

# B-7세션
# 리눅스 보안(1)

## 리눅스를 이용한 침입차단 시스템 구현

1. 목적
2. 보안이란?
3. 왜 보안이 중요한가?
4. 침입차단 시스템이란?
5. IPCHAINS
6. IPCHAINS FRONTEND-GFCC
7. IPCHAINS WEB INTERFACE-FWT
8. 실제 IP를 가지는 IP 포워딩 머신 구축
9. 결론


◎ 정정화
**jhjung@wyzsoft.com**


◎ 게시판 주소
**http://seminar.klug.or.kr/inform/teacher/index.php3?ses=b_7**



리눅스를 이용한 침입차단 시스템 구현

# 1 목적

패킷 포워딩 및 내부 침입차단시스템 모듈 등의 강력한 네트워크 기능을 자랑하는 리눅스를 네트워크 출입구로 사용하여 중/소 및 대규모의 네트워크를 보안성, 비용 절감성, 관리의 효율성면에서 향상시킨다.

# 2 보안이란?

## ① 보안이란 무엇인가?

보안을 정보화 시대의 관점에서 정의하면 각종 정보 데이터 및 전산 자원을 고의적 또는 실수에 의한 불법적인 노출, 변조, 파괴, 지체로부터 보호하고 또한, 전자적인 형태의 정보를 처리, 저장 및 전송 등의 모든 단계에 걸쳐서 보호하는 것을 말한다.

## ② UNIX 운영체제는 어떤 것인가?

-. UNIX는 처음부터 네트워크 기능을 탑재하고 있다.
-. UNIX는 멀티유저, 멀티태스크 운영체제이다.
-. UNIX는 프로그램 개발 환경을 공유할 수 있다.
-. UNIX는 시스템 프로그램의 단위별 이식이 용이하다.
-. UNIX는 운영체제를 비롯한 유틸리티들의 원시 프로그램이 공개되어 있다.
-. UNIX는 대부분 C 언어와 컴파일러를 이용한다.
-. UNIX는 특별한 보안기술을 포함하지 않고 있다.

## ③ UNIX 운영체제에 대한 보안 대책은?

-. 시스템 설치시 보안에 대한 대비책을 세워야 한다.
-. 시스템 사용자는 항상 자신의 데이터 보호에 만전을 기해야 한다.
-. 시스템 관리자의 역량과 정책에 따라 시스템의 안전도가 결정된다.
-. 시스템 및 응용프로그램 등의 허점이 그대로 노출되며, 수정과 변조가 용이하다.
-. 문제점 발생시 UNIX를 사용하는 모든 시스템에 동일한 영향력을 갖게 된다.

## ④ 인터넷의 좋은점은 무엇인가?

-. 전세계의 다종다양한 수많은 컴퓨터와 통신망 구성을 통해 상호 연동된 네트워크의 네트워크이다.
-. 인터넷의 확장과 기술의 진보는 매우 빠르며, 정확한 인터넷의 규모와 가입 기관의 현황 파악이 어렵다.
-. 인터넷은 UNIX, TCP/IP 등 개방된 네트워크 구조를 갖는다.
-. 인터넷 서비스는 지역별 기술 격차를 안은 채 전세계적으로 광범위하게 확산되고 있으며, 새로운 서비스 도구의 출현도 매우 빠른 속도로 진행되고 있다.
-. 인터넷 이용자의 대부분은 인터넷 정보 검색과 인터넷을 이용한 비지니스, 네트워크 구축 등에 투자와 관심이 집중되고 있다.
-. 인터넷은 시간과 공간을 초월하여 모든 사용자에게 개방되어 있다.

## ⑤ 인터넷이 왜 보안에 취약한가?

-. 인터넷에 연결된 모든 시스템은 전세계의 모든 사용자에게 공개되어 있으며, 항상 모든 이에게 접근 가능한 상태이다.
-. 하나의 보안 사고는 동시에 전세계의 인터넷에 연결된 모든 곳에 그대로 적용될 수 있으며, 확산 속도가 빠르고, 파생되는 영향력은 매우 심각하다.
-. 인터넷 보안 사고에 관한 보고, 대비책 등을 전달하는 특정 비상체제가 없다.
-. 보안 사고 경위와 피해, 그 대책, 방어 기술 등이 공개되지 않는다.
-. 각 기관에 따라 보안 정책, 시스템 운영 및 관리 기술 등의 보유 격차가 크다.
-. 보안에 대한 중요성이 크게 인식되지 않고 있으며, 관심도가 낮다.
-. 보안은 사고가 나지 않으면 아무리 보안 대책을 세워도 필요성을 인식하지 못하며 보안 사고 발생 후에도 시간이 지나면 관심에서 점차 멀어지며, 보안 대책의 효과를 측정하기 힘들다.
-. 보안 사고는 현존 서비스와 기술에서 부터 새로이 나타나는 모든 것에 대하여 외성을 갖고 발생할 수 있다.

## ⑥ 인터넷 보안을 해야하는 이유

-. 정보의 불법 공개 및 파괴 등으로 인한 서비스 및 업무 방해와 손실을 없애고, 정보의 변조 등에 의한 잘못된 의사결정, 기업 이미지 손상, 또는 산업 스파이로 부터의 정보 및 자원 유출에 의한 경쟁 우위 상실 등을 미연에 방지하고, 정보의 질을 높여 이익 창출과 자산 보호 경쟁력 우위 확보 등을 위해서 보안의 필요성은 강조되고 있다.

## ⑦ 보안(정보 보호)은 어떻게 해야 하나?

-. 정보 보호의 궁극적인 목적은 이를 위해 투자되는 비용과 무대책으로 인한 직간접적인 피해 수준간의 적절한 균형을 유지하며, 최선의 보안 대책을 수립하는 것이다.
-. 보안 정책이 요구되며 그에 따른 가이드라인(guide-line)이 필요하다.
-. 사용자의 보안 교육 및 보안전문가의 육성이 필요하다.
-. 관리 및 법령상의 문제점과 대비책이 필요하다.

