyển các tín hiệu, các mã lỗi và các giá trị trả về tới các tiến trình thông qua các kiểu thông báo nhất định.

2.2 CƠ CHẾ QUẢN LÝ LUỒNG (STREAMS MECHANISM)

2.2.1 Giới thiệu về cơ chế quản lý luồng

Phần này chỉ ra việc tạo, sử dụng và tháo dỡ luồng bằng cách dùng các lời gọi hệ thống STREAMS như thế nào. Các lời gọi hệ thống nói chung và lời gọi hệ thống STREAMS nói riêng cung cấp cho người dùng khả năng (facilities) để tạo các chương trình ứng dụng. Giao diện lời gọi hệ thống này hướng đến sự tương thích với các khả năng vào ra ký tự truyền thống. Lời gọi hệ thống open nhận ra một file STREAMS và tạo một luồng tới một trình điều khiển xác định.

Một tiến trình người dùng có thể nhận và gửi dữ liệu trên các file STREAMS dùng các lời gọi hệ thống read và write trong cách thức tương tự như các file ký tự truyền thống.

Lời gọi hệ thống ioctl cho phép người dùng tiến hành các chức năng đối với các thiết bị đặc trưng. Các lệnh ioctl cung cấp một vài hàm truy nhập và điều khiển các luồng. Lời gọi close dùng để tháo dỡ luồng.

Ngoài các lệnh ioctl truyền thống và các lời gọi hệ thống, còn có các lời gọi hệ thống khác được dùng bởi STREAMS. Lời gọi hệ thống poll cung cấp cho người dùng cơ chế vào ra đa luồng (multiplexing) thông qua một tập các mô tả file. Các lời gọi hệ thống putmsg, getmsg, getpmsg, putpmsg cho phép người dùng gửi và nhận các thông báo STREAMS và phù hợp cho việc giao tiếp với các mô đun và trình điều khiển STREAMS thông qua một giao diện dịch vụ. STREAMS cung cấp các khả khả năng và tiện ích nhận để hỗ trợ việc phát triển các mô đun và trình điều khiển. Đầu luồng điều khiển hầu hết các lời gọi hệ thống xử lý các mô đun và trình điều khiển.

Các lời gọi hệ thống streams bao gồm:

open : mở một luồng

close : đóng một luồng

read : đọc dữ liệu từ luồng

write :viết dữ liệu tới luồng

ioctl:điều khiển luồng

getmsg:nhận một thông báo tại đầu luồng

getpmsg: nhận một thông báo đặc quyền tại đầu luồng

putmsg:gửi một thông báo xuống

putpmsg:gửi một thông báo lên

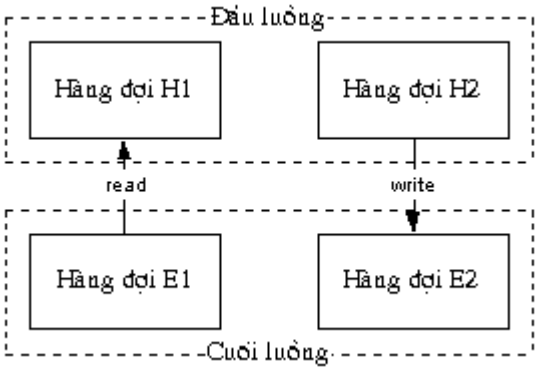
poll: xác định các file mà trên đó người dùng có thể gửi và nhận các thông báo hoặc các sự kiện nhất định đã xảy ra

pipe : tạo một kênh hai chiều cung cấp truyền thông giữa nhiều tiến trình.

2.2.2 Xây dựng luồng

Một luồng được tạo bởi một danh sách liên kết của các cấu trúc dữ liệu trong nhân. Một danh sách được tạo như một tập các hàng đợi liên kết. Cặp hàng

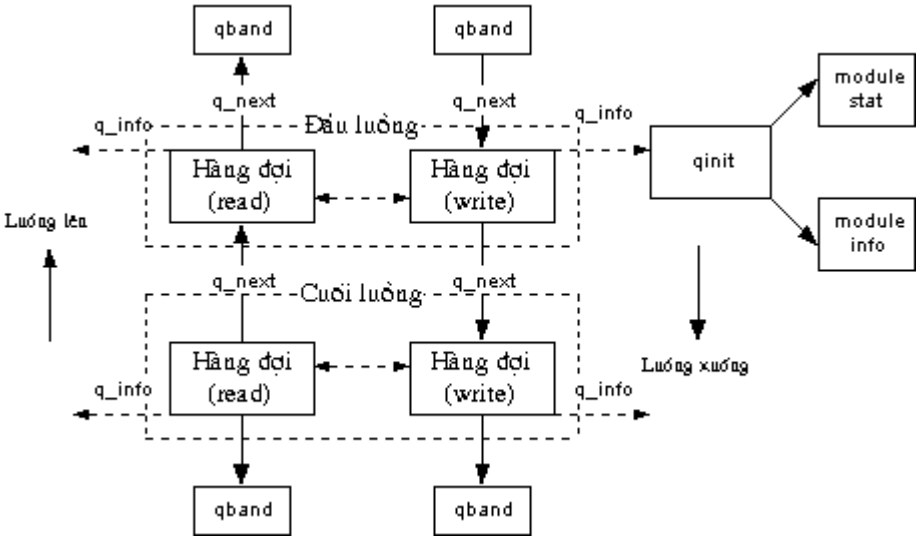
đợi đầu tiên là đầu của luồng và cặp hàng đợi thứ hai là phần cuối của luồng. Phần cuối của luồng biểu diễn một trình điều khiển thiết bị, trình điều khiển thiết bị ảo hoặc phần cuối của một pipe STREAMS. Các trình nhân (kernel routine) giao diện với đầu luồng để tiến hành các thao tác trên luồng. Hình 2.5 chỉ ra các cặp hàng đợi của một luồng.



Hình 2.5: Tổ chức các luồng lên và luồng xuống trong một stream

Tại cùng một vị trí trong mỗi hàng đợi là địa chỉ của đầu vào (entry point), một thủ tục để xử lý thông báo bất kỳ được nhận bởi hàng đợi. Thủ tục cho các hàng đợi H1 và H2 xử lý các thông báo gửi tới đầu luồng. Thủ tục cho các hàng đợi E1 và E2 xử lý các thông báo gửi tới phần cuối luồng.

Hình 2.6 chỉ ra các cấu trúc dữ liệu cho hàng đợi: queue, qinit, qband, module_info,module_stat.



Hình 2.6: Các cấu trúc dữ liệu cho mỗi hàng đợi

Các cấu trúc qband có các thông tin về các băng ưu tiên (priority band) trong hàng đợi. Cấu trúc dữ liệu hàng đợi chứa một vài giá trị có thể thay đổi cho hàng đợi . Cấu trúc qinit chứa một con trỏ tới các thủ tục xử lý, cấu trúc module_info chứa các giá trị hạn chế khởi tạo và cấu trúc module_stat được dùng để thu được các thống kê. Mỗi hàng đợi trong cặp hàng đợi chứa các tập cấu trúc dữ liệu khác nhau. Trong một vài tình huống, một cặp hàng đợi có thể chia sẻ một vài hoặc tất cả các cấu trúc dữ liệu. Cho ví dụ, có thể có một cấu trúc qinit riêng rẽ cho mỗi hàng đợi trong cặp hàng đợi và một cấu trúc module_stat biểu diễn cho cả hai hàng đợi. Hình 2.6 chỉ ra hai cặp hàng đợi liền kề với các liên kết trong cả hai chiều. Khi một mô đun được chèn vào luồng, STREAMS tạo một cặp hàng đợi và liên kết mỗi hàng đợi trong cặp hàng đợi tới hàng đợi liền kề theo hướng lên và hàng đợi liền kề theo hướng xuống. Sự kết hợp cho phép mỗi hàng đợi xác định hàng đợi liền kề tiếp theo của chúng. Quan hệ này được cài đặt giữa các hàng đợi liền kề bởi con trỏ q_next. Trong một cặp hàng đợi, mỗi hàng đợi xác định hàng đợi cùng cặp với nó bằng việc dùng các hàm STREAMS, do đó không có con trỏ nào giữa hai hàng đợi. Sự tồn tại của đầu luồng và cuối luồng giúp cho các thủ tục hàng đợi xác định được đích mà thông báo được gửi tới.

2.2.2.1 Mở một file thiết bị STREAMS

Một cách thức để xây dựng một luồng là mở một file đặc biệt của STREAMS. Tất cả các đầu vào trình điều khiển được định nghĩa bởi cấu trúc streamtab cho trình điều khiển này:

```
struct streamtab {
    struct  qinit       st_rdinit ;         /read queue */
    struct  qinit       st_wrinit ;         /write queue */
    struct  qinit       st_muxrinit ;  /lower read queue */
    struct  qinit       st_muxwinit ;/lower write queue */
```

Cấu trúc streamtab định nghĩa một mô đun hoặc một trình điều khiển.

st_rdinit trỏ tới các trúc qinit đọc cho trình điều khiển và st_wrinit trỏ tới cấu trúc qinit viết của trình điều khiển, st_muxrinit và st_muxwinit trỏ tới các cấu trúc qnit đọc và viết thấp hơn nếu driver là driver đa luồng (multiplexer driver). Nếu lời gọi open mở file khởi tạo thì một luồng được tạo.

Một đầu luồng được tạo từ một cấu trúc dữ liệu và một cặp cấu trúc hàng đợi. Nội dung của stdata và hàng đợi được khởi tạo với các giá trị định trước, bao gồm các thủ tục xử lý đầu luồng. Mỗi luồng có một đầu luồng. Đầu luồng được dùng bởi STREAMS trong khi tiến hành các thao tác trên luồng.

Một cặp cấu trúc hàng đợi được cấp phát cho mỗi đầu luồng. Các hạn chế về hàng đợi được khởi tạo với các giá trị được xác định trong cấu trúc module_info tương ứng. Các trình (rountine) xử lý hàng đợi được khởi tạo với các giá trị được xác định tại cấu trúc qinit. Tiếp đó, các giá trị q_next được xác lập sao cho hàng đợi viết của đầu luồng trỏ tới hàng đợi viết của trình điều khiển và hàng đợi đọc của trình điều khiển trỏ tới hàng đợi đọc của đầu luồng. Các giá trị q_next tại phần cuối của luồng được gán giá trị null. Cuối cùng thủ tục mở open (được cấp phát qua cấu trúc qinit đọc) của trình điều khiển được gọi . Nếu lời gợi open không mở file khởi tạo của luồng thì các thao tác được tiến hành là gọi một trình điều khiển và mở các thủ tục của tất cả các mô đun có thể chèn vào luồng. Khi một luồng đã được mở, việc thực hiện lời gọi open trên cùng một thiết bị sẽ mở các trình (rountine) của tất cả các mô đun và trình điều khiển trong luồng được gọi và theo thứ tự ngược lại với quá trình khởi tạo luồng. Đầu tiên trình điều khiển được mở và một mô đun được chèn vào luồng. Khi việc chèn xẩy ra một trình (rountine) của mô đun được gọi. Nếu lời gọi khác của cùng thiết bị được mở trình open của mô đun sẽ được mở sau trình (rountine) của trình điều khiển.

2.2.2.2 Thêm và huỷ các mô đun

Một mô đun có thể thêm với một lời gọi hệ thống ioctl I_PUSH. push chèn một mô đun ngay dưới đầu luồng. Bởi vì các thành phần của luồng là tương tự nhau nên thao tác push tương tự như thao tác open đối với trình điều khiển. Đầu tiên địa chỉ của cấu trúc qinit của mô đun được sử dụng. Tiếp đến, STREAMS cấp phát một cặp cấu trúc hàng đợi và khởi tạo nội dung của nó như trong thao tác mở tình điều khiển. Sau đó giá trị q_next được xác lập và một mô đun được chèn giữa đầu luồng và mô đun liền kề phía dưới nó. Cuối cùng thủ tục open (được cấp phát qua qinit) được gọi. Mỗi thao tác push của mô đun là độc lập trong cùng một luồng. Nếu cùng một mô đun được chèn nhiều lần vào luồng sẽ có nhiều sự cố đối với mô đun trong luồng. Tổng sô mô đun có thể chèn vào luồng được hạn chế bởi tham số nstrpush của nhân.

Một lời gọi hệ thống ioctl I_POP xoá một mô đun ngay dưới đầu luồng. Thao tác pop gọi thủ tục close của mô đun. Khi mô đun đóng, tất cả các thông báo còn trên mô đun được giải phóng. Sau đó STREAMS kết nối đầu luồng tới thành phần ngay dưới mô đun

bị xoá và bỏ cặp hàng đợi của mô đun. I_PUSH và I_POP cho phép một tiến trình người dùng thay đổi động cấu hình của một luồng bằng việc thêm (push) hoặc xoá (pop) các mô đun như yêu cầu. Ví dụ một mô đun có thể bị xoá và một mô đun mới được chèn vào dưới đầu luồng. Sau đó mô đun ban đầu có thể được thêm vào sau mô đun mới đã được chèn vào.

2.2.2.3 Đóng một luồng

Lời gọi close đối với file STREAMS sẽ tháo dỡ (dismantling) luồng. Việc tháo dỡ bao gồm việc đẩy các mô đun ra khỏi luồng và đóng trình điều khiển. Trước khi một mô đun bị đẩy ra , thao tác close có thể trễ để cho phép các thông báo trên hàng đợi viết của trình điều khiển được xử lý xong.

Chú ý: STREAMS chỉ giải phóng các thông báo chứa trên một hàng đợi thông báo. Các cấu trúc dữ liệu và thông báo bất kỳ được dùng bởi một trình điều khiển hoặc mô đun phải được giải phóng bởi thủ tục close của trình điều khiển hoặc mô đun.

2.3 CÁC TRÌNH XỬ LÝ LUỒNG

2.3.1 Các thủ tục put và service (srv)

Các thủ tục put và srv trong một hàng đợi là các trình (routine) xử lý các thông báo khi chúng chuyển qua hàng đợi. Việc xử lý được tiến hành dựa vào kiểu thông báo và làm thay đổi một thông báo, tạo một thông báo mới hoặc không tạo ra thông báo nào. Nói chung một thông báo được sửa hoặc tạo mới sẽ được gửi theo cùng hướng mà nó được nhận bởi hàng đợi nhưng cũng có thể được gửi theo hướng ngược lại. Mỗi thủ tục put đặt các thông báo lên hàng đợi của nó khi chúng đến, để được xử lý sau đó bởi thủ tục service.

Một queue luôn luôn chứa một thủ tục put và một thủ tục service đi cùng. Các thủ tục put và service được trỏ bởi queue.

2.3.1.1 Thủ tục put

Một thủ tục put là một trình (routine) của hàng đợi nhận thông báo từ hàng đợi trước trước nó trong luồng. Các thông báo được chuyển giữa các hàng đợi bởi thủ tục put. Một lời gọi tới thủ tục put trong hướng phù hợp là một cách thức để chuyển các thông báo giữa các thành phần STREAMS. Có một thủ tục put

riêng rẽ cho mỗi hàng đợi đọc và hàng đợi viết do các thao tác hai chiều (full-duplex) của hầu hết các luồng. Tuy nhiên có thể có một thủ tục put đơn được chia sẻ giữa hàng đợi đọc và hàng đợi viết. Cú pháp cho thủ tục put được chỉ ra như sau:

```
int prefixrput (queue_t *q, mblk_t *mp);
int prefixwput (queue_t *q, mblk_t *mp);
```

với q là con trỏ trỏ tới cấu trúc hàng đợi và mp là con trỏ trỏ tới khối thông báo. Thủ tục put cho phép đáp ứng nhanh những dữ liệu và sự kiện nhất định, nó có quyền ưu tiên cao hơn thủ tục service và gắn với việc xử lý ngay lập tức các thông báo.

Thủ tục put luôn thực hiện trước trình (routine) service đối với mỗi thông báo. Mỗi thành phần STREAMS truy nhập tới thủ tục put liền kề bằng cách dùng gián tiếp putnext.

Chú ý : một mô đun không bao giờ gọi trực tiếp các trình (routine) của các mô đun khác bao gồm các thủ tục put và service.

Ví dụ: giả sử modA,modB,modC là ba thành phần liên tiếp trong một luồng, với modC kết nối với đầu luồng. Nếu modA nhận một thông báo được gửi từ dưới lên, modA xử lý thông báo này và gọi thủ tục put đọc của modB, modB xử lý thông báo và gọi thủ tục put đọc của modC, modC xử lý thông báo và gọi thủ tục put đọc của đầu luồng. Như vậy thông báo được chuyển dọc theo luồng trong một dãy xử lý liên tục. Dãy xử lý này hoàn thành toàn bộ việc xử lý trong một thời gian ngắn

2.3.1.2 Thủ tục service

Thủ tục service được chứa trong mỗi hàng đợi để cho phép việc xử lý sau đó. Nếu một hàng đợi có cả thủ tục put và thủ tục service, việc xử lý thông báo sẽ được phân chia giữa hai thủ tục. Thủ tục put luôn được gọi đầu tiên từ hàng đợi trước đó. Sau khi hoàn thành phần xử lý thông báo, nó chuẩn bị cho thủ tục service được gọi bằng việc chuyển thông báo tới thủ tục putq. Thủ tục putq tiến hành hai việc: nó đặt thông báo lên hàng đợi thông báo của hàng đợi và liên kết hàng đợi tới phía cuối của hàng đợi xếp lịch của STREAMS. Sau khi putq thao tác xong, thủ tục put có thể kết thúc hoặc tiếp tục xử lý thông báo. Một khoảng thời gian sau đó, thủ tục service được gọi tự động bởi trình xếp lịch (scheduler) của streams.

Cú pháp cho thủ tục service như sau:

```
int  prefixrsrv(queue_t   *q);
int  prefixwsrv(queue_t   *q);
```

Trình xếp lịch STREAMS là riêng rẽ và phân biệt với trình xếp lịch tiến trình hệ thống. Nó chỉ quan tâm đến các hàng đợi được liên kết tới hàng đợi xếp lịch STREAMS. Trình xếp lịch gọi mỗi thủ tục service của hàng đợi xếp lịch tại một thời điểm theo cách thức FIFO (Vào trước ra trước). Các tiện ích STREAMS giao các thông báo tới thủ tục service trong cách thức FIFO trong mỗi lớp quyền ưu tiên (ưu tiên cao, băng ưu tiên, bình thường), bởi vì thủ tục service không biết các quyền ưu tiên thông báo và chi đơn giản là nhận thông báo tiếp theo. Thủ tục service nhận điều khiển theo thứ tự nó được xếp lịch. Khi một thủ tục service nhận điều khiển nó có thể gặp nhiều thông báo trên queue thông báo của nó. Số lượng thông báo này sẽ nhiều lên nếu có một khoảng thời gian dài giữa thời điểm một thôngbáo được được xếp hàng bởi thủ tục put và thời điểm trình xếp lịch STREAMS gọi thủ tục service kết hợp với nó. Trong khoảng thời gian này có thể có nhiều lời gọi tới thủ tục put và tạo ra nhiều thông báo. Thủ tục service phải luôn luôn xử lý tất cả các thông báo trên queue thông báo của nó trừ khi bị chặn bởi điều khiển luồng.

Một thủ tục service có thể dùng một putbq để đặt thông báo trở lại queue do điều khiển luồng hoặc các lý do khác. Một thông báo đặc quyền cao không bao giờ được đặt trở lại queue.

2.4 CÁC THÔNG BÁO

2.4.1 Giới thiệu về thông báo

Các thông báo là phương tiện (means) truyền thông trong luồng. Tất cả các đầu vào và đầu ra của của STREAMS được dựa trên thông báo. Các đối tượng (objects) chuyển giữa các thành phần của luồng đều trỏ tới các thông báo. Tất cả các thông báo trong STREAMS dùng hai cấu trúc dữ liệu để tra cứu đến dữ liệu trong thông báo. Các cấu trúc dữ liệu này mô tả kiểu của thông báo và chứa các con trỏ chỉ tới dữ liệu của thông báo cũng như các thông tin khác. Các thông báo được gửi qua luồng bằng cách gọi thành công trình (routine) put của mỗi hàng đợi trong luồng. Các thông báo có thể được sinh ra bởi trình điều khiển, mô đun hoặc đầu luồng.

Có một vài kiểu thông báo STREAMS khác nhau. Các thông báo khác nhau ở mục đích của nó và quyền ưu tiên xếTTp hàng. Nội dung của các kiểu thông báo nhất định có thể chuyển giao giữa một tiến trình và một luồng bằng việc dùng các lời gọi hệ thống. Các kiểu thông báo được mô tả ngắn gọn và phân lớp dựa theo quyền ưu tiên xếp hàng của chúng. Các thông báo thông thường:

M_BREAK :đòi hỏi đầu luồng gửi một "ngắt" (break)

M_CTL:Các đòi hỏi điều khiển/trạng thái được dùng cho truyền thông giữa các mô đun

M_DATA:thông báo dữ liệu người dùng cho các lời gọi hệ thống vào/ra

M_DELAY:đòi hỏi một trễ thời gian thực tại đầu ra.

M_IOCTL:Các đòi hỏi điều khiển/trạng thái được sinh bởi đầu luồng

M_PASFP: thông báo chuyển con trỏ file

M_PROTO:thông tin điều khiển hệ thống

M_SETOPTS: xác lập các lựa chọn tại đầu luồng, gửi lên

M_SIG :tín hiệu được gửi từ mô đun/trình điều khiển

Các thông báo ưu tiên cao:

M_COPYIN:sao chép dữ liệu vào cho ioctl, gửi xuống

M_COPYOUT: sao chép dữ liệu ra cho ioctl, gửi lên

M_ERROR: Báo cáo tình trạng lỗi của luồng xuống, gửi lên trên

M_FLUSH: flush hàng đợi của mô đun

M_HANGUP: xác lập một hangup của đầu luồng, gửi lên trên

M_IOCACK:thông báo "báo nhận " (acknowledgment) xác thực của ioctl

M_IOCDATA:Dữ liệu cho ioctl, gửi xuống

M_IOCNAK:thông báo 'báo nhận" (acknowledgment) từ chối của ioctl

M_PCPROTO:thông tin điều khiển giao thức

M_PCSIG:tín hiệu gửi từ môđun/trình điều khiển

M_READ: đọc khai báo, gửi xuống

M_START: khởi động lại đầu ra thiết bị đã dừng

M_STARTI: khởi động lại đầu vào thiết bị đã dừng

M_STOP:đình chi ra

M_STOPI:đình chỉ vào

2.4.2 Cấu trúc thông báo

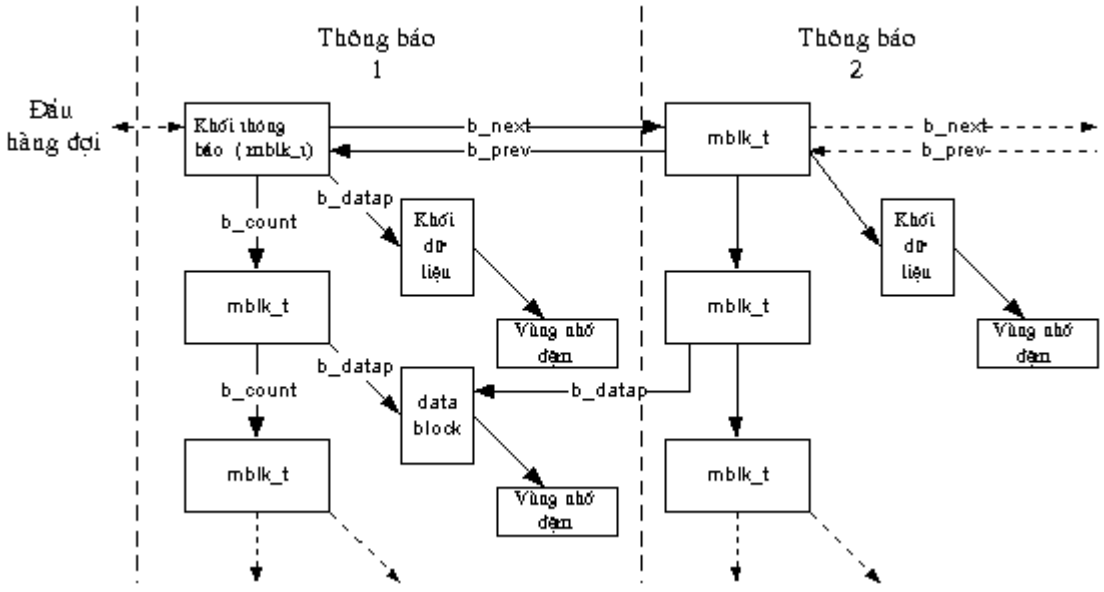
Tất cả các thông báo được tạo bởi một hoặc nhiều khối thông báo. Một khối thông báo là một danh sách liên kết của các bộ ba (triples), mỗi bộ bao gồm hai cấu trúc và một vùng nhớ đệm dữ liệu (buffer). Các cấu trúc là một khối thông báo (msgb) và một khối dữ liệu (datab). Vùng đệm dữ liệu dữ liệu được cấp phát trong bộ nhớ, ở đó dữ liệu của thông báo được chứa.

STREAMS dùng các b_next và b_prev để liên kết các thông báo trong hàng đợi. Các trình điều khiển và các mô đun có thể đọc nhưng không thể thay đổi trực tiếp các trường này. Các con trỏ b_rprt và b_wptr là các con trỏ đọc và viết ở trong vùng nhớ đệm dữ liệu được trở bởi con trỏ b_datap. Các con trỏ b_rptp và b_wptr được quản lý bởi các trình điều khiển và các mô đun.

Trường b_band xác định nơi thông báo được đặt khi nó được xếp hàng bằng cách dùng các trình (routine) tiện ích STREAMS. Khi một thông báo được cấp phát qua allocb(), trường b_band được gán giá trị 0, các mô đun và trình điều khiển có thể xác lập trường này với các giá trị từ 0 đến 255 dựa vào số băng (band) đặc quyền cần có. Các số thấp hơn có quyền ưu tiên thấp hơn.

Cấu trúc datab xác định các hạn chế cố định của vùng đệm dữ liệu (db_base và db_lim), một trường đếm (db_ref) và trường thông báo (db_type).

Mỗi thông báo bao gồm một hoặc nhiều khối thông báo được liên kết.



Hình 2.7: Cấu trúc và các liên kết thông báo

Một thông báo có thể là đơn (singly) như khi nó được xử lý bởi thủ tục put hoặc nó có thể được liên kết trên hàng đợi thông báo của hàng đợi và đợi để được xử lý bởi thủ tục service. Thông báo 2 trong hình 2.7 liên kết tới thông báo 1. STREAMS cho phép các khối thông báo của các thông báo khác nhau cùng chia sẻ một khối dữ liệu (data block) để giảm việc lưu trữ và tránh việc tràn sao chép (copying overhead) và được gọi là các khối thông báo kép (Multiple message block). Trong hình trên một khối dữ liệu trong thông báo 1 được chia sẻ giữa thông báo 1 và thông báo khác. Ví dụ, cùng một khối dữ liệu với vùng nhớ đệm kết hợp với nó, có thể được tham chiếu trong hai thông báo, từ các mô đun riêng rẽ cài đặt các mức giao thức riêng rẽ.

Chú ý: các mô đun và trình điều khiển không thể thay đổi b_next và b_prev. Các trường này được thay đổi bằng việc dùng các trình (routine) như putq, getq.

2.4.3 Gửi và nhận các thông báo

Hầu hết các kiểu thông báo có thể được sinh bởi các mô đun và trình điều khiển. Một vài thông báo dành cho đầu luồng. Các thông báo hay được dùng nhất là M_DATA, M_PROTO và M_PCPROTO. Các thông báo này có thể được chuyển giữa một tiến trình và một mô đun trên cùng (topmost) trong một luồng , với cùng một biên thông báo (mesage boundary) được quản lý tại cả hai phía của nhận. Các thông báo M_PROTO và M_PCPROTO có chức năng chuyển các thông tin giao diện dịch vụ giữa các mô đun, trình điều khiển và các tiến trình người dùng. Một vài kiểu thông báo chỉ có thể được dùngtrong một luồng và không thể gửi hoặc nhận từ mức người dùng. Các mô đun và trình điều khiển không giao tiếp trực tiếp với các lời gọi hệ thống trừ open và close. Đầu luồng (stream head) điều khiển tất cả các thông báo chuyển giữa các tiến trình người dùng và các thành phần streams . Các thông báo truyền giữa các tiến trình người dùng và đầu luồng (stream head) với nhiều dạng khác nhau. Ví dụ các thông báo M_DATA và M_PROTO có thể truyền trong dạng trực tiếp của chúng bởi các lời gọi hệ thống getmsg và putmsg. Lời gọi write làm cho một hoặc nhiều thông báo M_DATA được tạo trong vùng nhớ đệm (buffer) dữ liệu được chi ra trong lời gọi . Các thông báo M_DATA nhận tại đầu luồng (stream head) sẽ được dùng bởi lời gọi read và được sao chép vào vùng nhớ đệm (buffer) người dùng. Một ví dụ khác, thông báo M_SIG bắt buộc đầu luồng (stream head) phải gửi một tín hiệu tới một tiến trình.

Các mô dun và trình điều khiển (driver) có thể gửi thông báo bất kỳ theo các hướng trong luồng . Tuy nhiên dựa vào việc sử dụng các thông báo trong streams

và cách xử lý chúng bởi đầu luồng (stream head), các thông báo nhất định có thể được phân loại như luồng lên (upstream), luồng xuống (downstream), hai chiều.

Các thông báo M_DATA, M_PROTO hoặc M_PCPROTO có thể được gửi theo cả hai chiều. STREAMS cho phép các mô đun tạo các thông báo và chuyển chúng tới các mô đun liền kề. Tuy nhiên các lời gọi hệ thống read và write không cho phép một tiến trình người dùng tạo và nhận tất cả các thông báo. read và write chỉ có một vùng nhớ đệm (buffer) dùng cho việc truyền và nhận thông báo. Nếu thông tin điều khiển và dữ liệu được đặt trong một vùng đệm (buffer) đơn, người dùng sẽ phải phân tích nội dung của vùng đệm (buffer)để tách dữ liệu từ thông tin điều khiển.

2.4.4 Cấu trúc hàng đợi (queue)

Các thủ tục dịch vụ, các hàng đợi thông báo, quyền ưu tiên thông báo và các điều khiển luồng cơ sở gắn kết với nhau trong STREAMS. Một hàng đợi sẽ không dùng hàng đợi thông báo của nó nếu không có thủ tục dịch vụ trong hàng đợi. Chức năng của thủ tục dịch vụ là xử lý các thông báo trên hàng đợi của nó. Các quyền ưu tiên thông báo và điều khiển luồng được kết hợp với các hàng đợi thông báo. Thao tác của hàng đợi tập trung vào cấu trúc hàng đợi. Các hàng đợi luôn luôn được cấp phát từng cặp (đọc và viết), một cặp hàng đợi cho mỗi mô đun, một trình điều khiển và một đầu luồng. Một hàng đợi chứa một danh sách liên kết của các thông báo. Khi một cặp hàng đợi được cấp phát, các trường sau được khởi tạo bởi STREAMS:

q_info : từ streamtab

q_minp,q_map,q_hiwat,q_lowat : từ module_info

Sao chép các giá trị từ module_info cho phép chúng được thay đổi trong hàng đợi mà không cần thay đổi các giá trị trong streamtab và module_info

q_count được dùng trong việc tính toán điều khiển luồng và là số byte trong các thông báo trong queue.

2.4.5 Xử lý các thông báo

Các thủ tục put được đòi hỏi trong các mô đun có thể thêm vào luồng. Các thủ tục service là tuỳ chọn. Nếu một trình (routine) put xếp hàng các thông báo, sẽ phải tồn tại một trình (routine) service tương ứng điều khiển các thông báo

được xếp hàng. Nếu trình (routine) put không xếp hàng các thông báo thì trình (routine) service không cần tồn tại.

Việc xử lý nói chung khi có thủ tục service như sau:

Một thông báo được nhận bởi thủ tục put kết hợp với hàng đợi, trên đó có một vài thao tác xử lý có thể tiến hành trên thông báo.

Thủ tục put đặt các thông báo vào hàng đợi bằng việc dùng trình tiện ích putq cho thủ tục service để tiến hành việc xử lý tiếp theo.

putq() đặt các thông báo lên hàng đợi dựa trên quyền ưu tiên của nó.

putq làm cho hàng đợi sẵn sàng cho việc thực hiện của trình xếp lịch sau tất cả các hàng đợi khác đã được xếp lịch

Khi hệ thống chuyển từ chế độ nhân (kernel mode) sang chế độ người dùng (user mode), trình xếp lịch gọi thủ tục service

Thủ tục service lấy thông báo đầu tiên (q_first) từ hàng đợi thông báo bằng việc dùng tiện ích getq() .

Thủ tục service xử lý thông báo và đặt nó tới thủ tục put của hàng đợi tiếp theo nhờ putnext.

Thủ tục service lấy thông báo tiếp theo và xử lý nó. Việc xử lý này tiếp tục cho đến khi hàng đợi rỗng hoặc điều khiển luồng khoá việc xử lý.

Nếu không có đòi hỏi xử lý trong thủ tục put

2.4.6 Giao diện dịch vụ

STREAMS cung cấp một cách thức để cài đặt một giao diện dịch vụ (service interface) giữa hai thành phần trong một luồng, và giữa một tiến trình người dùng và mô đun phía trên cùng trong một luồng. Một giao diện dịch vụ được định nghĩa tại biên giữa một người dùng dịch vụ (service user) và một người cung cấp dịch vụ (service provider). Một giao diện dịch vụ là một tập các phần tử nguyên thuỷ (primitives) và các quy tắc định nghĩa một dịch vụ và trạng thái cho phép khi các phần tử nguyên thuỷ (primitive) được chuyển giữa người dùng dịch vụ và người cung cấp dịch vụ. Các quy tắc này được biểu diễn bởi máy trạng thái. Trong STREAMS, người dùng dịch vụ và người cung cấp dịch vụ được cài đặt trong một mô đun, trình điều khiển hoặc một tiến trình người dùng.Các phần tử nguyên thuỷ được chuyển theo hai chiều giữa một người dùng dịch vụ và người cung cấp dịch vụ trong các thông báo M_PROTO và M_PCPROTO. Các thông

báo PROTO có thể gồm nhiều khối. Khối đầu tiên trong một thông báo PROTO chứa phần điều khiển của một phần tử nguyên thuỷ trong dạng quy định giữa người dùng dịch vụ và người cung cấp dịch vụ.

Các khối M_DATA chứa các phần dữ liệu được kết nối với phần tử nguyên thuỷ. Phần dữ liệu có thể được xử lý trong một mô đun nhận nó hoặc có thể gửi tới thành phần luồng tiếp theo, cùng với dữ liệu được sinh ra bởi mô đun.

Các thông báo PROTO có thể gửi theo hai chiều trên luồng và giữa một luồng và một tiến trình người dùng.

Các lời gọi putmsg và getmsg tương tự với write và read ngoại trừ rằng chúng cho phép cả phần dữ liệu và điều khiển luồng được chuyển riêng rẽ và giữ biên thông báo ngang qua giao diện luồng và người dùng.

putmsg và getmsg sao chép riêng rẽ phần điều khiển (khối M_PROTO hoặc M_PCPROTO) và phần dữ liệu (khối M_DATA) giữa luồng và tiến trình người dùng.

2.4.7 Một số cấu trúc dữ liệu được dùng trong luồng

2.4.7.1 Cấu trúc Streamtab

Dùng để định nghĩa một mô đun hoặc trình điều khiển

```
streamtab     {
        struct  qinit   *st_rdinit;          /*định nghĩa hàng đợi đọc */
        struct  qinit   *st_wrinit;          /*định nghĩa hàng đợi viết*/
        struct  qinit   *st_muxrinit       /*hàng đợi đọc dùng cho đa luồng */
        struct  qinit   *st_muxwinit     /*hàng đợi đọc dùng cho driver của stream đa luồng*/
};
```

2.4.7.2 Các cấu trúc hàng đợi queue

```
Struct queue {
        struct  qinit   *q_qinfo;       /*Khởi tạo queue */
        struct  msgb    *q_first;            /* khối thông báo đầu tiên*/
        struct  msgb    *q_last;             /* khối thông báo cuối cùng*/
```

```
        struct  queue *q_next;              /* Queue tiếp theo của stream */
        struct  queue *q_link;              /* Queue tiếp theo để xếp lịch*/
        caddr_t         q_ptr;              /*Dữ liệu riêng */
        short   q_count;                    /* Số byte dữ liệu trong queue */
        unsigned  short    q_flag;    /* Trạng thái queue */
        short   q_minpsz;                /* Cỡ gói tối thiểu*/
        short   q_maxpsz;                /* Cỡ gói tối đa*/
        short   q_hiwat;                 /* Dấu nước cao*/
        short   q_lowat;                 /* Dấu nước thấp*/
        char    *q_pad;                  /*pad field*/
};
typedef struct queue queue_t;
```

2.4.7.3 Cấu trúc qinit

```
struct qinit  {
        int     (*qi_putp) ( );         /* thủ tục put */
        int     (*qi_srvp) ( );         /*thủ tục service */
        int     (*qi_qopen) ( );        /*được gọi để mở driver/chèn mô đun */
        int     (*qi_qclose) ( );       /*được gọi để đóng driver/đẩy mô đun lần cuối */
        int     (*qi_qadmin) ( );               /*dự trữ cho tương lai*/
        struct module_info *qi_minfo;   /*cấu trúc thông tin mô đun */
        struct module_stat   *qi_mstat;     /*Trường lựa chọn */
};
```

2.4.7.4 Cấu trúc module_info

```
struct module_info   {
        short           mi_idnum;           /*số định danh của mô đun */
        char            *mi_idname; /*tên mô đun */
        short           mi_minpsz;          /*cỡ gói tối thiểu  */
        short           mi_maxpsz;          /*cỡ gói tối đa */
        short           mi_hiwat;           /*mức nước cao */
```

```
        short           mi_lowat;           /*mức nước thấp */
};
```

2.4.7.5 Các cấu trúc thông báo msgb

```
struct msgb{
Struct  msgb        *b_next      /*thông báo tiếp theo trên queue */
Struct  msgb        *b_prev      /*thông báo trước trên queue*/
Struct  msgb        *b_cont      /*khối thông báo tiếp theo*/
Unsighed   char     * b_rptr     /*byte chưa đọc đầu tiên trong buffer*/
Unsigned char       b_wptr       /*byte chưa viết đầu tiên trong buffer*/
Struct datab        *b_datap     /*chỉ tới khối dữ liệu*/
Unsighed   char      b_band      /*quyền ưu tiên thông báo*/
                    Unsighed    short     b_flag        /*cờ thông báo*/
}
```

Giá trị cờ:

MSGMARK byte cuối cùng của thông báo được đánh dấu

MSGDELIM thông báo không bị giới hạn

2.4.7.6 Cấu trúc datab

```
struct datab {
    unsigned char  *db_base;  /* byte đầu tiên của buffer */
    unsigned char  *db_lim;   /* byte cuối cùng của buffer +1 */
    unsigned char  db_ref;    /*  số thông báo dùng chung khối dữ liệu này */
    unsigned char  db_type;   /* kiểu thông báo */
```

2.5 CÁC TRÌNH ĐIỀU KHIỂN (DRIVERS)

2.5.1 Tổng quan về trình điều khiển

Một trình điều khiển thiết bị (device driver) là một phần mềm cung cấp một giao diện giữa hệ điều hành và một thiết bị. Trình điều khiển điều khiển thiết bị đáp ứng các đòi hỏi của nhân (kernel). Các đòi hỏi được đưa ra thông qua các điểm vào (entry points). Trình điều khiển cung cấp và quản lý một đường dẫn cho

dữ liệu đi và đến từ thiết bị phần cứng, và các ngắt dịch vụ được dùng bởi trình điều khiển thiết bị. Trong STREAMS các trình điều khiển được mở (opened) và các mô đun được chèn vào (pushed). Trong SunOS5 có ba kiểu của trình điều khiển thiết bị:

Trình điều khiển phần cứng (Hardware Driver)

Kiểu này của trình điều khiển chỉ truyền thông với một thành phần đặc biệt của phần cứng. Nhận một số thiết bị ngoại vi, các trình điều khiển này có nhiều chức năng.