## ⑧ 보안의 목표는?

-. 정보의 비밀성 : 비인가자, 불법사용자로부터 비밀 자료의 누출 방지를 보장
-. 정보의 무결성 : 자료의 변경, 삭제, 생성, 파괴 등으로부터 보호하여 원상태 유지
-. 정보의 가용성 : 적시적소에서 정보에 접근 가능하고 사용할 수 있는 상태 확보
-. 서비스 기록성 : 서비스에 관한 Log의 기록 및 감시
-. 부인 봉쇄 : 부인할 수 없는 증거의 확보

## ⑨ 인터넷에서의 보안사고 사례(해외 사례)

-. 서독 스파이 해킹 사건
1987년 서독의 해커들이 서방국가 기밀들을 빼내 KGB로 넘긴 사건
-. 인터넷 웜 사건
미국 코넬대학의 대학원생이 스스로 복사되어 돌아다니도록 만든 바이러스 프로그램으로 전세계 약 7500대의 컴퓨터가 감염되어 정지한 사건
-. 죤 메카트닉 사건
2~3년간 타인을 위조하여 정보를 팔거나 돈을 빼낸 사건(1995년 FBI에 체포)
-. 시티은행 침입 사건
러시아 해커가 시티은행에 침입하여 수백만 달러를 불법인출한 사건

## ⑤ 인터넷이 왜 보안에 취약한가?

-. 인터넷에 연결된 모든 시스템은 전세계의 모든 사용자에게 공개되어 있으며, 항상 모든 이에게 접근 가능한 상태이다.
-. 하나의 보안 사고는 동시에 전세계의 인터넷에 연결된 모든 곳에 그대로 적용될 수 있으며, 확산 속도가 빠르고, 파생되는 영향력은 매우 심각하다.
-. 인터넷 보안 사고에 관한 보고, 대비책 등을 전달하는 특정 비상체제가 없다.
-. 보안 사고 경위와 피해, 그 대책, 방어 기술 등이 공개되지 않는다.
-. 각 기관에 따라 보안 정책, 시스템 운영 및 관리 기술 등의 보유 격차가 크다.
-. 보안에 대한 중요성이 크게 인식되지 않고 있으며, 관심도가 낮다.
-. 보안은 사고가 나지 않으면 아무리 보안 대책을 세워도 필요성을 인식하지 못하며 보안 사고 발생 후에도 시간이 지나면 관심에서 점차 멀어지며, 보안 대책의 효과를 측정하기 힘들다.
-. 보안 사고는 현존 서비스와 기술에서 부터 새로이 나타나는 모든 것에 대하여 외성을 갖고 발생할 수 있다.

## ⑥ 인터넷 보안을 해야하는 이유

-. 정보의 불법 공개 및 파괴 등으로 인한 서비스 및 업무 방해와 손실을 없애고, 정보의 변조 등에 의한 잘못된 의사결정, 기업 이미지 손상, 또는 산업 스파이로 부터의 정보 및 자원 유출에 의한 경쟁 우위 상실 등을 미연에 방지하고, 정보의 질을 높여 이익 창출과 자산 보호 경쟁력 우위 확보 등을 위해서 보안의 필요성은 강조되고 있다.

## ⑦ 보안(정보 보호)은 어떻게 해야 하나?

-. 정보 보호의 궁극적인 목적은 이를 위해 투자되는 비용과 무대책으로 인한 직간접적인 피해 수준간의 적절한 균형을 유지하며, 최선의 보안 대책을 수립하는 것이다.
-. 보안 정책이 요구되며 그에 따른 가이드라인(guide-line)이 필요하다.
-. 사용자의 보안 교육 및 보안전문가의 육성이 필요하다.
-. 관리 및 법령상의 문제점과 대비책이 필요하다.

## ⑧ 보안의 목표는?

-. 정보의 비밀성 : 비인가자, 불법사용자로부터 비밀 자료의 누출 방지를 보장
-. 정보의 무결성 : 자료의 변경, 삭제, 생성, 파괴 등으로부터 보호하여 원상태 유지
-. 정보의 가용성 : 적시적소에서 정보에 접근 가능하고 사용할 수 있는 상태 확보
-. 서비스 기록성 : 서비스에 관한 Log의 기록 및 감시
-. 부인 봉쇄 : 부인할 수 없는 증거의 확보

## ⑨ 인터넷에서의 보안사고 사례(해외 사례)

-. 서독 스파이 해킹 사건
1987년 서독의 해커들이 서방국가 기밀들을 빼내 KGB로 넘긴 사건
-. 인터넷 웜 사건
미국 코넬대학의 대학원생이 스스로 복사되어 돌아다니도록 만든 바이러스 프로그램으로 전세계 약 7500대의 컴퓨터가 감염되어 정지한 사건
-. 죤 메카트닉 사건
2~3년간 타인을 위조하여 정보를 팔거나 돈을 빼낸 사건(1995년 FBI에 체포)
-. 시티은행 침입 사건
러시아 해커가 시티은행에 침입하여 수백만 달러를 불법인출한 사건

-. 특별한 동기 없이 현실 체제에 대한 고의적 파괴 행위
-. 가치있는 자료를 불법적으로 획득하고자 하는 행위
-. 고의적인 태업을 유도하거나, 보복적인 행위
-. 몰래 염탐하거나, 중요한 사실을 얻어내는 간첩 행위

## ⑭ 해커의 침해 사안을 살펴보면?

-. 특정 프로세스(process)를 계속 가동시켜 시스템을 마비시킨다.
-. 시스템내 여유공간(free space)과 파일 시스템 인지정보(inode)를 소멸시킨다.
-. 시스템의 시각, 일자 등 운영체제의 기본 정보를 바꾼다.
-. 시스템 지원을 위한 파일이 수행될 때 혼란을 야기시킨다.
-. 특정한 파일, 디렉토리를 파기하거나, 내용을 변조한다.
-. 이용자의 단말기, 장비 등의 사용을 강제로 중지시킨다.
-. 이용자의 파일 시스템, 단말기 등의 이용권한을 바꾼다.
-. 이용자들에게 이상한 메일을 보내거나, 메시지를 남긴다.

## ⑮ 인터넷 이용자들이 가장 먼저 지켜야 할 보안 사항은 무엇인가?

패스워드 관리는 이용자가 지켜야할 보안 사항으로, 모든 보안 사고의 시작점이다.
따라서 이용자는 자기 계정의 패스워드에 대한 관리를 철저히 해야 하는데, 다음은 패스워드 관리에 대하여 정리한 것이다.