Trình điều khiển ảo (Pseudo Driver):

Trình điều khiển này được cấu hình và cài đặt giống như trình điều khiển phần cứng, nhưng không giao tiếp với phần cứng.

Trình điều khiển đa luồng (Multiplexer Driver):

Đây là một trình điều khiển STREAMS thông thường nhưng có nhiều luồng kết nối với nó. Không giống như mô đun, một trình điều khiển thiết bị có một trình (routine) ngắt có thể truy nhập từ một ngắt phần cứng cũng như từ một luồng, trừ khi nó là một trình điều khiển ảo hoặc một trình điều khiển đa luồng. Tuy nhiên những sự khác nhau này không được nhận biết bởi cơ chế STREAMS. Chúng được điều khiển bởi mã được cung cấp bởi nhà phát triển bao gồm trong các thủ tục trình điều khiển.

2.5.2 Đa luồng (Multiplexing)

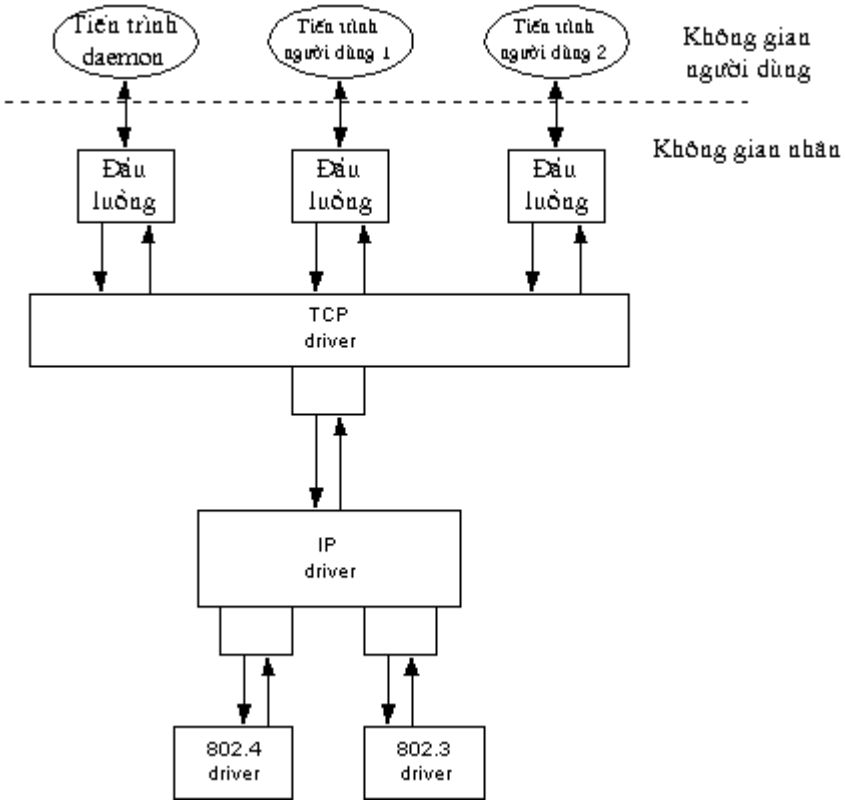
2.5.2.1 Giới thiệu về đa luồng:

Một đa luồng STREAMS (STREAMS Multiplexer) là một trình điều khiển với nhiều luồng kết nối tới nó. Chức năng chính của trình điều khiển đa luồng là chuuyển các thông báo trong các luồng được kết nối với nhau.

Các cấu hình đa luồng được tạo từ mức người dùng bởi các lời gọi hệ thống.

2.5.2.2. Xây dựng đa luồng STREAMS TCP/IP

Hình 2.8 chỉ ra một đa luồng với giao thức TCP/IP



Hình 2.8: Mô hình STREAMS TCP/IP đa luồng

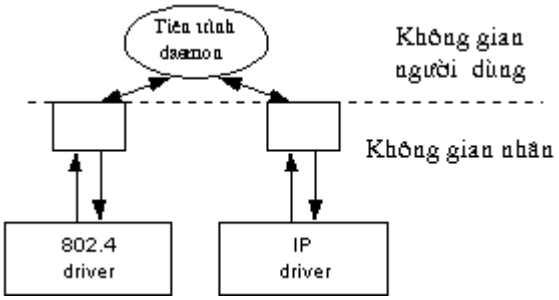
Để người dùng không cần có những hiểu biết về cấu trúc giao thức nền, một tiến trình ngầm (daemon) mức người dùng sẽ được xây dựng để quản lý việc cấu hình đa luồng. Người dùng sau đó có thể truy nhập giao thức vận tải một cách trực tiếp bằng việc mở một đầu mối (node) thiết bị điều khiển giao thức TCP. Một trình điều khiển giao thức liên mạng IP chuyển dữ liệu từ một luồng đơn phía trên tới một trong hai luồng thấp hơn.

Trình điều khiển này cung cấp hai thành phần luồng bên dưới nó.Hai luồng này kết nối đến hai mạng phân biệt. Một mạng dùng chuẩn IEEE 802.3 thông qua trình điều khiển 802.3 và mạng khác dùng chuẩn IEEE 802.4 thông qua trình điều khiển 802.4.

Trình điều khiển TCP đa kết nối (multiplexes) các luồng phía trên qua một luồng đơn tới trình điều khiển IP. Đa luồng STREAMS nhiều mức này được xây dựng từ dưới lên. Bắt đầu từ việc xây dựng đa luồng IP (IP Multiplexer). Trình

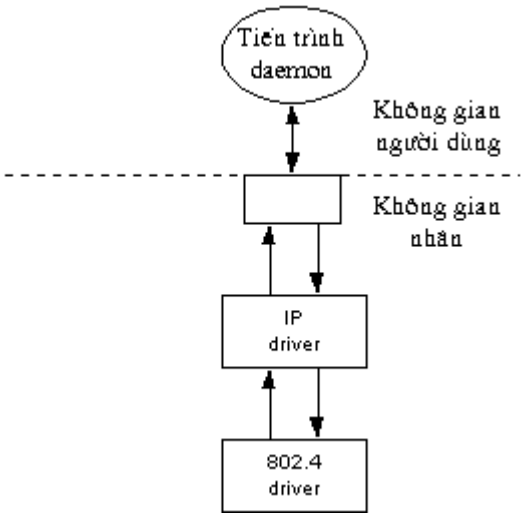
điều khiển đa luồng IP cũng giống như các trình điều khiển phần mềm khác. Nó có một node trong hệ thống file của Solaris và được mở giống như các trình điều khiển thiết bị khác.

Bước đầu tiên là mở một trình điều khiển đa luồng và trình điều khiển 802.4 để tạo ra hai luồng riêng rẽ trên mỗi trình điều khiển như chỉ ra trong hình 2.9



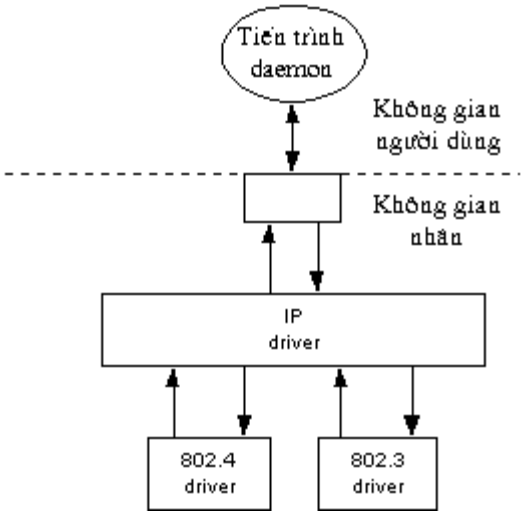
Hình 2.9: Quá trình tạo STREAMS IP đa luồng

Bây giờ luồng gắn với 802.4 có thể được kết nối với phía dưới trình điều khiển IP để được một STREAMS IP như sau:



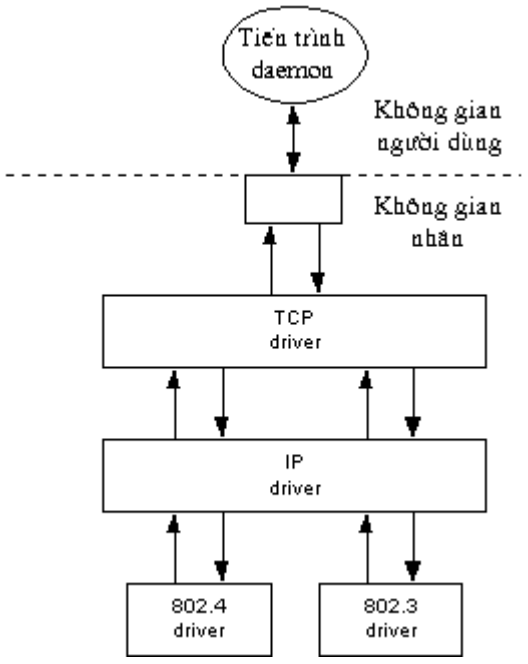
Hình 2.10: Mô hình STREAMS IP

Tiếp tục kết nối với trình điều khiển 802.3 để được một STREAMS IP đa luồng như sau:



Hình 2.11: Mô hình STREAMS IP đa luồng

Cuối cùng trình điều khiển IP sẽ liên kết với phía dưới của trình điều khiển đa luồng TCP để được một đa luồng STREAMS TCP/IP như sau:



Hình 2.12: Xây dựng một STREAMS TCP/IP đa luồng

CHƯƠNG III
GIẢI PHÁP BẢO VỆ DỮ LIỆU TRONG NHÂN HỆ ĐIỀU HÀNH SOLARIS

3.1 GIẢI PHÁP BẢO VỆ GÓI IP TRÊN SOLARIS BẰNG KỸ THUẬT MẬT MÃ

3.1.1 Đặt vấn đề

Trong giao thức TCP/IP, dữ liệu của các ứng dụng khi truyền trên mạng được chứa trong các gói IP. Mỗi gói IP bao gồm dữ liệu ứng dụng và các thông tin điều khiển (header của tầng Internet và tầng vận tải). Trong quá trình truyền từ máy gửi tới máy nhận, gói IP phải đi qua các đường truyền công khai, các thiết bị kết nối mạng và các máy tính trung gian. Tại bất cứ đâu, các gói IP đều có thể bị tấn công từ những người không uỷ quyền mà ta gọi chung là hacker:

- Đọc nội dung gói : hacker tìm cách đọc được các thông tin trên gói như dữ liệu ứng dụng, các thông tin điều khiển trong IP header, TCP (UDP) header. Không dừng lại ở việc đọc nội dung của một gói riêng lẻ, có những chương trình phần mềm hacker được cài đặt ở các máy tính trung gian trên đường truyền cho phép thu được toàn bộ nội dung của các file văn bản, thư tín điện tử, trang WEB,...từ những gói IP thu được.
- Sửa đổi nội dung gói IP: hacker có thể thu và sửa một số thông tin trong gói IP như dữ liệu ứng dụng, các thông tin trong các đầu (header) để nhằm các mục đích như :

+ Làm sai lệch thông tin được truyền bằng cách sửa nội dung của dữ liệu ứng dụng trong gói IP.

+ Thay đổi đích đến của gói IP, bằng cách sửa địa chỉ đích của gói IP .

+ Phá một liên kết TCP bằng cách sửa các giá trị cờ trong TCP header ,....

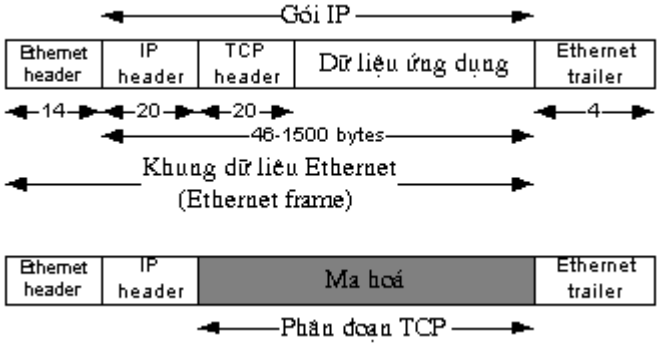
- Giả địa chỉ nguồn : Một máy nào đó của hacker giả địa chỉ IP của một máy được uỷ quyền để trao đổi dữ liệu với người nhận hợp pháp
- Chèn các gói IP giả : khi một liên kết đã được xác lập giữa hai người dùng hợp pháp, một người thứ ba chèn các gói IP giả vào.

Mục đích của việc bảo vệ gói IP trên đường truyền là:

- Bí mật nội dung của dữ liệu được truyền: bằng cách mã hoá dữ liệu ứng dụng trong gói IP. Khi dữ liệu ứng dụng của các gói IP được mã hoá, hacker không thể thu được nội dung của dữ liệu truyền trên mạng như các file, thư tín, trang WEB,...
- Xác thực gói IP: phát hiện và huỷ những gói IP bị sửa đổi trái phép bởi hacker hoặc những gói IP giả bằng cách mã hoá TCP (UDP) header. Trong TCP (UDP) header có trường tổng kiểm tra có tác dụng kiểm tra nhằm phát hiện ra những gói IP bị sửa đổi trái phép. Do TCP (UDP) header được mã hoá , nên trường tổng kiểm tra được che dấu, chính vì vậy nếu một hacker đã sửa được nội dung gói IP thì họ không thể tính lại được tổng kiểm tra, chính vì vậy tại máy nhận TCP sẽ phát hiện ra việc gói IP bị sửa bất hợp pháp bằng cách dựa vào tổng kiểm tra và sẽ huỷ gói.

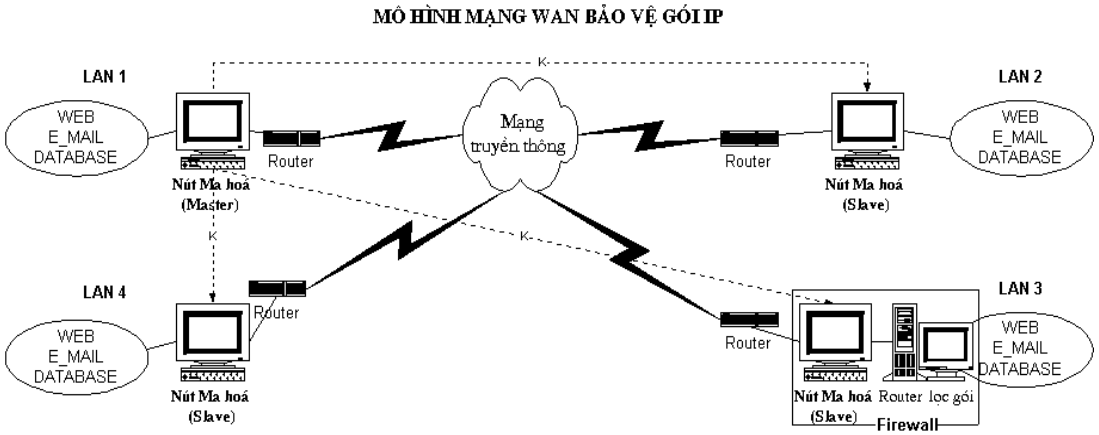
Hơn nữa việc mã TCP (UDP) header cũng hỗ trợ cho việc nâng cao tính bí mật của nội dung dữ liệu được truyền trên mạng. Khi TCP/UDP header được mã, một số trường mà hacker cần sử dụng để tấn công gói IP như số cổng đích chẳng hạn sẽ giúp cho hacker tách được các gói IP tương ứng với các ứng dụng khác nhau để thu được thông tin truyền từ máy nguồn đến máy đích như nội dung một file, một bức thư điện tử, một trang WEB,...Khi cổng đích được mã hoá phần mềm bắt gói của hacker không thể tách các gói dữ liệu của từng ứng dụng để tạo ra file ban đầu. Qua thử nghiệm khi gặp những gói IP có phân đoạn TCP được mã thì những chương trình bắt gói đó bị treo. Ngoài ra việc mã TCP (UDP) header sẽ rút ngắn thời gian xử lý gói. Khi mã hoá cả phân đoạn TCP , hệ thống chỉ cần xác định byte đầu tiên của phân đoạn TCP bằng cách tính độ dài IP header. Nếu chỉ mã dữ liệu ứng dụng, hệ thống phải tiến hành thêm bước xác định giao thức vận tải là TCP hay UDP, từ đó truy nhập đến byte đầu tiên của phần dữ liệu ứng dụng (TCP header có độ dài 20 bytes, UDP header có độ dài 8 bytes).

Hình 3.1 chỉ ra phần nội dung của gói IP được mã hoá.



Hình 3.1: Phân đoạn TCP của gói IP được mã hoá

3.1.2 Mô hình mạng WAN bảo vệ gói IP bằng kỹ thuật mật mã



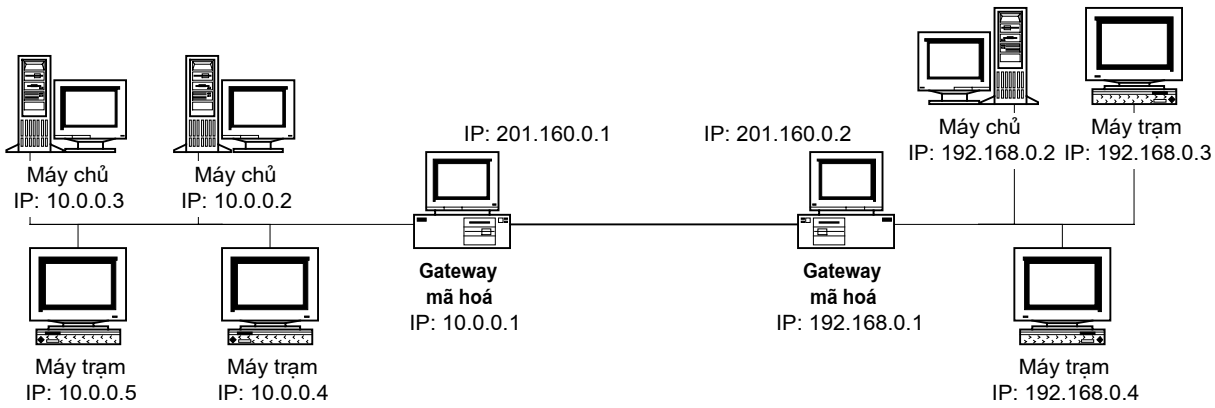
Hình 3.2: Mô hình mạng WAN bảo vệ gói IP dùng kỹ thuật mật mã

- Có n mạng Lan được kết nối với nhau để tạo thành một mạng WAN. Tại mỗi mạng Lan có một Solaris gateway có chức năng bảo vệ gói IP bằng kỹ thuật mật mã và gọi là nút mã hoá
- Các gói IP được sinh ra bởi các ứng dụng dùng giao thức TCP/IP trên các mạng Lan (Thư tín điện tử, Cơ sở dữ liệu, WEB,...) được bảo vệ tại các nút mã hoá. Thông tin của gói IP được bảo vệ là phân đoạn TCP (TCP segment), bao gồm TCP/UDP header và dữ liệu ứng dụng. Chúng ta quy ước gọi là "dữ liệu của gói IP"
- Kỹ thuật mật mã được sử dụng để mã hoá "dữ liệu của gói IP" là kỹ thuật mã khối
- Tại một thời điểm, các nút mã hoá sử dụng chung một khoá phiên K
- Quản lý và phân phối khoá:

+ Phân phối khoá: Khoá phiên được phân phối bằng hệ mật khoá công khai RSA thông qua một giao thức xác thực. Khoá công khai được phân phối đến từng nút

+ Thay đổi khoá phiên : Theo định kỳ, khoá phiên cũ được thay thế bởi khoá phiên mới.

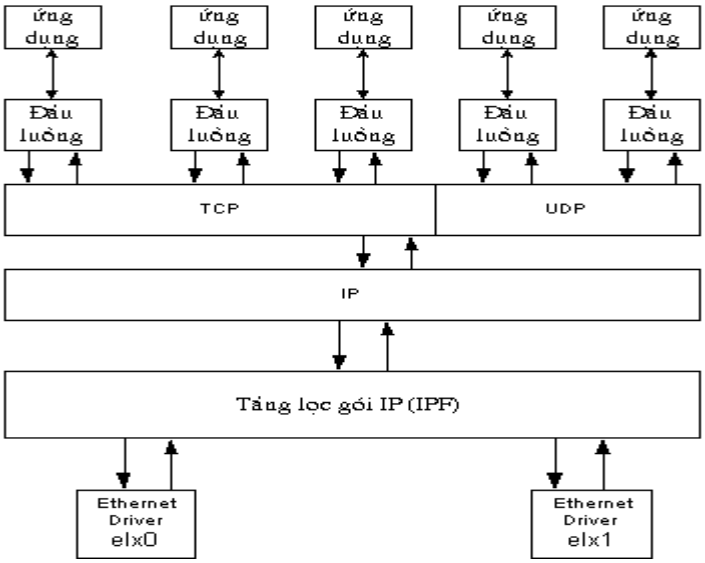
Trong hệ thống có một nút có chức năng quản lý khoá phiên cho toàn mạng bao gồm việc tạo, phân phối và thay đổi khoá. Nút này được gọi là Master. Các nút khác nhận khóa phiên từ nút Master và được gọi là Slave. Hình 3.3 là mô hình thử nghiệm của hệ thống bảo vệ gói IP trên mạng LAN:



Hình 3.3 Mô hình thử nghiệm của hệ thống bảo vệ gói IP

3.1.3 Giải pháp bắt gói IP để thực hiện việc mã hoá:

Để có thể bảo vệ gói IP, chúng ta cần phải bắt được các gói IP để mã hoá phân đoạn TCP sau đó lại trả lại gói IP để hệ thống xử lý tiếp. Thực chất của việc bắt gói IP chính là khả năng can thiệp được vào nội dung của gói IP. Cơ chế STREAMS của UNIX cho phép chúng ta xây dựng một tầng lọc gói IP trong mô hình STREAMS TCP/IP.



Hình 3.4: Mô hình STREAMS TCP/IP có chèn thêm tầng lọc gói IP

Theo cơ chế STREAMS chúng ta có thể chèn thêm một môđun có tên gọi là "Tầng lọc gói IP", gọi tắt là IPF vào dưới tầng IP (hình 3.4). Mỗi máy sẽ có hai cạc mạng tương ứng với hai giao diện elx0 và elx1. Tầng IPF cho phép chúng ta chặn bắt và mã hoá phân đoạn TCP của các gói IP. Dữ liệu của gói IP tại tầng IPF sẽ được quản lý bởi cấu trúc hàng đợi trong STREAMS TCP/IP. Chúng ta tìm hiểu về các thành phần của STREAMS TCP/IP trước và sau khi chèn tầng IPF.

STREAMS TCP/IP trước khi chèn tầng IPF:

Khi hệ thống khởi động, các mô đun được tải (load) tự động vào hệ thống là: IP, TCP, UDP, ELX. Dùng lệnh modinfo, chúng ta có thể xem thông tin về các môđun trên:

Ví dụ: dòng dưới đây là thông tin về mô đun IP.

11 f56099e8 18b28 0 1 ip (ip streams module)

Ngoài ra một môđun có thể được tải vào hệ thống bằng lệnh modload, với tham số là tên của file tương ứng với môđun trong thư mục dev và dừng tải bằng lệnh modunload. Các mô đun trên được kết nối tự động để tạo thành một đa luồng TCP/IP. Người dùng không thể dùng lệnh modunload để dừng việc tải một môđun nào đó.

Streams TCP/IP sau khi có module ipf

Module IPF được tải và dừng tải vào nhân nhờ lệnh modload và modunload với đường dẫn file là /usr/kernel/drv/ipf. Để kiểm tra xem mô đun IPF đã được tải vào nhân chưa ta dùng lệnh modinfo | grep ipf . Sau khi được tải vào nhân, mô đun IPF được chèn vào giữa môđun IP và môđun elx .

3.1.4 Quản lý dữ liệu của gói IP tại tầng IPF

Theo cấu trúc STREAMS, mỗi luồng bao gồm một danh sách các cặp hàng đợi. Mỗi cặp bao gồm một hàng đợi đọc và một hàng đợi viết. Hàng đợi đọc dùng để đón dữ liệu từ dưới lên. Hàng đợi viết dùng để đón dữ liệu từ trên xuống. Trong STREAMS TCP/IP, mỗi luồng có tất cả 4 cặp hàng đợi tương ứng với đầu luồng, tầng vận tải, tầng Internet, tầng truy nhập mạng. Riêng tầng vận tải có hai giao thức TCP và UDP nên có hai cặp hàng đợi, một cặp cho giao thức TCP, một cặp cho giao thức UDP.

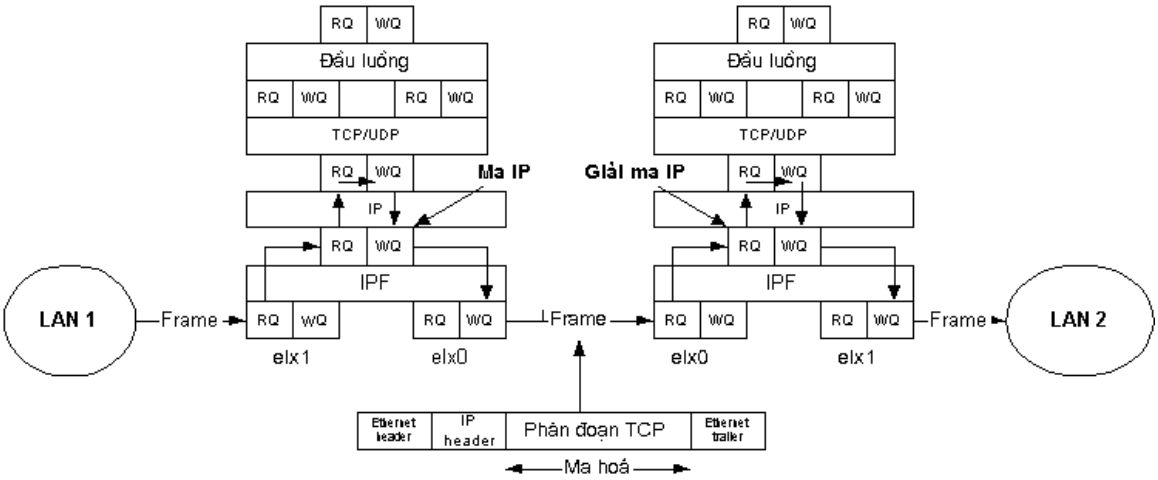
Chính vì điều này nên STREAMS TCP/IP còn được gọi là STREAMS đa luồng. Dữ liệu của các gói IP khi chuyển qua giao thức TCP/IP được quản lý tại

các hàng đợi và được chứa trong các vùng đệm của các khối thông báo. Mỗi khối thông báo bao gồm một khối dữ liệu, mỗi khối dữ liệu quản lý trực tiếp một vùng đệm bởi một loạt các con trỏ.

Khi một tầng IPF được chèn vào dưới tầng IP, một cặp hàng đợi đọc và viết của tầng IPF được tạo ra. Việc xử lý các gói IP được thực hiện tại các hàng đợi này. Sau đây chúng ta tìm hiểu về quá trình điều khiển và mã hoá dữ liệu của gói IP tại STREAMS TCP/IP có chèn thêm tầng IPF. *Hình 3.5* mô tả các cặp hàng đợi trong mô hình STREAMS TCP/IP có chèn thêm tầng IPF:

3.1.5 Cơ chế mã hoá dữ liệu gói IP của STREAMS TCP/IP

Gói IP được sinh ra bởi các ứng dụng trên mạng Lan được truyền theo cáp mạng đến giao diện elx1 của "nút mã hoá" của mạng LAN và được chứa trong hàng đợi đọc của giao diện elx1. Tiếp đó gói IP lần lượt được chuyển lên hàng đợi đọc của tầng IPF và hàng đợi đọc của tầng IP. Tại đây địa chỉ đích của gói IP được sử dụng để hệ thống quyết định đường đi tiếp theo nhờ vào các lệnh route.



Hình 3.5: Cấu trúc queue và quá trình mã/giải mã gói IP trong mô hình STREAMS TCP/IP với hai giao diện mạng

Gói IP được chuyển sang hàng đợi viết của tầng IP, sau đó được chuyển xuống hàng đợi viết của tầng IPF. Tại hàng đợi viết của tầng IPF, phân đoạn TCP (TCP segment) được mã hoá và được chuyển tiếp xuống hàng đợi viết của giao diện elx0 và được chuyển theo lên mạng để đi tiếp . Sau khi qua kênh truyền công khai, gói IP sẽ đến giao diện elx0 của "nút mã hoá" của mạng Lan đích và được chứa trong hàng đợi đọc của giao diện elx0. Tiếp đó gói IP được chuyển lên

hàng đợi đọc của tầng IPF và được giải mã. Sau khi được giải mã, gói IP lần lượt được chuyển lên hàng đợi đọc và hàng đợi viết của tầng IP, sau đó được chuyển xuống hàng đợi viết của tầng IPF và tiếp đến là hàng đợi viết của giao diện elx1 của nút mã hoá để đi vào mạng LAN đích.

3.1.6 Quản lý gói tại STREAMS TCP/IP

Mỗi gói IP tương ứng với một thông báo (message) của luồng TCP/IP. Mỗi hàng đợi quản lý nhiều gói. Mỗi gói được quản lý bởi nhiều khối thông báo (msgb), mỗi khối thông báo quản lý một một số bytes thông tin của gói và phần thông tin này được chứa trong một vùng đệm của khối thông báo. Vùng đệm này được trỏ bởi hai con trỏ b_rptr và b_wptr trong cấu trúc khối thông báo msgb, chúng lần lượt chỉ đến địa chỉ của byte đầu và byte cuối của dữ liệu IP trong vùng đệm. Việc chuyển gói IP giữa các thành phần của luồng thực chất là chuyển các con trỏ của hàng đợi.

3.1.7 Mã dữ liệu trong gói IP:

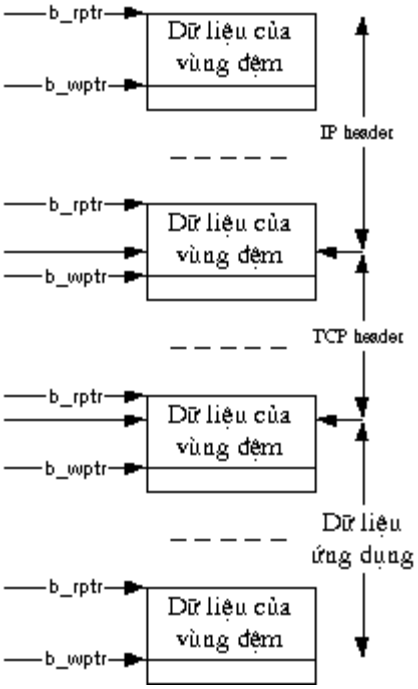
Hình 3.6 mô tả các vùng đệm của một thông báo chứa gói IP. Để mã hoá phân đoạn TCP của gói IP tại tầng IPF, cần thực hiện các việc sau:

- Xác định khối thông báo chứa gói IP.
- Xác định địa chỉ bắt đầu của phân đoạn TCP cần mã hoá.

Để xác định địa chỉ bắt đầu của phần dữ liệu cần mã ta phải xác định được vùng đệm nào chứa byte đầu tiên và địa chỉ offset của byte đó. Để làm được điều này, phải căn cứ vào độ dài IP header (ngầm định là 20 bytes) và sử dụng cấu trúc con trỏ của msgb.

Dùng thuật toán mã khối để mã hoá phân đoạn TCP.

Các thành phần của gói IP chứa trong các vùng đệm của một khối thông báo.

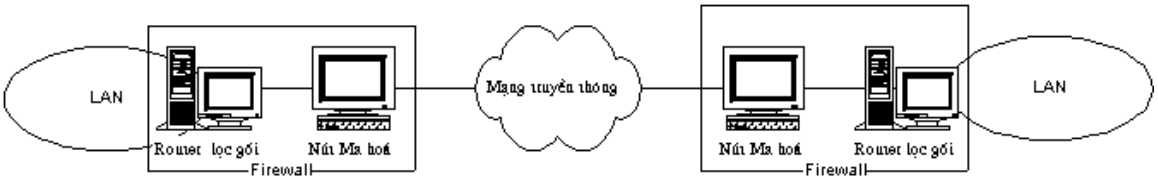


Hình 3.6: Các vùng đệm của một thông báo chứa gói IP

Việc mã hoá được thực hiện tại từng vùng đệm. Khi mã hết dữ liệu ở một vùng đệm nào đó thì hệ thống chuyển xuống vùng đệm của khối thông báo tiếp theo, cứ tiếp tục cho đến khi mã hết dữ liệu của vùng đệm cuối cùng trong thông báo.

3.1.8 Tích hợp nút mã hoá với Router lọc gói .

Chúng ta có thể tích hợp nút mã hoá gói IP với một router lọc gói để có một firewall có độ an toàn cao. Chức năng mã hoá của nút mã hoá sẽ tăng cường khả năng của firewall trong việc chống lại những tấn công đối với gói IP mà những firewall không có chức năng mật mã không thể giải quyết được. Trước hết chúng ta quan tâm đến vị trí của nút mã hoá khi kết hợp với router lọc gói.



Hình 3.7: Kết hợp router lọc gói với nút mã hoá

Khi kết hợp với router lọc gói, nút mã hoá sẽ nằm ở bên ngoài router lọc gói. Vì hệ thống mã hoá cả phân đoạn TCP nên những thông tin về TCP/UDP header bị che lấp. Nếu nút mã hoá nằm trong router lọc gói thì router lọc gói không thể dùng các luật liên quan đến những thông tin trong TCP/UDP header như số cổng nguồn, số cổng đích, các cờ,...

Nếu chúng ta chỉ mã hoá phần dữ liệu ứng dụng trong gói IP thì nút mã hoá có thể đặt ở trong bộ router lọc gói. Khi tích hợp nút mã hoá và router lọc gói trong một firewall sẽ có những ưu điểm là giúp firewall chống lại những tấn công đối với router lọc gói :

- Giả địa chỉ một máy được router lọc gói uỷ quyền

Giả sử một gói IP từ một máy có địa chỉ IP nguồn là "giả" (dùng địa chỉ IP của một máy được uỷ quyền của router lọc gói), vì máy gửi gói IP "giả" không thuộc hệ thống an toàn của chúng ta nên gói IP "giả" không được mã hoá nhưng khi qua nút mã hoá, thông tin trong phân đoạn TCP của gói IP "giả" vẫn bị giải mã dẫn đến thông tin của gói IP bị sửa đổi, chính vì vậy gói sẽ bị huỷ ngay tại router lọc gói nếu router lọc gói thực hiện các luật liên quan đến thông tin của TCP/UDP header. Còn nếu router lọc gói không dùng các luật liên quan đến thông tin của TCP/UDP header thì gói sẽ qua router lọc gói và đến máy đích. Tại đây gói sẽ bị huỷ tại tầng TCP do nhờ vào tổng kiểm tra, hoặc số cổng nguồn, cổng đích.

- Soi gói: Thông tin về phân đoạn TCP được giữ bí mật do được mã hoá.

- Sửa nội dung gói: Việc sửa nội dung gói sẽ dẫn đến sự thay đổi giá trị các thành phần trong TCP header và tầng TCP của máy đích sẽ không chấp nhận gói với các lý do như tổng kiểm tra sai, số cổng đích sai, giá trị cờ sai,...