- 이용자 계정(USER ID)을 그대로 패스워드로 사용하지 말 것
- 이용자 전체 이름을 패스워드로 사용하지 말 것
- 이용자 개인 관련 정보를 패스워드로 사용하지 말 것
- 동일 숫자 또는 문자로 반복한 패스워드를 사용하지 말 것
- 6글자 이하, 또는 영어 단어를 패스워드로 사용하지 말 것
- 빨리 입력(key-in)하기 어려운 패스워드는 사용하지 말 것
- 이전에 사용했던 것과 동일한 패스워드를 사용하지 말 것
- 다른 시스템에서 사용 중인 것과 동일한 패스워드를 사용하지 말 것
- 한글의 특성을 최대로 활용한 패스워드를 사용할 것
- 대소문자, 특수문자를 함께 조합한 패스워드를 사용할 것
- 패스워드에 관한 기록을 남기거나 패스워드 정책을 알리지 말 것
- 다른 사용자에게 잠시라도 빌려주거나, 도용당하지 않도록 할 것
- 주기적으로 패스워드를 변경할 것
- 장기간 사용치 않거나 퇴직시에는 계정 사용 중지를 요청하여 조치할 것

## ⑯ 시스템 관리자의 보안 대책에는 어떤 것이 있는가?

**(ftp://ftp.auscert.org.au/pub/auscert/papers/unix_security_checklist_1.0)**

- 시스템의 알려진 버그를 패치한다.
- 네트워크에 관련된 사항들을 적절히 셋업한다.
  - 데몬들의 필터링(Filtering)으로 Customizing하여 불필요한 서비스를 막는다.
  - 불필요한 "r" 코맨드의 사용 중지(rlogin, rsh 등)

- /etc/hosts.equiv 파일의 점검
- $HOME/.rhosts 파일의 사용금지
- 가능한 NFS 사용을 금지하며, 필요한 경우 라우터에서 NFS 트래픽을 Filtering하고 NFS Port를 모니터링 하며, /etc/exports 파일에서 꼭필요한 시스템만 Access가 가능하도록 한다.
- /etc/hosts.lpd, /etc/ttytab, /etc/inetd.conf, /etc/services 등의 점검
- tftp 서비스가 필요없다면 막는다.
- /etc/aliases, /etc/sendmail.cf 등의 점검
- majordomo, fingerd, UUCP 등의 점검
- tcp_wrapper 설치

- ftpd 및 Anonymous ftp 설정의 점검
- 패스워드 및 어카운트 정책의 수립
  - Npasswd, Passwd+, Shadow 설치
- 파일 시스템의 보안 강화
  - 파일들의 Permissions, Modes, Ownership 등을 점검하고 root에 의해 수행되는 파일 및 rc 파일 등을 점검
  - /etc/rc.local, /usr/lib/expreserve, system/devices, /etc/rc* 등의 점검
  - Tiger, COPS 설치
- X-Windows를 사용하면 그에 따른 보안 대책이 필요하다.

## ⑰ 유용한 보안툴로는 어떤 것이 있는가?

- 패스워드 강화를 위한 툴
  - Npasswd, Passwd+, Shadow, Chacct
- 패스워드 검출을 위한 툴
  - Crack, Sniffer
- Sniffer 가동 여부 검출 툴
  - CPM
- 파일 시스템 변경 및 Checksum 관리용 툴
  - MD5, MD5 check, Tripwire, COPS
- 시스템 및 네트워크 보안상태 분석 및 보안 툴
  - Tcp_wrapper : 네트워크 접근 콘트롤(Network Access Control)
  - SATAN : 보안 상태 분석
  - Courtney, Gabriel : SATAN의 가동 시스템 검출
  - NETMAN : 네트워크 모니터링
- 기타 보안에 유용한 툴들
  - Swatch, Simple watch, Triger, PORTUS, Sudo
  - NID & NetMap, SPI, TAP, NOCOL, TAMU, ISS
- 이상에 열거한 보안 관련 툴들은 사용하는 사람과 방법에 따라 보안을 위한 것일 수도 있으며, 반대로 해킹 수단으로 이용될 수 있는 것도 상당수 존재하므로 구체적인 사항은 언급하지 않았다. 자세한 내용을 알고 싶으면 필자에게 연락 바란다.

  단, 위에서 언급한 내용들은 모두 공개 소프트웨어이므로 누구나 입수할 수 있으며, 그것을 이용하여 새로운 해킹 방법이 파생되므로 완벽한 보안에는 한계가 있다.
- 해커는 SATAN, Crack, Sniffer 등등 무수히 많은 방법을 이용하여 언제든지 당신의 컴퓨터에 침입하여 귀중한 자료를 열람 또는 변조 파괴할 수도 있다.

- 각각의 시스템 관리자는 패스워드 강화를 위한 툴로써 Npasswd, Passwd+를 설치하여 사용토록 하므로써 사용자들의 보안의식을 고취시키고, COPS, SATAN 등에 의한 보안상태 분석을 통해 보안 취약점을 보완하여야 하고, Swatch, CPM, Gabriel, NETMAN 등을 이용하여 감시 및 모니터링 할 수 있으며, 기타 유용한 툴들을 적용하여 보안을 강화해야 한다.

## ⑱ 만일 당신의 시스템이 침입자에 의해 해킹을 당했다면?

- 시스템 관리자가 모니터링하거나 시스템 파일의 변경 상태, 또는 타지역으로 부터의 통보에 의해 침입자를 발견했을 경우, 우선 침착하게 침해당한 상황을 판단하고 증거물 획득, log 파일 점검 등을 통해 침입자를 포착하고 그 내용을 문서화 한다.
- 비정상적인 활동이나 현상을 파악하기
  - 한 사용자가 둘 이상 불필요하게 많이 로그인하고 있다.
  - 일반 사용자가 컴파일러, 디버거를 사용한다.
  - 네트워크에 로드를 걸고 이상한 프로그램을 실행한다.
  - 한 사용자가 많은 외부 접속을 시도하고 있다.
  - 일반 모뎀 사용자가 아닌데 모뎀으로 로그인하고 있다.
  - 관리자가 아닌데 관리자로서 명령어를 사용한다.
  - 휴가이거나 근무중이 아닌데 사용한다.
- 침입자가 있다면?
  - 침입자를 추적한다.
    (who, w, last, lastcomm, netstat, snmpnetstat, router information, syslog, /var/adm/messages, wrapper logs, finger, history files 등의 점검 및 이용)
  - 외부로 부터의 Access를 차단
  - 변경된 파일이 있는가 조사
  - 백도어 프로그램의 조사
  - 패스워드 변경 및 강화, NFS, Anonymous FTP, tftp, .forward, .rhosts, 파일들의 변경 시간 등을 조사
  - 기타 시스템 및 네트워크 보안의 취약점 조사 및 보완
- 시스템에 손상을 입었다면 가능한 빨리 복구해야 하며, 관리자를 통해 필요한 조치를 취한다.