CHƯƠNG IV

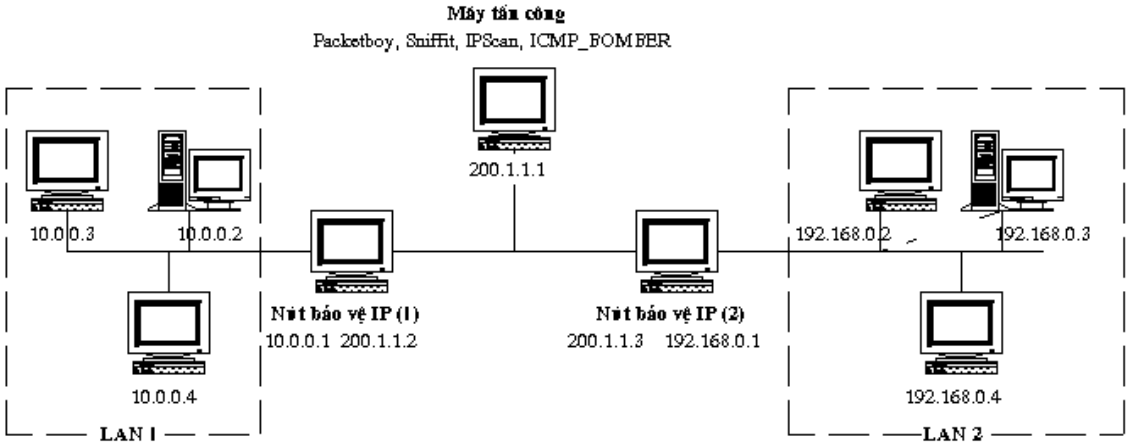
KHẢO SÁT KHẢ NĂNG CHỐNG LẠI CÁC PHẦN MỀM HACKER VÀ TỐC ĐỘ TRUYỀN DỮ LIỆU CỦA HỆ THỐNG BẢO VỆ GÓI IP TRÊN SOLARIS

Chương này giới thiệu kết quả khảo sát một số đặc trưng của bộ phần mềm bảo vệ gói IP bằng kỹ thuật mật mã (IPSEC_SUN) trên Solaris như sau:

- Khả năng ngăn chặn các tấn công của một số phần mềm Hacker
- Thời gian và tốc độ truyền dữ liệu của hệ thống trong các trường hợp không chạy và chạy phần mềm IPSEC_SUN

Phần này cũng giới thiệu các đặc tính trên của bộ phần mềm FreeS/WAN với tư cách như một sản phẩm đối chứng.

4.1 KHẢO SÁT KHẢ NĂNG BẢO VỆ GÓI IP TRÊN MẠNG LAN CỦA BỘ PHẦN MỀM IPSEC_SUN



Hình 4.1: Mô hình thử nghiệm bộ phần mềm IPSEC_SUN trên mạng LAN

Hình 4.1 là mô hình mạng thử nghiệm các chức năng của bộ phần mềm IPSEC_SUN. Hệ thống mạng được dùng để khảo sát gồm hai phân đoạn mạng được kết nối với nhau qua cáp mạng, ta ký hiệu là LAN 1 và LAN 2. Mỗi phân đoạn có một Gateway có chức năng bảo vệ gói IP bằng phần mềm IPSEC_SUN và được gọi là nút bảo vệ gói IP. Các ứng dụng được cài đặt trên mạng bao gồm: hệ thư điện tử Lotus Notes, WEB, truyền tệp FTP, Cơ sở dữ liệu Oracle, Netmeeting với ảnh động và âm thanh, môđun thu phát gói Multicast. Trên kênh truyền giữa hai Gateway có càI đặt một máy tấn công mạng với một giao diện mạng. Có 4 phần mềm được càI đặt trên trạm tấn công là Packeyboy 1.3, Sniffit V.0.3.5. , IPScan và ICMP_BOMBER. Trong đó ba phần mềm Packeyboy 1.3, Sniffit V.0.3.5., IPScan có chức năng chặn bắt thông tin và phần mềm ICMP_BOMBER có chức làm cạn kiệt tàI nguyên hệ thống hay còn gọi là kiểu tấn công từ chối dịch vụ.

Sau đây là những kết quả thu được trong việc khảo sát khả năng ngăn chặn các tấn công dùng các phần mềm Packeyboy 1.3, Sniffit V.0.3.5. , IPScan và ICMP_BOMBER của IPSEC_SUN.

4.1.1 Khả năng ngăn chặn các tấn công bằng phần mềm Sniffit V.0.3.5

Phần mềm Sniffit V.0.3.5 làm việc trên môI trường LINUX, cho phép thu được toàn bộ dữ liệu giữa hai ứng dụng được truyền trên mạng như thư đIện tử, trang WEB, file dữ liệu, dữ liệu chatting, mật khẩu và tên người dùng, ...

Dưới đây là một số ví dụ về khả năng chặn bắt thông tin của Sniffit V.0.3.5.

4.1.1.1 Các thông tin về tên, mật khẩu người dùng, các lệnh của dịch vụ truyền tệp FTP

USER root

```
pass admin
TYPE I
REST 100
REST 0
CWD /root/
pwd
TYPE A
PORT 192,168,1,7,4,27
LIST
CWD /
pwd
TYPE I
PORT 192,168,1,7,4,28
```

```
REST 29962240
RETR filetest
ABOR
```

4.1.1.2 Nội dung của một file dạng text được truyền bởi FTP

#include <stdio.h>

```
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <iostream.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
main()
{
      char a[20];
      int handle,i,n=5;
      int *p;
      cout <<"Enter the string for a: ";
      cin >> a;
      cout << "\n " <<a << endl;
      handle  =  open("/root/data.txt",O_CREAT  |  O_RDWR  |S_IREAD  |
S_IWRITE);
      if (handle == -1)
            {
            cerr << "I/O error 1"<<endl;
            exit (1);
            }
      for (i=1;i<=n;i++)
            {
            cout << " "<<i;
            *p = i;
            write(handle,p,1);
            }
      close(handle);
      cout <<endl;
      return 0;
}
```

3.1.1.3 Kết quả của lệnh ls -l trên máy khách hàng của dịch vụ FTP

```
total 18756
-rw-------    1 root     root         2009 May  9 23:11 .ICEauthority
-rw-------    1 root     root          265 May  9 23:09 .Xauthority
-rw-r--r--    1 root     root         1126 Aug 24  1995 .Xdefaults
-rw-------    1 root     root        11676 Jul 20 09:57 .bash_history
-rw-r--r--    1 root     root           24 Jul 14  1994 .bash_logout
-rw-r--r--    1 root     root          238 Aug 24  1995 .bash_profile
-rw-r--r--    1 root     root          267 May  9 22:33 .bashrc
```

```
drwx------    2 root     root         4096 May 13 03:30 .cedit
-rw-r--r--    1 root     root          182 Mar 22  1999 .cshrc
drwxr-xr-x    3 root     root         4096 Apr  5 22:43 .eXtace
drwx------   3 root     root         4096 May  9 23:11 .enlightenment
drwx------    6 root     root         4096 May  9 23:11 .gnome
drwxr-xr-x   2 root     root         4096 Mar 22 01:41 .gnome-desktop
drwxr-xr-x   2 root     root          4096 Mar 21 23:53 .gnome-help-
browser
drwx------   2 root     root         4096 Mar 21 23:52 .gnome_private
-rw-------    1 root     root        12288 Apr  7 05:26 .hello.c.swp
drwxr-xr-x    3 root     root         4096 Jul 20 09:36 .mc
drwxr-xr-x    5 root     root         4096 Apr 14 03:50 .netscape
-rw-r--r--    1 root     root          166 Mar  4  1996 .tcshrc
-rw-r--r--    1 root     root        27999 May 13 04:49 README.FIRST
-rw-r--r--    1 root     root         1259 Apr 21 00:41 SSS
drwxr-xr-x    2 root     root         4096 Apr 20 05:45 So_nguyento
-rw-r--r--  1 root     root        30720 Apr  6 21:55 So_nguyento.tar
-rw-r--r--    1 root     root         2442 Apr 14 03:02 bimat
drwxr-xr-x    3 root     root         4096 May 13 03:27 config
drwxr-xr-x    4 root     root         4096 Apr 25 03:27 config.old1
drwxr-xr-x    3 root     root         4096 Apr  1 05:22 config_luu
-r-sr-----    1 root     root            0 Apr  7 05:25 data.txt
-rwxr-xr-x    1 root     root        13790 Apr  9 23:15 expfile
-rwxr-xr-x    1 root     root          209 Apr 21 02:58 free_vers
drwxr-xr-x  10 root   root         4096 Apr  2 02:49 freeswan.idea.ok
-rwxr-xr-x 1 root root      6778880 Apr  2 02:13 freeswan_idea_ok.tar
drwxr-xr-x    2 root     root         4096 Mar 22 20:05 h1
-rwxr-xr-x    1 root     root        13639 Apr  7 05:25 hello
-rw-r--r--    1 root     root          551 Apr  7 05:23 hello.c
drwxr-xr-x 2 root     root         4096 Apr  6 05:45 install_Freeswan
drwxr-xr-x    2 root     root         4096 Apr 14 00:18 keys
-rw-r--r-- 1 root     root        97455 May  7 04:40 ldthich_3105.taz
-rw-------    1 root     root         1447 Apr  6 04:21 mbox
drwx------    2 root     root         4096 Mar 22 00:04 nsmail
-rw-r--r--    1 root     root            0 Apr  9 23:15 output.txt
-rw-r--r--1 root     root        69539 Mar 31 07:29 pfkey_v2_parser.c
-rw-r--r--    1 root     root      2652160 Mar 31 03:27 pluto.tar
```

Nguyên tắc hoạt động của Sniffit là chặn bắt các gói IP được truyền trên kênh, sau đó dựa vào giá trị của cổng ứng dụng đích trong header tầng vận tảI để khôI phục được toàn bộ dữ liệu cuả ứng dụng từ các gói IP thu được. Khi header tầng vận tảI được mã hoá làm cho thông tin về cổng ứng dụng bị che lấp, chính vì vậy Sniffit không thể khôI phục được dữ liệu của ứng dụng. Qua khảo sát chúng tôI thấy rằng khi không chạy IPSEC_SUN, phần mềm Sniffit hoạt động rất tốt và chặn bắt được toàn bộ dữ liệu của ứng dụng như thư đIện tử, nội dung file được truyền, trang WEB, thông tin người dùng,... Nhưng khi chạy phần mềm

IPSEC_SUN, phần mềm Sniffit sẽ bị cheo và không thể cung cấp được thông tin gì.

Kết luận : Phần mềm IPSEC_SUN có khả năng làm vô hiệu hoá các tấn công dùng phần mềm Sniffit.

4.1.2 Khả năng ngăn chặn các tấn công bằng phần mềm IPSCAN

Phần mềm IPScan làm việc trên môI trường LINUX, cho phép người sử dụng có thể giám sát hoạt động của các cổng TCP/UDP trên một hoặc nhiều trạm ở xa. Nó có khả năng quét các cổng TCP/UDP của một trạm trên mạng và cho biết tại thời điểm đó có bao nhiêu ứng dụng đang chạy và chạy tại cổng số nào.

Việc kết hợp IPScan với Sniffit cho phép tạo ra các tấn công hiệu quả. Trong đó IPScan có vai trò cung cấp cho Sniffit giá trị các cổng và giao thức tương ứng đang có giao dịch để Sniffit tiến hành chặn bắt thông tin. Dưới đây là ví dụ về thông tin do phần mềm IPSCAN xác định được, bao gồm địa chỉ IP, giá trị cổng, giao thức.

```
200.1.1.2       echo          7/tcp                        Echo
[95,JBP]
200.1.1.2       discard       9/tcp                      Discard
[94,JBP]
200.1.1.2       daytime       13/tcp                     Daytime
[93,JBP]
200.1.1.2        chargen       19/tcp      Character Generator
[92,JBP]
200.1.1.2        ftp           21/tcp           File Transfer
[Control] [96,JBP]
200.1.1.2       telnet        23/tcp                     Telnet
[112,JBP]
200.1.1.2        smtp           25/tcp   Simple Mail Transfer
[102,JBP]
200.1.1.2       time          37/tcp                       Time
[108,JBP]
200.1.1.2       finger        79/tcp                     Finger
[52,KLH]
200.1.1.2       sunrpc       111/tcp SUN Remote Procedure Call
[DXG]
200.1.1.2        exec           512/tcp         remote process
execution (rexecd)
200.1.1.2        login         513/tcp           remote login
(rlogind)
200.1.1.2       shell         514/tcp        rlogin style exec
(rshd)
```

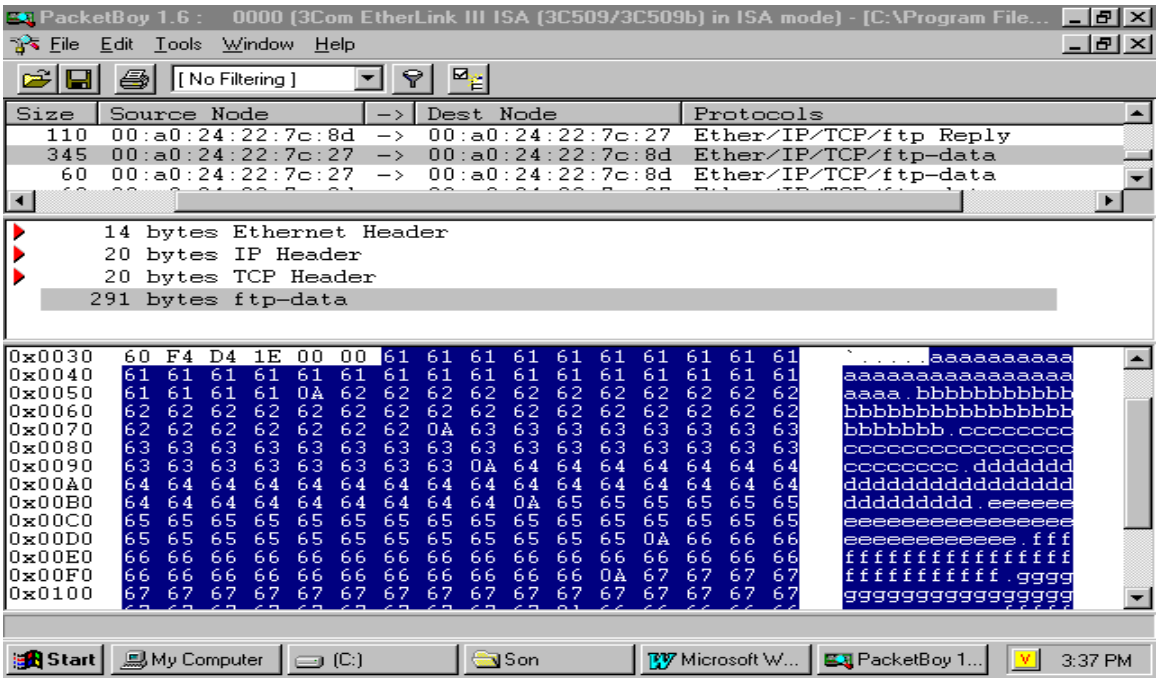
```
200.1.1.2                    uucp                         540/tcp
uucpd
200.1.1.2                     unknown                     2525/tcp
unassigned
200.1.1.2                     unknown                     4045/tcp
unassigned
```

Tương tự như đối với Sniffit, khi header tầng vận tảI bị mã hoá phần mềm IPScan không thể xác định được cổng ứng dụng nào được dùng đối với giao thức tầng vận tải. Chính vì vậy nó không thể hoạt động được và bị vô hiệu hoá.

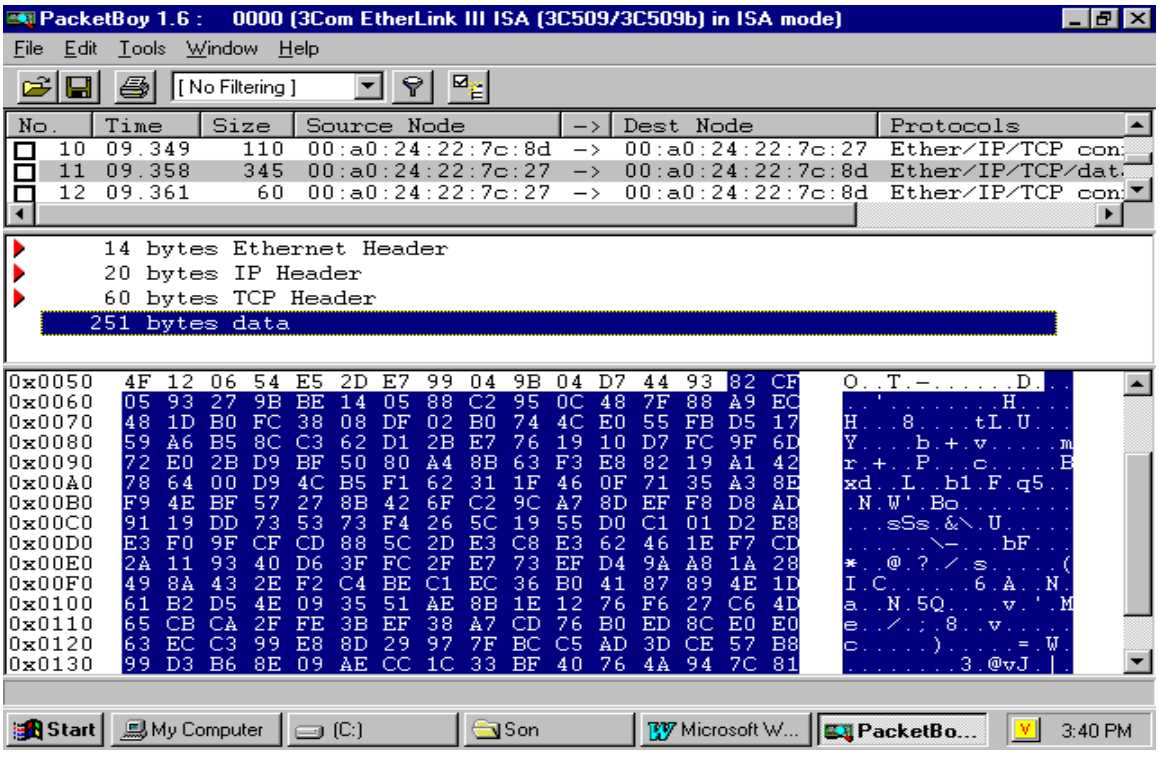
Nhận xét : Phần mềm IPSEC_SUN có khả năng ngăn chặn các tấn công dùng phần mềm IPScan.

4.1.3 Khả năng ngăn chặn các tấn công bằng phần mềm Packetboy

Phần mềm Packetboy làm việc trên môi trường Windows, cho phép chặn bắt và hiển thị toàn bộ thông tin về các thành phần của Ethernet frame bao gồm Ethernet header, IP header, TCP(UDP) header, dữ liệu ứng dụng. Khác với hai phần mềm Sniffit và IPSCan, khi chạy IPSEC_SUN phần mềm Packetboy vẫn hoạt động bình thường và hiển thị thông tin về các gói IP mà nó chặn bắt được. Chính vì vậy, với Packetboy kẻ tấn công vẫn thu được Frame chứa gói IP với một số thành phần bị mã, nhưng họ không thể đọc được nội dung của TCP(UDP) header và dữ liệu ứng dụng mà chỉ có thể đọc được các thông tin trong IP header và Ethernet header. Dưới đây là một số hình ảnh minh hoạ khả năng chặn bắt gói IP của phần mềm Packetboy và chức năng mã hoá các thành phần gói IP của IPSEC_SUN.

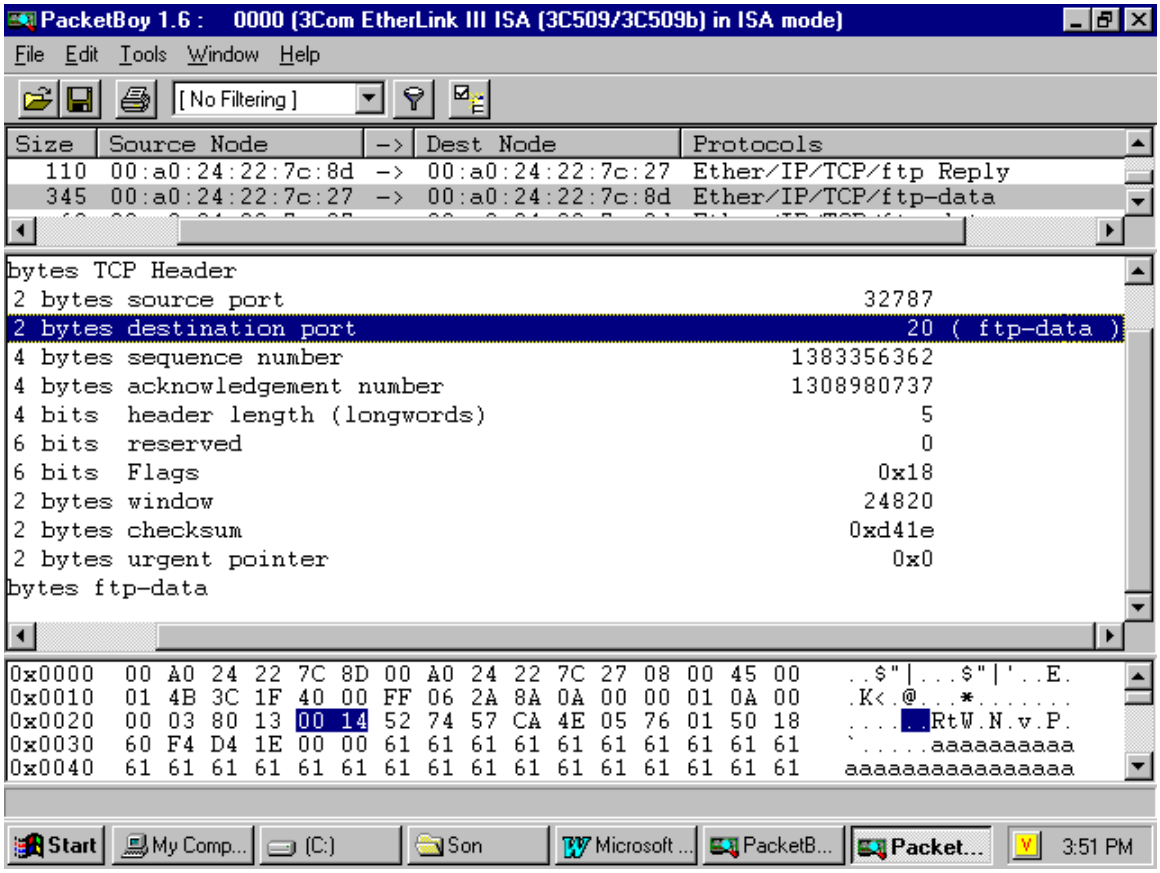


Hình 4.2 : Dữ liệu ứng dụng trong gói IP trước khi mã hoá



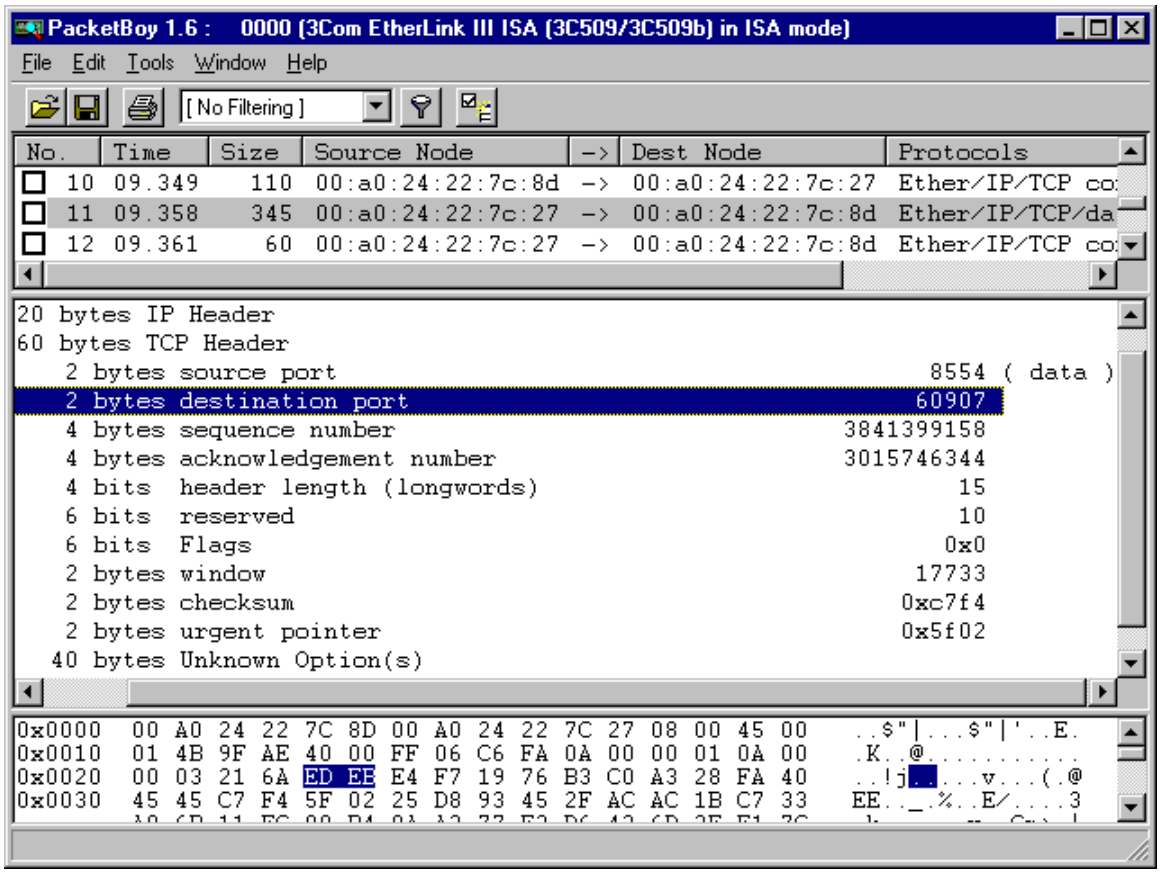
Hình 4.3 : Dữ liệu ứng dụng trong gói IP trước và sau khi mã hoá

Trong hình 4.3 , dữ liệu của gói IP đã được mã hoá nên kẻ tấn công không thể đọc được nội dung thông tin của nó. Tuy nhiên Packetboy vẫn cho phép đọc các thông tin trong Ethernet header và IP header. Riêng thông tin về TCP header đã bị mã hoá.



Hình 4.4 Thông tin về 20 bytes của TCP header trong gói IP khi chưa mã

Hình 4.4 chứa các thông tin về các thành phần của một TCP header, trong đó giá trị của cổng đích trong phiên FTP-data là 20.

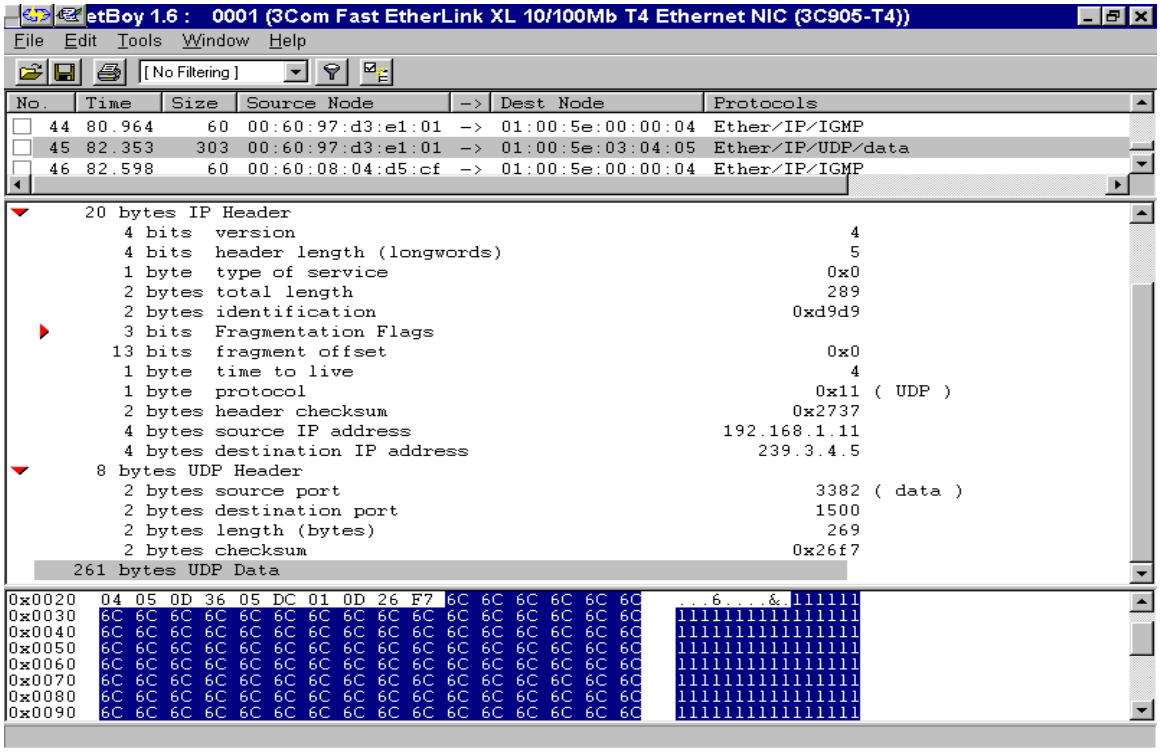


Hình 4.5: Thông tin về 20 bytes của TCP header sau khi mã

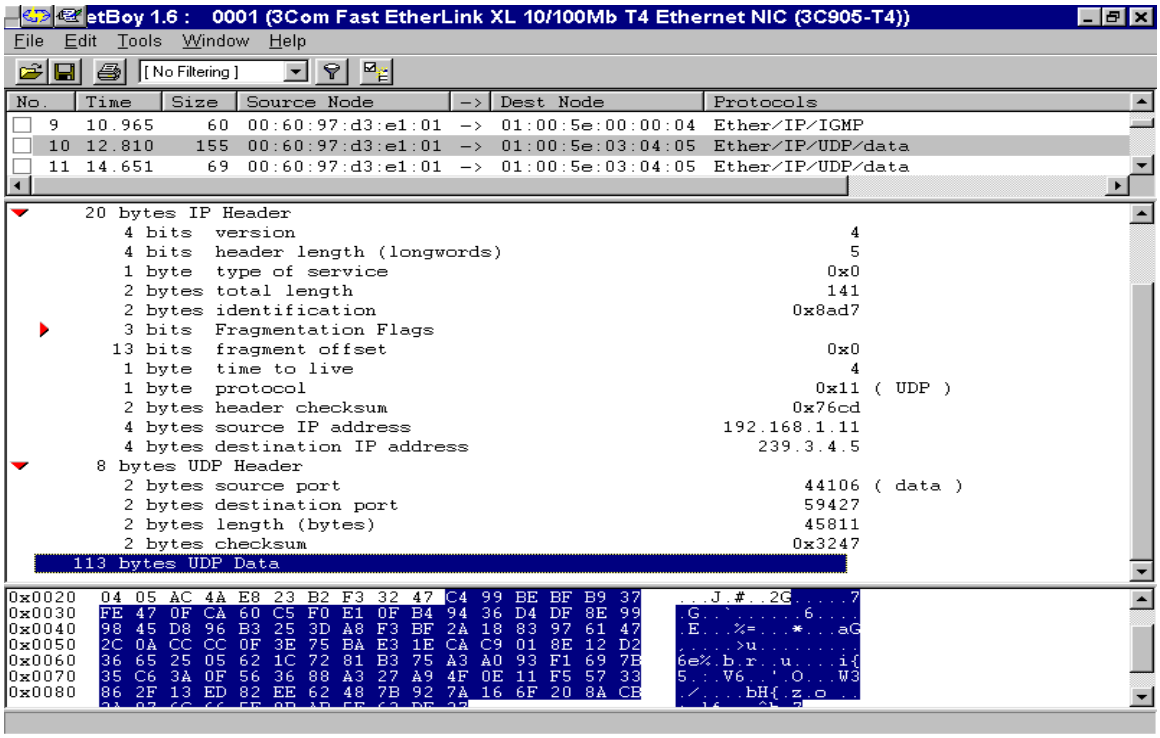
Hình 4.5 chứa các giá trị của các trường trong TCP header đã bị mã hoá, trong đó giá trị của cổng đích trong phiên giao dịch FTP-data là 60907. Lý do là vì các bít biểu diễn giá trị của cổng đích trong phiên FTP-data luôn là 20 dưới dạng nhị phân đã bị mã hoá để trở thành dãy nhị phân khác tương ứng với giá trị 60907.

Nhận xét : Phần mềm IPSEC_SUN có khả năng bảo vệ toàn bộ phân đoạn tầng vận tảI của gói IP trước tấn công bằng phần mềm Packetboy.

Hình 4. 6 là hình ảnh một gói Multicast khi chưa can thiệp mật mã.



Hình 4.6: Các thành phần của gói Multicast trước khi mã

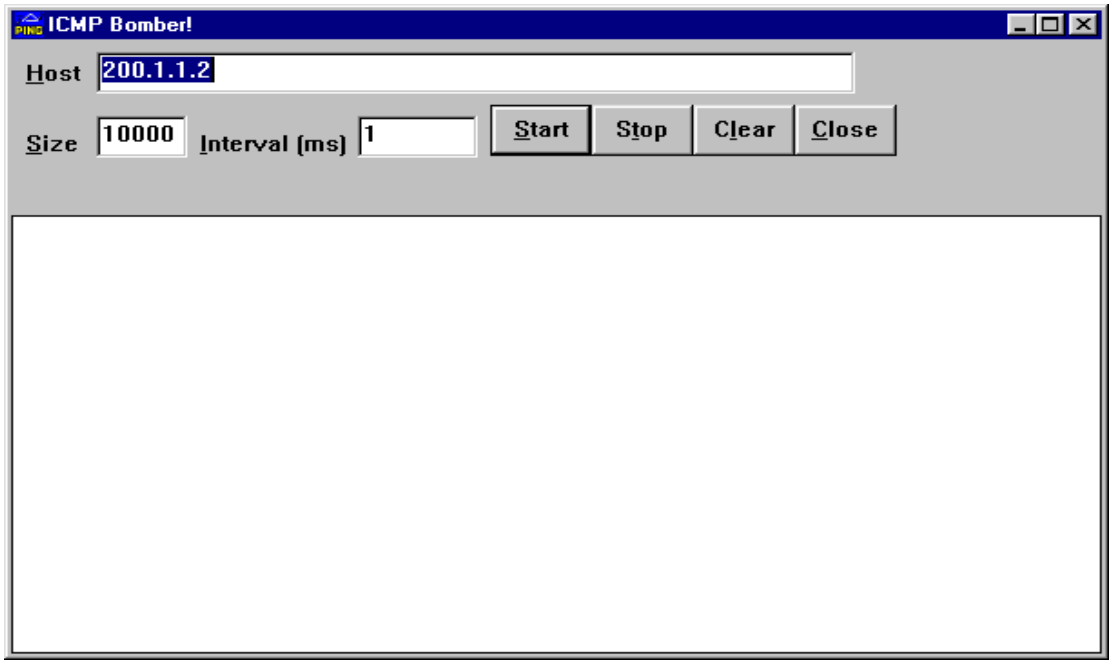


Hình 4.7 : Dữ liệu ứng dụng của một gói Multicast đã được mã

4.1.4 Khả năng ngăn chặn các tấn công bằng phần mềm ICMP_ Bomber

ICMP_Bomber là một phần mềm tiêu biểu cho một loạt các phần mềm mà các Hacker chuyên dùng để tấn công các hệ thống an toàn thông tin theo kiểu "từ chối dịch vụ" hay còn gọi là các phần mềm làm lụt hệ thống (flooder). Nó hoạt động dựa trên nguyên tắc là khi được kích hoạt, nó sẽ liên tiếp gửi tới máy đích (máy bị tấn công) các gói tin ICMP và yêu cầu máy đích trả lời, trong khi tự thân nó sẽ không nhận các gói trả lời này. Khi số gói tin được gửi tới là lớn, lúc đó máy bị tấn công sẽ phải liên tục xử lý và dẫn đến hiện tượng bị suy kiệt dần tài nguyên. Nếu sử dụng đồng thời nhiều máy để tấn công vào một máy chủ cung cấp dịch vụ thì dịch vụ đó sẽ bị tê liệt.

ICMP_Bomber được viết để chạy trên các hệ thống Windows, ta chỉ cần sao chép file ICMP_Bomber.exe vào đĩa cứng hoặc đĩa mềm là có thể chạy được ngay. Giao diện của chương trình như sau:



Hình 4.8: Giao diện người dùng của ICMP BOMBER

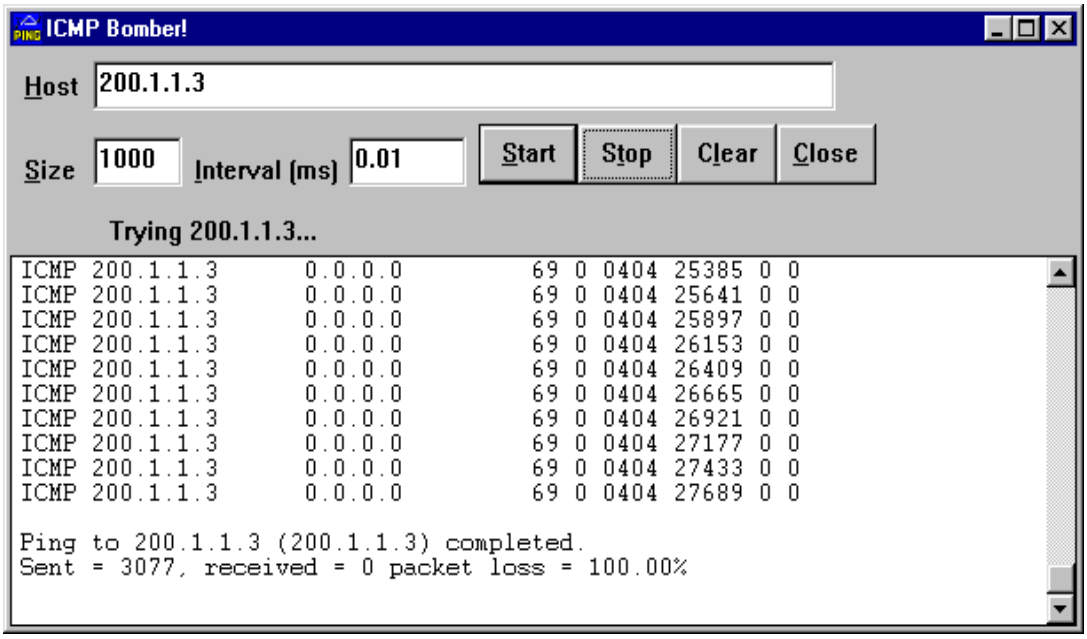
Các tham số để chạy bao gồm:

- Host: Địa chỉ IP của máy bị tấn công
- Size: Độ lớn của gói tin gửi đi

- Interval: Khoảng thời gian giữa các lần gửi gói tin (Tốc độ gửi), được tính bằng mili giây.