## ⑲ 침입차단시스템(방화벽, 방호막)의 설치는 반드시 필요한가?

- 인터넷에 접속한 호스트는 93년 1백만대 수준에서 95년에는 6백40만대로 급격히 증가했으며, 현재 인터넷 사용자 분포는 정부 7%, 군대 6%, 교육 37%, 기업체 45%로 나타나 전세계의 어느 기관을 막론하고 인터넷을 사용하고 있음을 알 수 있다.
- 인터넷 호스트의 급격한 증가에 따라 국제적인 해킹사건이 빈번하게 발생하고 있으며, 해킹사건의 95%가 감지조차 되지 못하는 실정이다.
- 인터넷 방화벽 시스템은 어떤 한 도메인의 기관 네트워크 보안을 위해 현재로선 가장 최선의 해결책이다
  a) 우수한 방법론에 의해 만들어진 프로그램이라 할지라도 결국 사람에 의해 만들어졌으므로 버그가 있을 수 있으며, 결국 그 버그는 보안 취약점이 된다.
  b) 프로그램을 실행시키지 않으면 보안 문제는 없다.
  c) 모든 컴퓨터는 많은 프로그램을 실행한다.
  d) 만약 보안 문제를 최소화하려면 컴퓨터는 최소한의 프로그램을 실행해야 한다.

따라서 위의 a), b), c)에 동의한다면 d)에 의해 하나의 시스템에 그러한 환경을 갖추고(방화벽 시스템을 구축하고) 외부와 내부와의 트래픽에서 불순한 의도의 트래픽을 걸러준다면 다른 대다수의 시스템은 안전을 보장할 수 있다.

## ⑳ 인터넷 응용 보안

- 전자우편(E-mail)은 전세계의 모든 사용자와 서로 주고받을 수 있어야 하며, 가장 중요하면서도 취약한 부분 중의 하나이다.
- 인터넷상의 모든 데이터는 감청될 수 있으며, 개인의 프라이버시 보호 및 전자우편의 보안을 위해 패스워드화 방법을 사용할 수 있다.
- PGP 공개키 패스워드 방식의 이용으로
  - 파일의 패스워드화 및 저장
  - 전자우편을 특정인만을 위해 송신 가능
  - 전자우편/파일에 전자서명이 가능
  - 문서의 변조 방지 메카니즘을 쓸 수 있다.
  - 키 관리 기능이 있다.
- 단, PGP는 미국의 보안 관련 해외 수출 통제품으로 되어 있으며, 국내에서는 공개된 국제판 PGP를 사용해야 한다.

## ㉑ 유닉스에서 사용하는 보안 프로그램

### 1. npasswd

텍사스 Austin 대학의 Clyde Hoover에 의해 개발된 npasswd는 UNIX passwd를 대체하기 위해 만든 프로그램으로서, 지원기능은,

- 최소한의 패스워드 길이를 조정할 수 있다.
- 사용자로 하여금 대소문자, 숫자, 마침점 등을 강요 가능하다.
- 단순한 패스워드를 체크해 낼 수 있다.
- 호스트 이름, 호스트 정보 등을 체크할 수 있다.
- 로그인 이름, 성명 등을 체크할 수 있다.
- 시스템 사전을 비롯, 여러 사전의 단어를 체크할 수 있다.

### 2. cops

- /dev/kmem과 다른 디바이스 파일의 읽기/쓰기를 체크
- 여러 중요 파일의 잘못된 모드를 체크
- 단순 패스워드를 체크
- passwd 파일의 잘못을 체크
- group 파일의 잘못을 체크
- 모든 사용자의 .cshrc, .profile, .login과 .rhost 파일 체크
- /etc/rc와 cron 파일 체크
- NFS의 외부 마운트를 위한 "root"의 경로 체크
- 주어진 사용자를 체크하는 expert 시스템 기능
- setuid 상태의 변화를 체크

### 3. kerberos

MIT의 Athena 프로젝트에서 개발한 인증(Authentication) 시스템으로서, Kerveros는 사용자의 로그인을 인증한 후, 그 사용자의 신원을 네트워크에 흩어져있는 서버, 호스트에 증명해준다. 이 Kerberos는 rlogin, mail, NFS 등에 다양하게 보안 기능을 제공하고 있지만 이 Kerberos를 설치하려면 많은 구성상의 수정이 불가피하여 어려운 점이 있다. 향후 Berkely 4.4 버전에 함께 이식할 계획이 있다.

### 4. Crack

Unix pw 파일의 pw 해독

## ㉒ 해커와의 전쟁

### 인터넷 벌레 Worm

Cornell 대학의 전산학 대학원생인 로버트 모리스(Rovert Morris, Jr.)는 자기 복제 기능과 전파 기능을 가지고 있는 프로그램을 개발하여 1988년 11월 2일에 Internet을 통하여 퍼트리기 시작하였다.
그는 Worm이라 불리는 이 프로그램을 자신이 다니고 있는 Cornell 대학이 아닌, MIT 대학에 최초로 저장하였는데, 이는 Worm이 MIT로부터 확산되었다고 믿게 하기 위해서였다. 이후 Worm은 모리스가 예상했던 것보다 훨씬 빠른 속도로 자신을 복제하여 다른 컴퓨터를 감염시키기 시작하였다.
결국 Worm에 감염되어 손상을 입거나, 심한 시스템 장애를 일으키는 컴퓨터가 전세계적으로 급증하기 시작하였다. 이런 사태를 알게 된 모리스는 자신이 예상했던 것보다 사태가 심각해지자 Harvard에 있는 친구에게 도움을 요청하게 되었다.

결국 그들은 Worm을 제거하는 방법과 예방하는 방법을 알리는 메시지를 Harvard에서 전세계 Internet 사용자에게 익명으로 발송하였다. 그러나 네트워크의 경로상의 문제로 인하여 이 메시지를 이미 감염되어 피해를 입은 후에 알게 되는 경우가 많이 발생하였다.
결국 대학교와 군 기관, 그리고 의료 연구소들을 포함하는 여러 곳의 컴퓨터들이 Worm의 피해를 입게 되었고, 이들의 피해를 대략적인 비용으로 환산하면 각 컴퓨터당 $200에서 $53,000에 이르는 것으로 추산되었다.

한편, 캘리포니아 버클리 대학(University of California at Berkeley)과 MIT에서는 Worm 프로그램을 분석하기 시작하였는데, 얼마후 이 프로그램이 Unix 운영체계의 전자메일을 수행하는 sendmail 프로그램의 문제점(디버그 모드)과 finger를 처리하는 fingerd 프로그램의 문제점을 이용하여 개발되었음을 알아내었다.( 6.1. Finger 참조) 이어서 이들 두 대학의 연구팀들은 Worm이 계속적으로 확산되어가는 것을 방지하기 위한 잠정적인 대처 방안을 마련하기 시작하였는데, 12시간후 버클리 대학의 연구팀은 바이러스의 확산을 늦추는 방법을 알아내게 되었다.
한편 비슷한 시기에 Perdue 대학에서도 해결책을 알아내어 이를 네트워크를 통하여 사용자들에게 알려주기 시작하였다. 하지만 상당히 많은 지역이 이미 Worm에게 피해를 입어 네트워크 연결이 되지 않았으므로, 이런 해결책이 확산되어가는 데는 시간이 많이 소요되어 피해를 입는 컴퓨터가 줄어들지 않았다.