Các lệnh chạy:

- Start: Bắt đầu chạy ICMP_Bomber
- Stop: Dừng chạy ICMP_Bomber



Hình 4.9: Kết quả khi chạy

Qua quá trình khảo sát khả năng tấn công của ICMP_bomber đối với các hệ thống an toàn thông tin, ta thu được các kết quả sau: Khi ta sử dụng máy Windows98 (có cấu hình như sau: Processor Pentium Celeron 333 Mhz, 32 Mb RAM, 7.5 Gb HDD, ...) để kích hoạt phần mềm ICMP_Bomber để tấn công một máy chủ Sun Slaris (IP address: 200.1.1.3, Cấu hình phần cứng : Processor Pentium Celeron 300, 32 Mb RAM, 7.5 Gb HDD) theo các tham số sau:

- **Host** : 200.1.1.3 (Đây chính là địa chỉ IP của máy chủ Sun)
- **Size** : 1000 (Độ lớn dữ liệu của mỗi gói là 1000 byte)
- **Interval**: 0.01 (Khoảng cách giữa 2 lần gửi là 0.01 mili giây)

Ta thấy rằng, trong cả 2 trường hợp là có chạy và không chạy phần mềm bảo vệ gói IP thì máy chủ Sun đều chịu một sự tấn công như nhau, hiện tượng nhận biết được là tốc độ xử lí của máy chậm hẳn lại. Khi ta tiếp tục gia tăng sức tấn công bằng cách cho chạy ICMP_Bomber trên một máy Windows NT (Có cấu hình phần cứng như sau: Processor PII 450, 64 Mb RAM, 9.2 Gb HDD,..) để tấn công vào máy chủ Sun ở trên thì kết quả là máy chủ sun hoàn toàn bị tê liệt, không thể đáp ứng được các dịch vụ mạng thông thường nữa (ta có thể kiểm nghiệm bằng cách từ một máy ở trên mạng ftp hoặc telnet vào địa chỉ 200.1.1.3).

Nhận xét : Phần mềm IPSEC_SUN không có khả năng ngăn chặn các tấn công kiểu từ chối dịch vụ dùng phần mềm ICMP_BOMBER.

4.1.4 So sánh khả năng chống lại các phần mềm tấn công của bộ phần mềm IPSEC_SUN và bộ phần mềm FreeS/WAN

FreeS/WAN là một sản bảo vệ gói IP trên mạng LINUX, được xây dựng dựa trên chuẩn IPSEC và có mã nguồn mở. Giao thức IPsec cung cấp chức năng an toàn, các dịch vụ xác thực (*authentication)* và mã hoá (*encryption*) ở mức IP (*Internet Protocol*) của chồng giao thức mạng, đồng thời nó yêu cầu một giao thức IKE ở mức cao hơn mức IP để thiết lập các dịch vụ ở mức IP. Được bắt đầu với phiên bản 1.0, đến nay đã có phiên bản 1.9. FreeS/WAN dùng kỹ thuật mã khối DES, 3DES trong mode CBC để để mã hoá dữ liệu , dùng phép biến đổi HMAC với các hàm Hash là MD5, SHA-1 để xác thực dữ liệu. Việc trao đổi khoá được càI đặt dùng giao thức IKE với giao thức Diffie-Hellman 768, 1024 and 1512 bits được xác thực bằng cách dùng khoá bí mật chia sẻ hoặc chữ ký số RSA

Có 3 giao thức được dùng trong **IPsec** đó là:

AH (Authentication Header): cung cấp một dịch vụ xác thực ở mức gói (packet-level).

ESP (Encapsulating Security Payload): cung cấp dịch vụ mã hoá cộng với xác thực dữ liệu.

IKE (Internet Key Exchange): thoả thuận các tham số của kết nối, bao gồm các khoá cho hai đối tượng khác nhau. Hạn chế của **IPsec** là không đáng kể, có trường hợp IPsec dùng cả 3 giao thức trên, có trường hợp chỉ dùng AH và ESP.

Hai chế độ làm việc của FreeS/WAN là chế độ tunnel và chế độ transport. Trong chế độ tunnel toàn bộ gói IP được bảo vệ còn trong chế độ transport phân đoạn tầng vận tảI cùng một số trường không thay đổi trong IP header được bảo vệ

Thông tin về FreeS/Wan được giới thiệu trong http://www.freeswan.org/

Trước hết chúng tôi giới thiệu một số kết quả trong việc thiết kế xây dựng bộ phần mềm bảo vệ gói IP trên LINUX theo công nghệ IPSEC dựa trên mã nguồn mở của FreeS/WAN. Cụ thể là giữ nguyên thiết kế hệ thống hiện có, chỉ thay thế, bổ xung các module mật mã như các module mã khối và các tham số mật như các số nguyên tố cho thuật toán *Diffie-Hellman* và *RSA*

Các công việc mà chúng tôi đã hoàn thành là:

-Đã khảo sát được cấu trúc hệ thống và cơ chế hoạt động của FreeS/Wan

-Thiết kế, xây dựng và tích hợp mô đun của kỹ thuật mã khối IDEA phục vụ cho việc bảo vệ dữ liệu

-Thiết kế, xây dựng mô đun sinh các tham số (bao gồm khoá bí mật và khóa công khai) phục vụ cho việc tạo chữ ký số RSA với đầu vào là các cặp số nguyên tố p,q của Học viện Kỹ thuật mật mã (512 bits và 1024 bits).

-Tích hợp được 3 số nguyên tố lớn của Học viện Kỹ thuật mật mã phục vụ cho việc xác thực lẫn nhau giữa các máy chủ LINUX, thoả mãn đầy đủ các yêu cầu an toàn (các số nguyên tố có dạng p=2q+1, trong đó q là các số nguyên tố và 64 bit cao và 64 bit thấp có giá trị bằng 1).

-Xây dựng xong bộ phần mềm bảo vệ gói IP trên một phiên bản của LINUX là RedHad 6.2 theo công nghệ IPSEC dựa trên FreeS/Wan 1.5 với các mô đun và tham số mật mã của Học viện kỹ thuật mật mã.

-Cài đặt kiểm tra nhiều lần bộ phần mềm trên hệ thống gồm nhiều máy chủ LINUX và các mạng LAN phía sau kết nối qua cáp mạng.

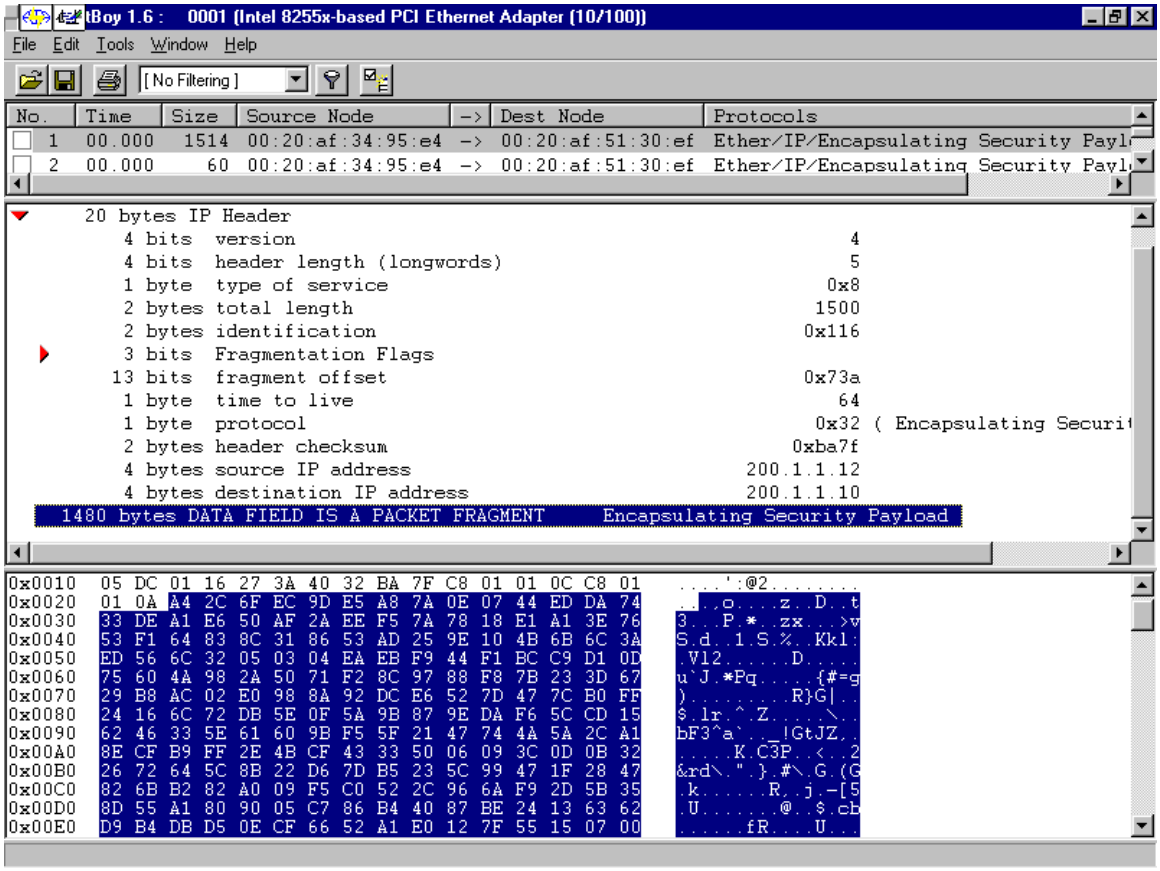
-Xây dựng quy trình tạo, quản lý và phân phối các tham số của hệ mật khoá công khai RSA cho các máy chủ LINUX bảo vệ gói IP trên mạng diện rộng

-Xây dựng quy trình cài đặt và cấu hình bộ phần mềm bảo vệ gói IP trên mạng diện rộng

Cũng với sự khảo sát tương tự như đối với phần mềm IPSEC_SUN, chúng tôI đã khảo sát khả năng của FreeS/WAN trong việc chống lại các tấn công bằng bốn phần mềm Packeyboy 1.3, Sniffit V.0.3.5. , IPScan và ICMP_BOMBER và đã thu được kết quả sau:

Phần mềm FreeS/WAN có khả năng chống lại các tấn công dùng các phần mềm Sniffit, IPScan, Packetboy và không có khả năng chống lại tấn công dùng phần mềm ICMP_BOMBER. Tuy nhiên, có một sự khác nhau giữa khả năng ngăn chặn tấn công của hai phần mềm FreeS/WAN và IPSEC_SUN là trong FreeS/WAN ở chế độ Tunnel, IP header của gói IP ban đầu được che dấu (do toàn bộ gói IP được mã hoá), thay vào đó là một IP header mới với địa chỉ IP là địa chỉ của hai Gateway.

Dưới đây là hình ảnh minh hoạ về gói IP đã được càI đặt các dịch vụ an toàn bằng FreeS/WAN do Packetboy chặn bắt được.



Hình 4.10: Gói IP đã được càI đặt dịch vụ an toàn trong chế độ Tunnel với giao thức ESP của FreeS/WAN

Khi chạy FreeS/WAN, tất cả các gói dữ liệu được truyền từ client 1 đến client 2 đã được chặn bắt, mã hoá và đóng gói lại tại Gateway 1. Một IP header mới được bổ xung với địa chỉ nguồn và đích là địa chỉ IP của hai Gateway. Trong hình trên, packetboy đã đọc được các thông tin của IP header mới với địa chỉ nguồn (200.1.1.12), địa chỉ đích (200.1.1.10) là các địa chỉ của 2 gateway có cài

đặt FreeS/WAN, giao thức tầng trên của IP lại là ESP (có giá trị là 0x32). Packetboy không thể phân tích để lấy ra được các thông tin của gói IP bên trong được truyền giữa hai máy client, dữ liệu của chúng được mã hoá và chứa trong thành phần ESP (Encapsulating Security Payload).

Từ kết quả trên chúng ta có nhận xét là: Cả hai phần mềm IPSEC_SUN và FreeS/WAN đều có khả năng chống lại các tấn công chặn bắt thông tin bằng các phần mềm như Packeyboy 1.3, Sniffit V.0.3.5. , IPScan và không thể ngăn chặn các tấn công kiểu từ chối dịch vụ như ICMP_BOMBER. Cụ thể là những thành phần của gói IP đã được mã hoá sẽ được giữ bí mật khi tấn công bằng Packetboy và do thông tin về cổng ứng dụng trong header tầng vận tảI được giữ bí mật nên cả hai phần mềm Sniffit và IPScan đã bị vô hiệu hoá và không thể hoạt động được. Còn đối với phần mềm ICMP_BOMBER, tàI nguyên hệ thống bị cạn kiệt do phảI đáp ứng lại rất nhiều các "đòi hỏi giả" và cả hai phần mềm IPSEC_SUN và FreeS/WAN đều không có khả năng chống lại được.

Từ các kết quả khảo sát ở trên, chúng ta có kết luận sau:

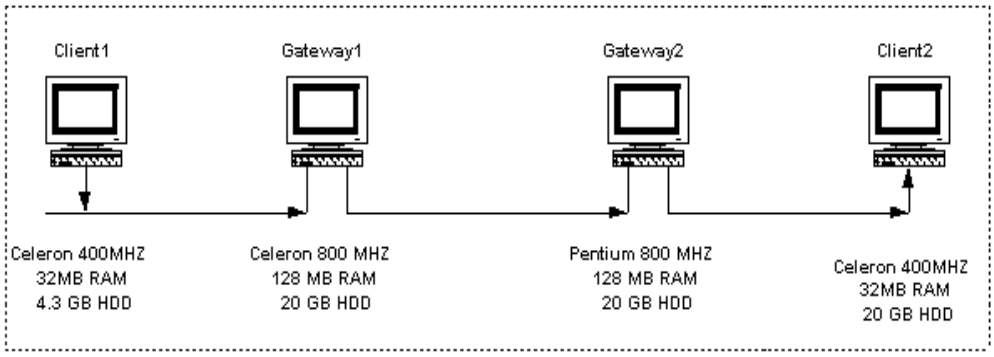
- Các ứng dụng vẫn làm việc ổn định khi các gói IP được bảo vệ tại hai Gateway bằng phần mềm IPSEC_SUN.
- Hệ thống có khả năng chặn bắt và can thiệp mật mã vào các gói IP.
- Hệ thống có khả năng chống lại các tấn công chặn bắt thông tin dùng các phần mềm Packetboy 1.3, Sniffit V.0.3.5 và IPScan.
- Hệ thống không có khả năng chống lại tấn công dùng phần mềm ICMP_BOMBER.

4.2 KHẢO SÁT SỰ ẢNH HƯỞNG CỦA BỘ PHẦN MỀM IPSEC_SUN ĐỐI VỚI THỜI GIAN TRUYỀN DỮ LIỆU CỦA MỘT SỐ DỊCH VỤ.

Để cài đặt các dịch vụ an toàn đối với các gói IP, IPSEC_SUN phải tiến hành chặn bắt và can thiệp mật mã vào phân đoạn tầng vận tải. Mục tiêu khi xây dựng phần mềm an toàn bằng kỹ thuật mật mã là việc càI đặt các dịch vụ an toàn sẽ làm ảnh hưởng ít nhất tới hiệu năng hệ thống. Chúng tôi đã khảo sát sự ảnh hưởng của bộ phần mềm IPSEC_SUN và bộ phần mềm FreeS/WAN đối với thời gian truyền dữ liệu của dịch vụ truyền tệp FTP.

4.2.1 Khảo sát sự ảnh hưởng của bộ phần mềm IPSEC_SUN đối với thời gian truyền dữ liệu của dịch vụ truyền tệp FTP (File transfer Protocol)

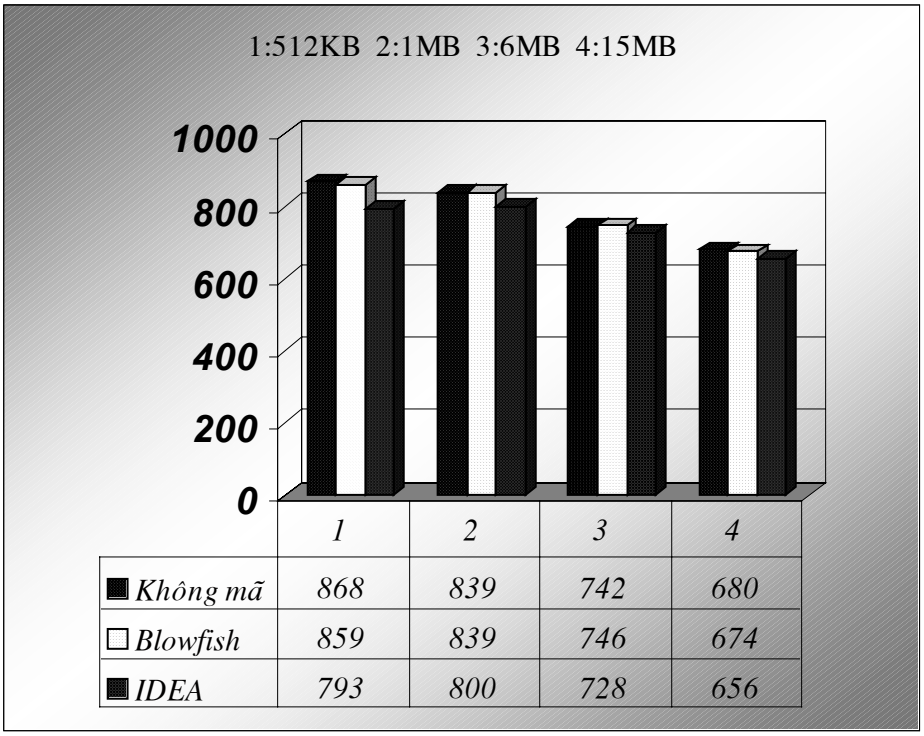
Hình 4.11 là mô hình mạng được dùng để khảo sát, bao gồm 2 Gateway có cài đặt bộ phần mềm IPSEC_SUN trên hệ điều hành Solaris và hai máy client 1 và client 2 chạy trên hệ đIều hành LINUX 6.2 dùng để truyền số liệu dùng dịch vụ FTP. Với mỗi file dữ liệu chúng tôI đã tiến hành truyền 5 lần để đo thời gian và tốc độ và có được các kết quả khác nhau từ thông báo của dịch vụ FTP.