며칠후, Worm이 제거되기 시작하면서 정상적으로 네트워크가 운영되어가자 사용자들은 서서히 Worm 프로그램의 개발자에게 관심이 집중되기 시작하였다. 얼마후 결국 뉴욕 일간지에 모리스가 worm 프로그램의 개발자로 지목되었고, 몇일후 컴퓨터 관련 법안(The Computer Fraud and Abuse Act : Title 18)에 의거 유죄 판정을 받아, 3년의 집행유예와 400시간의 공공봉사, $10,500의 벌금형을 선고받았다. 이후 12월에 Morris는 상소하였지만 다음해 3월, 그의 상소는 기각되었다.

### 뻐꾸기알(Cuckoo's Egg) 침입

"Stalking the Wily Hacker"라는 기사에서 소개된 적이 있고, The Cuckoo's Egg라는 책에서 소개하고 있는 내용 중에 캘리포니아의 Lawrence Berkely 연구소에 있는 컴퓨터에 침입한 해커를 추적하는 사례가 있다. 당시 연구원Clifford Stoll은 연구실 컴퓨터에 해커가 침입하였음을 알아내고, 이를 추적하기 시작했는데, 불충분하지만 연구실의 account 기록(당시 75%의 시간 오차)으로부터 정보를 분석하기 시작하였다. 결국 캘리포니아, 버지니아, 유럽의 컴퓨터를 거쳐 서독 Hannover에 있는 한 작은 아파트에까지 추적하여 해커를 알아내게 되었다.

이 추적 과정에서 Stoll은 여러 기관의 공무원들과 FBI, CIA 그리고 서독의 도움으로 결국 해커를 밝혀내었다. 이러한 Stoll의 경험으로 부터 네트워크를 통하여 각 기관 간의 정보 공유는 상당한 위험을 내포하고 있음을 알 수 있으며, 결국 모든 사용자들이 안전하게 네트워크에서 공존할 수 있는 방안은 현존하는 보안 문제를 신속하게 인식하는 것으로 부터 출발해야 한다고 할 수 있다.

# 3 왜 보안이 중요한가?

## • 정보전쟁

산업스파이
각국 정보기관 ——〉 해킹, 도청, 변조 ——〉 경쟁력 약화,
해커, 내부 비인가자 파괴, 불법 유출 시도 사업 기회 상실,
이미지 하락

- 무력전쟁 —〉 경제전쟁 —〉 정보전쟁
- 국가경제 이익을 위해 각국이 정보기관을 기술정보 수집에 동원
- 산업스파이, 해킹의 위협 증대

## • 정보의 전자화

- 눈에 보이는 통제(위협 : 복사, 절취..) —〉 시공 제약 초월
- 물리적, 관리적 통제 가능
—〉 전자결재 확산으로 자동화된 통제의 중요성 증대
- 전자메일이 일반우편의 5배(미국 95년)
- 컴퓨터 판매량이 TV 판매량을 추월(미국 95년)

## • 유무선 통신을 통한 global network 환경

내부망 —— 외부망 —〉 내부망 외부망

- 전용망을 폐쇄적으로 운영
- 위험이 그룹 전용망, 국내로 제한
- 인터넷, 인트라넷 등 개방 환경으로 변화
- 위험 발생 세계화

# 4 침입차단 시스템이란?

## ① 침입차단시스템은 무엇을 하는 것인가?

외부로부터 내부망을 보호하기 위한 네트워크 구성 요소 중의 하나로써 외부의 불법침입으로부터 내부의 정보 자산을 보호하고 외부로부터 유해정보 유입을 차단하기 위한 정책과 이를 지원하는 H/W 및 S/W를 총칭한다.

당신의 내부 네트워크를 외부와 연결했을 때, 장점은 당신의 전산화된 정보를 상호 교환하는 것이 매우 편리해진다는 것이고, 단점은 인터넷 해커 또는 산업스파이에 당신의 네트워크 자원이나 조직을 노출시킬 수 있다는 것이다. 그래서 조직은 이익을 확대하고 위험을 줄이기 위하여 사용자의 보안 인식 교육, 네트워크 보안 관리자의 능력 향상, 보안을 강화할 수 있는 네트워크 구조, 적절한 네트워크 보안에 관련된 구성 요소 등을 포함하는 포괄적인 네트워크 보안 계획을 개발해야 한다. 침입차단시스템은 잘 정의된 네트워크 보안 구조의 중요한 구성 요소 중의 하나이다.

## ② "방어"의 방법은 무엇인가?

해킹을 막기 위한 방어 개념에는 개별 방어와 경계선 방어가 있다.

### • 개별 방어

모든 host가 네트워크상에 각각 연결되어 있다.
각각의 host는 개별적으로 방어되어 진다.
개별적으로 보안툴을 적용하여야 하기 때문에 비용이 많이 소요된다.

### • 경계선 방어

네트워크에 연결된 경로를 방어한다.
경계선 안에 있는 host는 상호 신뢰할 수 있다.
경계선에만 보안툴을 적용하면 되기 때문에 비용이 저렴하다.

## ③ 침입차단시스템의 종류에는 어떤 것이 있는가?

### • Packet Filtering 침입차단시스템

Packet Filtering 침입차단시스템은 한 Source Address(Host)와 Port(Service)가 두번째 Destination Address와 Port로의 접근을 허용 또는 거부하기 위하여 Router과 Packet Filtering Rule을 사용한다. 예를 들어, 관리자가 외부 네트워크의 특정 Machine이 내부 네트워크의 특정 Machine에 FTP 사용을 허가하기 위하여 Router Rule을 사용할 수 있다. 그러나 동일한 Machine의 내부 네트워크에 대한 Telnet 사용은 거부할 수 있다. 또 비슷한 예로, 모든 다른 Address가 내부 네트워크 상의 특정 Address로의 FTP 사용이 금지될 때, 한 외부 네트워크의 특정 Address는 내부 네트워크의 그 Address에 대한 FTP 사용이 허가될 수 있다.

#### Packet Filtering 침입차단시스템의 장점

Packet Filtering 침입차단시스템의 장점은 빠르고, 일반적으로 비싸지 않으며, 유연성이 높고, 투명한 점을 들 수 있다. 또한 대부분의 조직이 Router를 갖고 있다.

### Packet Filtering 침입차단시스템의 단점

Packet Filtering 침입차단시스템은 수많은 단점을 갖고 있다.

첫째로, IP Packet Header 안에 포함되어 있는 Source, Destination Address와 Port는 내부 네트워크에 대한 접근을 허용 또는 거부하기 위하여 Router를 이용할 수 있는 유일한 정보이다. 그러나 불행하게도, 당신이 접근 요구를 하는 사람이 누군지 모르게 속이기 위하여 Source, Destination과 Port를 조작할 수 있다. 이것은 아주 중요한 문제이다. 실제로 만약 당신이 어떤 특정인에게 당신의 내부 Host에 있는 S/W를 당신의 Router를 통하여 접근하도록 허용하였다면, 누구나 그 Host에 있는 그 S/W에 접근할 수 있다. 그리고 만약 접근되어지는 그 S/W가 강력한 인증을 하지 않으면, 그것은 곧 Hole이며, 침입자에게 당신의 네트워크에 접근을 허용하는 것이다.

둘째로, 일반적으로 Router는 강력한 Logging 기능을 제공하지 않으며, 당신의 네트워크가 침입당한 것을 인지하기 어렵고, 해커의 공격이 성공했을 경우 복구하는 것이 매우 힘들다.

셋째로, Packet Filtering 침입차단시스템은 일반적으로 강력한 사용자 인증 개념을 제공하지 않는다.

넷째로, Router는 일반적으로 보안성(security)이 아니라, 성능을 위주로 개발되었기 때문에, H/W와 S/W 양쪽 다 개발시에 보안 측면에 대한 고려가 불충분했다는 약점을 안고 있다.

다섯째로, Router Rule이 복잡하며, 매우 어렵다. 비록 고도의 네트워크 전문가들이라도 Router의 Rule base에 Rule을 더하거나 뺄 때 사고로 Router에 Hole을 만들어 놓을 수 있다.

마지막으로, 내부 네트워크에 Router를 통한 접근이 허용되었다면, 그 공격자는 S/W 또는 그 Host의 configuration 안에 있는 약점에 직접 접근할 것이다.

## • Circuit Level 침입차단시스템

Circuit Level 침입차단시스템은 침입차단시스템으로 동작되어지는 한 Machine이 내부 네트워크 상의 client로부터 나오는 연결 요구를 처리하는 것을 의미한다. 그래서 침입차단시스템으로부터의 연결 요구는 remote site임이 분명하다.

### Circuit Level 침입차단시스템의 장점

Circuit Level 침입차단시스템의 장점은 내부와 외부 Machine간의 직접 연결을 막는다는 것이다. 들어오는 모든 요구가 차단되어 진다. 만약 내부 Machine상에서 사용자가 어떤 non-standard Port 상의 Code를 쓴다면, 외부 Host 상의 사용자는 그 Port에 접근하는 방법이 없다. 이것은 보안 담당자에게 들어오는 연결 요구를 제어하는 데 하나의 point를 제공한다.

### Circuit Level 침입차단시스템의 단점

첫째로, 만약 Circuit Level 침입차단시스템이 부적절하게 구성되었다면, 그것은 내부 네트워크 상의 사용자가 non-standard Port 상에서 FTP, TELNET 등의 Service를 통하여 침입차단시스템을 우회하는 것이 가능하다. 그리고 그러한 서비스를 외부 네트워크 상의 사용자들이 사용할 수도 있다.

둘째로, 만약 Socks이 사용된다면, 내부 네트워크 상의 client S/W는 Socks S/W와 필요한"handshake"를 수행하기 위하여 수정되어져야 한다.

셋째로, Circuit Level Gateway는 application Protocol을 자체적으로 번역하지 못한다. 그러므로 application Level에서 Traffic을 감시하거나 제어하지 못한다. 여기서 단지 Address와 Port number만이 이용 가능하다. 그래서, 제어의 정도가 Router보다 좋지 못하다. 그것이 대부분의 조직이 단지 circuit Level Gateway를 통하여 나오는 요구만을 처리하는 이유이다.

### • Application Level 침입차단시스템

Application Level 침입차단시스템은 가장 안전한 침입차단시스템이다. Circuit Level 침입차단시스템과 같이, 침입차단시스템을 통하여 내부 네트워크에 연결을 요구하는 모든 외부 네트워크의 Host Address를 확인할 수 있도록 구성할 수 있다. Application Level 침입차단시스템은 FTP, TELNET과 같은 Service를 위하여, 내부 네트워크 상의 Service에 대해 직접 access를 방해하는 Proxy를 사용한다.

#### Application Level 침입차단시스템의 장점

내부 시스템과 외부 시스템간에 침입차단시스템의 프록시를 통해서만 연결이 허용되고 직접 연결(IP Connection)은 허용되지 않기 때문에 외부에 대한 내부망의 완벽한 경계선 방어 및 내부의 IP 주소를 숨길 수 있다. 따라서, 패킷 필터링 기능의 침입차단시스템보다 보안성이 뛰어나다. 다른 침입차단시스템에 비해서 강력한 Logging 및 Audit 기능을 제공한다. S/Key, Secure ID 등 일회용 패스워드를 이용한 강력한 인증 기능을 제공할 수 있다. 프록시의 특성인 프로토콜 및 데이터 전달 기능을 이용하여 새로운 기능 추가가 용이하다.

#### Application Level 침입차단시스템의 단점

트래픽이 OSI 7계층에서 처리되기 때문에 다른 방식과 비교해서 침입차단시스템의 성능이 떨어지며, 또한 일부 서비스에 대해서는 사용자에게 투명한 서비스를 제공하기 어렵다. 침입차단시스템에서 새로운 서비스를 제공하기 위해서 새로운 프록시 데몬이 있어야 한다. 즉 새로운 서비스에 대한 유연성이 없다.

### • Hybrid 침입차단시스템

앞에서 기술된 방식을 복합적으로 이용한다. Packet Filtering과 Application Gateway 기능이 혼용된 구조로 보안성과 성능면에서 다른 구조에 비해 월등히 뛰어나다.

네트워크 인터페이스 카드를 통하여 TCP/IP 커널로 들어온 패킷은 커널의 패킷 필터링 모듈에 의해 먼저 검사되어 탈락되거나 라우팅되어 응용프로그램으로 전달된다.