Hình 4.11: Mô hình mạng khảo sát

Bảng sau là kết quả thu được qua khảo sát, trong đó ghi thời gian truyền được tính bằng giây của 5 lần truyền file trong ba trường hợp không mã, mã dùng Blowfish, mã dùng IDEA :

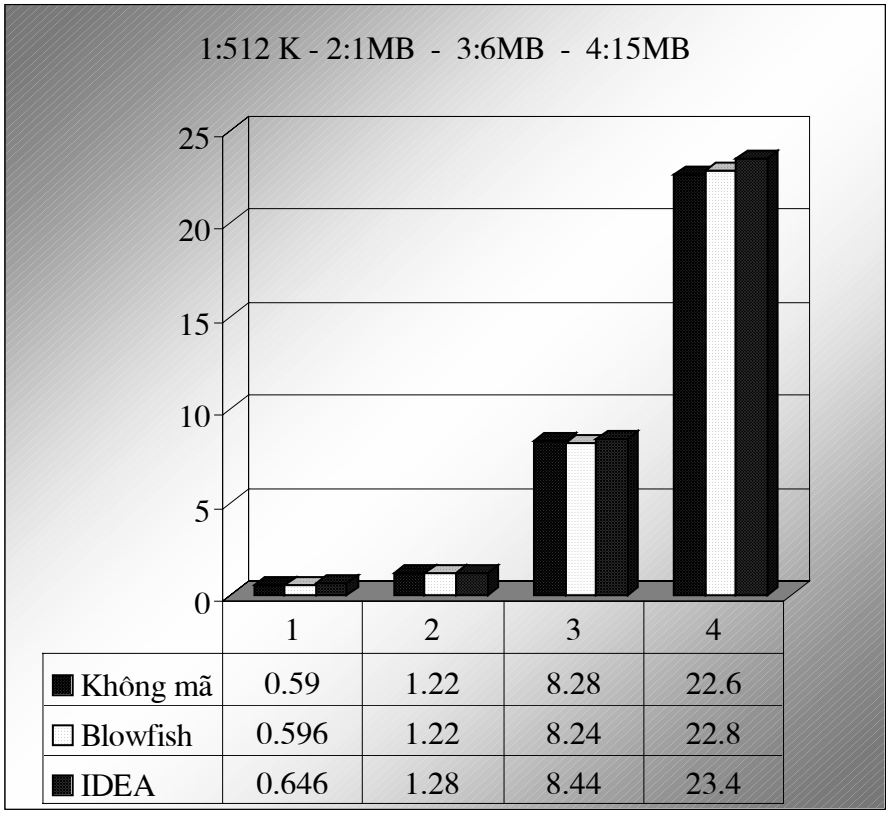
Kích thước	Thời gian truyền (s), tốc độ (Kytes/s)		
	Không mã	**Blowfish**	**IDEA**
512KB	0.54 0.57 0.61 0.63 0.6	0.63 0.59 0.58 0.57 0.61	0.74 0.62 0.61 0.61 0.65
Trung bình	***0.59 (868)***	***0.596 (859)***	***0.646 (793)***
1MB	1.2 1.1 1.3 1.3 1.2	1.2 1.1 1.3 1.3 1.2	1.4 1.3 1.3 1.2 1.2
Trung bình	***1.22 (839)***	***1.22 (839)***	***1.28 (800)***
6MB	8.4 8.3 7.8 8.9 8	7.6 8.5 9.4 8.1 7.6	9.4 7.4 8.3 8.8 8.3
Trung bình	***8.28 (742)***	***8.24 (746)***	***8.44 (728)***
15MB	25 24 22 22 20	23 22 23 22 24	24 24 22 23 24
Trung bình	***22.6 (680)***	***22.8 (674)***	***23.4 (656)***



Hình 4.12: Biểu đồ về thời gian truyền dữ liệu của hệ thống càI đặt IPSEC_SUN

Hình 4.12 là biểu đồ của thời gian truyền trung bình các file dữ liệu trong các trường hợp không chạy và chạy IPSEC_SUN với IDEA và Blowfish.

Hình 4.13 là biểu đồ về tốc độ truyền dữ liệu được tính theo Kbytes/s.



Hình 4.13 Biểu đồ tốc độ truyền dữ liệu của hệ thống dùng IPSEC_SUN

Bảng dưới đây chỉ ra tỷ lệ % giữa tốc độ truyền dữ liệu khi chạy và không chạy IPSEC_SUN :

Kích thước	Không mã	Blowfish	IDEA
512 KB	100%	99%	91%
1 MB	100%	100%	95%
6 MB	100%	101%	98%
15 MB	100%	99%	96%
Trung bình		99,75%	95 %

Từ các số liệu trên chúng ta có nhận xét là:

Với một file dữ liệu, thời gian và tốc độ truyền từ client 1 đến client 2 là khác nhau.

Với mô hình mạng như trên, nói chung việc mã hoá gói IP hầu như không làm giảm tốc độ truyền số liệu giữa hai client 1 và client 2.

Khi mã với IDEA, tỷ lệ tốc độ đạt 95 %, đặc biệt khi mã với Blowfish tỷ lệ tốc độ đạt 99,75%.

Với file có kích thước càng lớn, tốc độ truyền dữ liệu có xu hướng giảm nhưng tỷ lệ % giữa tốc độ truyền dữ liệu khi chạy và không chạy IPSEC có xu hướng tăng lên

Khi sử dụng mã khối Blowfish, nói chung tốc độ truyền dữ liệu nhanh hơn khi dùng kỹ thuật mã khối IDEA.

Một trong các lý do để thời gian truyền dữ liệu không tăng đáng kể khi cài đặt bộ phần mềm IPSEC_SUN là do các máy được dùng để khảo sát có cấu hình mạnh. Với các máy với cấu hình thấp, việc càI đặt IPSEC_SUN sẽ ảnh hưởng đáng kế đến thời gian truyền dữ liệu. Với mô hình mạng như trong hình 4.11 nhưng các máy có cấu hình thấp hơn như sau:

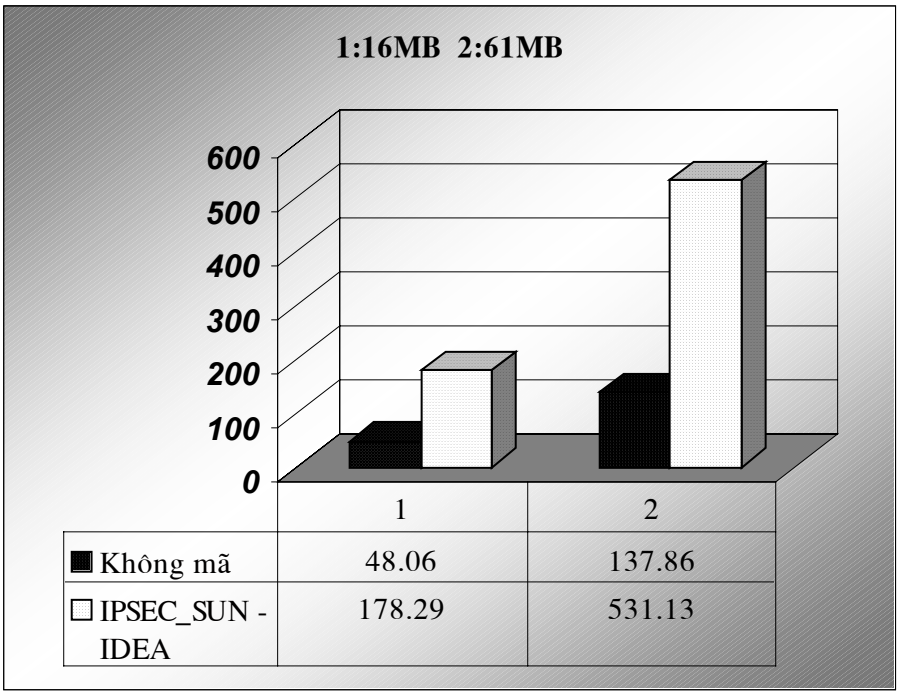
Gateway1 : Compaq Celeron 400MHZ, 32 MB RAM, 4.3 GB HDD.

Gateway2 :Compaq Celeron 400MHZ, 32 MB RAM, 4.3 GB HDD.

Client1 : Celeron 400MHZ, 32 MB RAM, 4,3 GB HDD.

Client 2 : IBM Pentium II 133 MHZ, 32 MB RAM, , 2,1 GB HDD.

Chúng tôi đã truyền hai file có độ lớn 16 MB và 61 MB từ Client 1 chạy Windows 95 đến máy Client 2 chạy Windows NT 4.0 và đã thu được kết quả trong biểu đồ sau:



Hình 4.14: Biểu đồ về thời gian truyền dữ liệu của hệ thống càI đặt IPSEC_SUN (Với các máy có cấu hình thấp)

Qua biểu đồ trên chúng ta thấy rằng khi cấu hình máy thấp, việc càI đặt IPSEC_SUN gây ảnh hưởng đáng kể đến thời gian truyền dữ liệu. Trong trường hợp này thời gian truyền dữ liệu khi chặn bắt và mã hoá gói IP sẽ tăng khoảng 2,8 lần. Chúng ta cũng lưu ý rằng ngay cả khi không chạy IPSEC_SUN thời gian truyền dữ liệu trong trường hợp này đã gấp đôI thời gian truyền dữ liệu khi không mã trong sơ đồ.

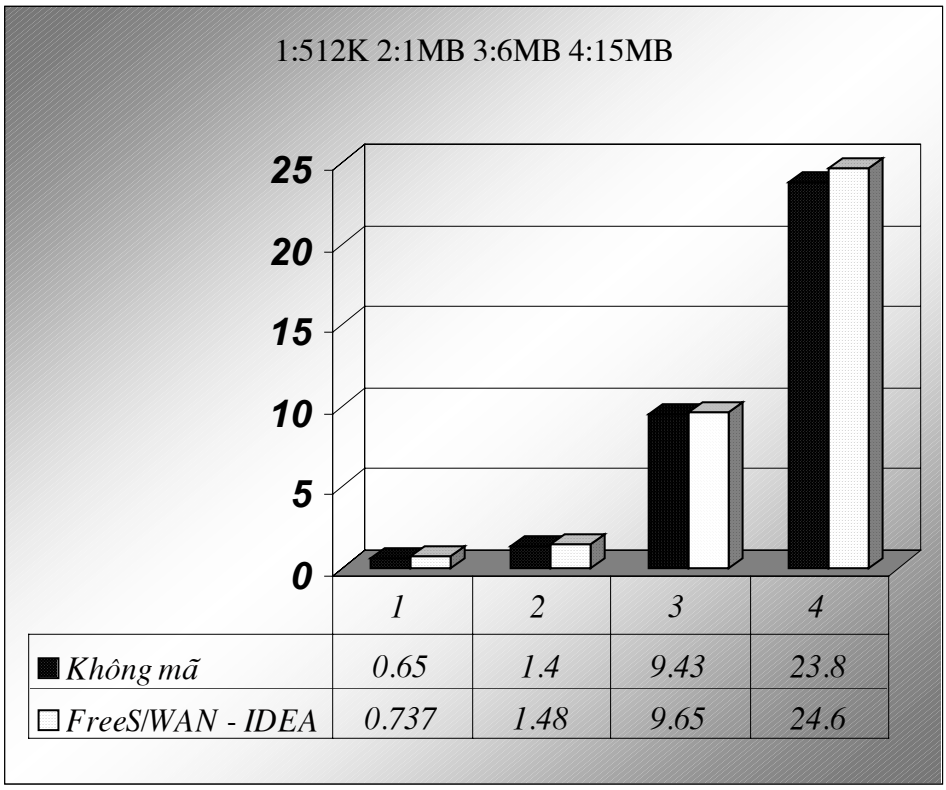
4.2.2 So sánh thời gian truyền dữ liệu giữa hai hệ thống dùng IPSEC_SUN và FreeS/WAN

Do các phiên bản hiện tại của FreeS/WAN không càI đặt kỹ thuật mã khối IDEA, nên để so sánh sự ảnh hưởng của hai phần mềm IPSEC_SUN và FreeS/WAN đến tốc độ truyền dữ liệu của mạng, chúng tôI đã càI đặt bổ xung mô đun mã khối IDEA vào bộ phần mềm FreeS/WAN. Với mô hình mạng như trong hình 4.11, chúng tôI đã khảo sát thời gian và tốc độ truyền dữ liệu trong hai trường hợp không mã và mã dùng FreeS/WAN với kỹ thuật mã khối IDEA.

Kích thước	Thời gian truyền (s)		Tỷ lệ % của tốc độ
	Không mã	FreeS/WAN (IDEA)	
512K ***Tốc độ***	0.65 **788**	0.737 **695**	**88%**
1 MB ***Tốc độ***	1.48 ***692***	1.48 ***692***	***100%***
6 MB ***Tốc độ***	9.43 **652**	9.65 **637**	**98%**
15 MB ***Tốc độ***	23.8 **645**	24.6 **624**	**96%**
Tỷ lệ % trung bình của tốc độ			**95.5 %**

Bảng trên chỉ ra thời gian (s) và tốc độ (Kbytes/s) truyền các file dữ liệu có kích thước 512KB, 1MB, 6 MB và 15 MB từ client 1 đến client 2 trong hai trường hợp chạy và không chạy FreeS/WAN với mã khối IDEA.

Hình 4.15 là biểu đồ về thời gian truyền dữ liệu của hệ thống càI đặt FreeS/WAN với mã khối IDEA



Hình 4.15: Biểu đồ về thời gian truyền dữ liệu của hệ thống dùng FreeS/WAN với mã khối IDEA

Từ các số liệu trên, chúng ta nhận xét rằng, cũng như IPSEC_SUN, với cấu hình máy mạnh như trong hình 4.11, phần mềm FreeS/WAN không ảnh hưởng đáng kể đến tốc độ truyền dữ liệu. Với các máy có cấu hình thấp, thời gian truyền dữ liệu khi chạy FreeS/WAN cũng sẽ tăng khoảng 2,85 lần so với thời gian truyền dữ liệu khi không can thiệp mật mã.

Bảng dưới đây tổng hợp các số liệu về tốc độ và thời gian truyền các file dữ liệu của các hệ thống mạng khi không chạy và khi chạy IPSEC_SUN và FreeS/WAN với cùng mã khối IDEA.

Kích thước	**IPSEC_SUN - IDEA**			**FreeS/WAN – IDEA**		
	Mã	**Không mã**	**Tỷ lệ tốc độ**	**Mã**	**Không mã**	**Tỷ lệ tốc độ**
512K	0.646 (793)	0.59 (868)	**91%**	0.737 (695)	0.65 (788)	**88%**
1 MB	1.28 (800)	1.22 (839)	**95%**	1.48 (692)	1.48 (692)	**100%**
6 MB	8.44 (728)	8.28 (742)	**98%**	9.65 (637)	9.43 (652)	**98%**
15 MB	23.4 (656)	22.6 (680)	**96%**	24.6 (624)	23.8 (645)	**96%**
Tỷ lệ tốc độ (trung bình)			**95 %**	**Tỷ lệ tốc độ (trung bình)**		**95.5%**

Chúng ta thấy rằng thời gian truyền dữ liệu của hai hệ thống khi can thiệp mật mã tăng không đáng kể so với thời gian truyền dữ liệu khi không can thiệp mật mã. Tính trung bình, tỷ lệ của tốc độ truyền dữ liệu khi mã và không mã bằng 95% đối với IPSEC_SUN và bằng 95,5 % khi chạy FreeS/WAN.

Từ các số liệu khảo sát, chúng ta thấy rằng thời gian trễ của gói IP do quá trình chặn bắt và mã hoá phụ thuộc vào các yếu tố sau:

Cấu hình máy: Cấu hình máy càng mạnh thì thời gian mã hoá gói IP càng ít.

Thuật toán mã khối: Trong các thuật toán mã khối thông dụng hiện nay là DES, IDEA và Blowfish thì thuật toán Blowfish có tốc độ tính toán nhanh nhất

Các dịch vụan toàn được càI đặt

Về mặt lý thuyết ta thấy rằng, việc mã hoá toàn bộ phân đoạn tầng vận tảI (bao gồm TCP/UDP header và dữ liệu ứng dụng) sẽ nhanh hơn nếu ta chỉ mã dữ liệu ứng dụng.

KẾT LUẬN

Bảo vệ gói IP bằng kỹ thuật mật mã là một hướng nghiên cứu có nhiều triển vọng và có ý nghĩa thực tiễn cao. Giải pháp được giới thiệu trong phần này có thể áp dụng cho các mạng sử dụng các hệ đIều hành thuộc họ UNIX có hỗ trợ cơ chế STREAMS và không cần mã nguồn mở. Thiết kế của hệ thống là mở và cho phép chúng ta dễ dàng thay đổi các môđun mật mã và các giao thức quản lý khoá.

Chúng ta có thể hoàn thiện bộ phần mềm IPSEC_SUN theo mô hình của IPSEC bằng cách chèn header xác thực, phân phối khoá theo từng cặp Gateway hoặc xây dựng Firewall có chức năng mật mã.

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. Đặng Vũ Sơn (2000), *Nghiên cứu bảo vệ dữ liệu ở tầng IP cho các mạng máy tính sử dụng hệ điều hành UNIX*, Báo cáo đề tài cấp bộ - Ban Cơ yếu Chính phủ.

2. Nguyễn Hữu Giao, Vũ Quốc Khánh, Đặng Vũ Sơn (2001), "Về một giao thức phân phối khoá phiên trong bộ phần mềm bảo vệ gói IP trên Solaris", *Giới thiệu các kết quả nghiên cứu của Học viện kỹ thuật mật mã - Năm 1999-2000*, tr. 155-162, Học viện Kỹ thuật Mật mã (KTMM), Ban Cơ yếu chính phủ.

Tiếng Anh

1. Arto Pulkki (1996), *The IP Security Architecture*, Department of Physis Helsinki University of technology.

2. Bill Rieken, Lyle Weiman (1992), *Adventures in Unix Network Applications Programming*, John Wiley & Sons, Inc. , Canada.

3. D.R. Stinson (1995), *Cryptography: Theory and Practice*, CRC Press, Inc., USA.

4. David A.Curry (1992), *UNIX System Security*, Addition-Wesley Publishing company, INC., USA.

5. D. Harkins, D. Carrel, (1998), "The Internet Key Exchange (IKE)", *RFC 2409*.

6. D.W. Davies and W.L. Price (1989), *Security for Computer Networks - Second Edition*, John Wiley & Sons Ltd, USA.

7. D. Harkins, D. Carrel (1998), "The Internet Key Exchange (IKE)", *RFC 2049*.

8. D. Brent Chapman and Elizabeth D.Zwicky (1995), *Building Internet Firewalls*, O'Reilly & Associates, Inc., USA.

9. D. Maughan, M. Schertler, M. Schneider, J. Turner (1998) 'Internet Security Association and Key Management Protocol (ISAKMP)', *RFC2408*.

10. Gary R.Wright, W. Richard Stevens (1995), *TCP/IPIllustrated, Volume 2,* Adddison-wesley publishing company, USA.

11.George Pajari (1992), *Writing unix device drivers*, Addison-Wesley Publishing Company, Inc., Canada.

12. Janice Winsor (1993), *Solaris System Administrator's Guide*, Ziff-Davis Press, USA.

13. John R. Vacca (1996), *Internet Security Secrets*, IDC books Worldwide, Inc. USA.

14. John Hughes (1998), *Implementation and application of virtual private networks*, Trusted information systems, England.

15.Marcus Goncalves (1998), *Firewalls complete*, McGraw-Hill, USA.

16. Randall Atkinson (1995), "Security Architecture for Internet Protocol", *RFC 1825*.

17. Randall Atkinson (1995), "IP Authentication Header ", *RFC 1826*.

18. Sean Boran (2001), "Hardening Solaris - Security installing a firewall bastion host", *http://www.boran.com/security/sp/solaris-hardening2.html*.

19. Sun Microsystems, Inc. (1995), *Streams Programming Guide,* USA.

20. Sun Microsystems, Inc. (1995), *Writing Device Drivers,* USA.

21. *SUN*(2002) *,* "*Solaris Security Guide*"*,*
http://home.attbi.com/~cabernet/paper/solaris.html.

22. William Caelli, Dennis Longley, Michael Shain (1994), *Information Security Handbook*, Macmillan Press Ltd., Great Britain.

23. William Stallings, Ph.D. (1995) *Network and internetwork security - Principles and Practice*, Prentice -Hall, Inc., USA.

24. William Stallings Ph.D. (1999), *Cryprography and Network security*: *Principles and Practice - Second edition*, Prentice -Hall, Inc.,USA.

25. *http://www.freeswan.org/*.