## ● 침입차단시스템을 구축하기 위한 형태는 어떤것 들이 있는가?

### Screened Host 침입차단시스템

이 형태에서 외부로부터 사내 네트워크로 직접 들어오는 모든 패킷(Packet)은 스크린 라우터를 거친다. 스크린 라우터는 외부 패킷이 바로 사내 네트워크로 들어오는 것을 차단하고, 모든 패킷을 배스천(Bastion) 호스트로 보낸다. 배스천 호스트는 정해진 셋업에 의해 패킷을 판별하여 통과시키거나 막거나, 모니터링 한다. 이 방법의 특징은 외부에 공개되어 퍼블릭 서비스를 하는 서버가 사내 네트워크 내부에, 즉 침입차단시스템 안에 있다는 특징을 지닌다. 이 방법의 장점은 네트워크 사용자들에 대한 보안시스템의 투명성(Transparency)이 유지된다는 점이다. 즉, 내부적으로 보안 시스템이 작동되고 있지만, 사용자들은 이를 느끼지 못하며 사용상의 불편도 최소화된다. 또한 보안 관리자는 공개된 서버마다 일일이 신경쓰지 않아도 배스천 호스트만 완전하게 운용된다면 보안이 유지될 수 있다는 장점이 있다. 그러나 어떤 이유에서건 공개된 서버가 침입당할 경우 사내 네트워크 전체가 해킹의 사정권에 노출되는 효과를 갖는다는 단점이 있다. 공개 서버가 침입차단시스템 뒤에 위치하고 있는 만큼 뒤에서 구멍이 뚫린다면 침입차단시스템로서는 속수무책이다. 또한 침입차단시스템에 의해 공개 서버의 보안은 강화되지만, 대신 침입차단시스템의 트래픽이 가중되어 네트워크 속도의 저하를 가져올 수 있다.

스크린드 호스트 방식은 패킷 필터링 또는 스크리닝 라우터의 한 포트는 외부망에 연결되어 있고 다른 포트는 내부망에 연결되어 있다. 또한 내부망에 베스천 호스트가 연결된 구조이다. 전체적으로 보면 패킷 필터링 라우터와 베스천호스트 2가지가 복합되어 침입차단시스템 역할을 한다.

## Screend Subnet 침입차단시스템

이 형태가 갖는 특징은 인터넷과 내부 네트워크의 사이에 중립지역이 존재한다는 점이다. 이를 DMZ라고도 부르는데, 기업에서 외부에 공개하여 사용하는 모든 서버들은 이 서브넷에 위치한다. 즉, 배스천 호스트는 물론 일반적으로 공개된 웹서버, 전자메일(E-Mail) 서버 등이 위치하는 곳이다.

DMZ는 외부에서 비교적 자유롭게 접근할 수 있는 영역이며, 여기서 사내 네트워크로 들어가기 위해서는 다시 라우터(Interior Router)를 거쳐야 한다. 이 방법은 신뢰할 수 있는 높은 수준의 보안을 가능케 해주는 장점이 있으나, 사용자 입장에서는 DMZ를 통해 내/외부간 통신이 일어나는 불편이 따른다. 또한 공개 서버의 보안은 각 서버 자체에서 책임을 져야한다. 대신 외부로부터 특별한 경우 외에는 사내 네트워크에 접근할 일이 없어지므로 불법적 침입을 막기가 용이하고, 침입차단시스템의 트래픽이 상대적으로 줄어든다.

## Dual Homed Gateway 침입차단시스템

듀얼-홈드(Dual-Homed) 구조의 침입차단시스템 시스템은 외부 네트워크과 내부 네트워크 사이에 2개의 인터페이스를 가지면서 라우팅 기능이 없는 침입차단시스템을 설치하는 형태이다.

침입차단시스템 시스템은 하나의 네트워크에서 다른 네트워크로 IP 패킷을 라우팅하지 않기 때문에 이 침입차단시스템에 프록시 기능을 부여함으로써 결국 하나의 네트워크에서 발생한 IP 패킷이 직접 다른 네트워크로 전달되지 않도록 한다. 침입차단시스템이 라우팅 기능을 제공하지 않기 때문에 내외부 사용자 모두 침입차단시스템을 통과해야 하기 때문에 보안성을 제공할 수 있을지 모르지만 사용자에게 투명한 서비스를 제공하지 못한다. 보통 이러한 구조에서 보안성을 고려한다면 프록시 기능을 갖는 침입차단시스템을 이용하게 되고, 사용자 투명성을 제공해야 한다면 패킷 필터링 기능을 갖는 침입차단시스템을 이용한다.

## 침입차단시스템의 기능 중 NAT란 무엇인가?

NAT(Network Address Translation)란 인터넷에 연결시킬 수 없는 비공인 사설 IP를 인터넷에서 사용할 수 있는 공인 IP Address로 변환시키는 역할을 하는 시스템을 말한다, 여기서 시스템이란 H/W 혹은 S/W를 말하는데 S/W의 경우 일반적으로 침입차단시스템을 설치함으로써 이 기능을 지원한다

## 침입차단시스템 관리자의 자격 요건은?

우리는 침입차단시스템 관리자가 어떤 자격을 갖고있어야 하는지 빈번하게 요청받는다. 침입차단시스템 관리자는 뛰어난 UNIX system 관리자가 되어야 한다. 이것은 침입차단시스템이 네트워크 환경 하에서 어떻게 기능하는지(DNS configuration, Packet Filtering 등) 이해하는 것이 중요하기 때문이다. 침입차단시스템은 독립적인 구성 요소들의 복잡한 구조 속의 한 구성 요소에 불과하다. 그리고 침입차단시스템 관리자는 침입차단시스템에 가해지는 어떤 변경이 나머지 네트워크에 어떤 영향을 미칠 것인지 이해해야 한다.

그리고 우리는 얼마나 많은 시간을 침입차단시스템 관리에 투자해야 하는지에 대해서 많은 질문을 받는다. 그 시간은 site마다 다르다. 침입차단시스템 자체는 유지 보수를 거의 필요로 하지 않는다. 침입차단시스템은 조직의 보안 정책을 반영한다. 보안정책이 변경되지 않는 한 침입차단시스템의 변경은 필요치 않다. 침입차단시스템 관리자가 투자해야 하는 시간은 주기적으로 log report을 읽고, 침입차단시스템상의 중요한 File의 무결성을 점검하고, 침입차단시스템 log를 backup하고 외부 네트워크로부터 강력한 인증 테이블에 연결을 요구할 때 새로운 사용자를 부가하는 것이다. 통상적으로 수천명의 사용자를 가진 네트워크를 관리하는 데 주일당 하루에서 하루 반 사이의 시간이 필요하다.

# 6 IPCHAINS

Linux IPCHAINS-HOWTO

Paul Russell, Paul.Russell@rustcorp.com.au

v0.6, Sun May 17 14:29:26 CST 1998

번역: 이만용, geoman@nownuri.net

이 문서는 리눅스를 위해 개선된 일반 IP 방화벽 체인 소프트웨어를 어떻게 구하고 설치하고 설정하는지, 그리고 이를 사용할 때 알아두어야 할 사항들에 대하여 자세히 기술하고자 한다.

〈역자의 글〉

리눅스에서 IP 방화벽 코드와 ipfwadm 명령은 매우 흥미롭고 귀중한 자원이다.
IP 패킷 수준에서의 걸러내기 기능의 유용성을 구태여 말할 필요가 없으리라 본다. 특히 매스커레이드 기능은 정말 돋보이는 기능 중 하나이다.
개발 버전 2.1.102 부터 공식적으로 들어와 있는 '개선된' IP 방화벽 코드와 훨씬 더 나아진 관리 도구 ipchains는 리눅스 방화벽 기능을 애호하는 사람들이 꼭 익혀야 할 내용이라 생각한다.

또 하나의 획기적인 작품인 커널 2.2를 기다리며 변화에 적응할 수 있도록 돕기 위해 번역을 하였다.

용어 선택이 적절치 못한 것에 대한 의견을 듣고 계속적으로 수정해 나갈 생각이다.

## 1 소개

### ① 무엇

이 글은 리눅스 IPCHAINS-HOWTO이다. 리눅스 NET-3-HOWTO, IP 매스커레이딩 하우투, PPP 하우투, 이더넷 하우투 등도 읽어 볼 만한 가치가 있는 글이다. (또한 alt.fan.bigfoot FAQ도 괜찮다)

패킷 필터링에 대하여 이미 알고 있다면 '왜?' 섹션과 '어떻게?' 섹션을 읽고 '일반적인 IP 방화벽 체인' 섹션을 읽어 나가기 바란다.

ipfwadm로부터 변환을 하고 있다면 '소개' 섹션, '어떻게?' 섹션, 그리고 별첨 'ipchains와 ipfwadm 간의 차이점', 'ipfwadm-wrapper 스크립트 사용하기'를 읽어나가면 된다.

리눅스 ipchains는 리눅스 IPv4 방화벽 코드(주로 BSD로부터 따온 것이다.)를 다시 만든 것이며 BSD의 ipfw를 다시 만든 ipfwadm을 다시 작성한 것이다.

### ② 왜?

현재 사용 중인 리눅스 방화벽 코드는 패킷 조작을 다루지 못하며 32 비트 카운터(인텔의 경우)를 사용하고 있고, TCP, UDP, ICMP 이외의 프로토콜을 명시할 수 없다. 쉽게 많은 내용에 변화를 가할 수 없고 역규칙을 명시할 수 없으며 몇몇 결함을 가지고 있다. 그리고 관리하기도 힘들다.(때문에 오류를 유발하기 쉽다)

### ③ 어떻게?

현재 이 코드는 커널 2.1.102부터 공식적으로 포함되어 있다.
2.0 커널의 경우에는 웹 페이지로부터 커널 패치를 다운로드 받아야 한다.

### ④ 어디서?

리눅스 일반 IP 방화벽 체인에 대한 공식 웹 페이지는 다음과 같다.
〈http://www.adelaide.net.au/~rustcorp/ipfwchains〉

만약 커다란 익명 FTP 서버를 사용할 수 있다면 FTP에 소프트웨어를 두었겠지만 그렇지 못하기 때문에 HTTP 전송으로 제공하고 있다.

버그 보고, 토론, 개발 그리고 사용에 관한 문제를 다루는 메일링 리스트가 있다. wantree.com.au의 ipchains-request로 subscribe라는 단어가 포함된 메시지를 보내면 가입된다. 리스트에 메일을 보낼 때는 'ipchains-request'가 아니라 'ipchains' 앞으로 메일을 보내야 한다.

## 2 패킷 필터링 기본기

### ① 무엇?

네트워크를 통하는 모든 것은 패킷의 형태를 띤다. 예를 들어 지금 이 패키지(약 50k)를 받는 데는 하나의 크기가 1460바이트인 패킷 약 36개 정도를 발생시킨다.(값은 그냥 아무렇게나 잡았다.)

각 패킷의 앞 부분에는 패킷이 어디로 향하고 있는지, 어디서 왔는지, 어떤 종류의 패킷인지, 그리고 그 밖의 관리상 필요한 세부 사항이 적혀있다.
이 부분을 패킷의 헤더(header)라 부른다. 나머지 부분에는 전송하고자 하는 실제 자료가 들어있으며 몸체(body)라 부른다.

웹, 메일, 원격 로그인에서 사용되는 TCP와 같은 프로토콜은 접속이라는 개념을 사용한다. 보내고자 하는 실제 자료를 포함한 패킷을 보내기에 앞서 다양한 설정 패킷(특별한 헤더를 가지고 있다)이 교환되는데 그 내용은 '접속을 원한다', '승인한다', '고맙다'와 같은 메시지라고 보면 된다. 그리고 나서야 보통의 패킷이 교환된다.

패킷 필터란 지나가는 패킷의 헤더를 보고 패킷의 운명을 결정짓는 소프트웨어이다. 패킷을 무시(deny)할 수 있고(마치 패킷을 안받은 것처럼 버리는 행위) 패킷이 지나가도록 허용하거나 패킷을 거절(reject)할 수 있다.(무시와 같지만 패킷을 보낸 곳에 거절함을 통보해준다.)

리눅스에서는 패킷 필터링 기능이 커널 안에 들어있고 패킷을 처리함에 있어 고려해야 할 몇 가지 사항들이 있지만 헤더를 살펴보고 패킷의 운명을 결정한다는 일반적인 원칙은 똑같다.

## ② 왜?

제어. 보안. 감시.

### • 제어(Control):

내부 네트워크를 다른 네트워크(말하자면 인터넷)에 연결해 주는 용도로 리눅스 박스를 사용하고 있다면 특정 형태의 네트워크 자료를 허용하거나 불허할 수 있다. 예를 들어 패킷 헤더에는 목적지 주소가 들어있기 때문에 특정 외부 네트워크으로 패킷이 나가는 것을 막을 수 있다.
다른 예를 들어보자. 나는 Dilbert 아카이브에 접근하기 위해 네스케이프를 사용한다. 페이지에는 doubleclick.net의 광고들이 있어 넷스케이프가 광고들을 받아대느라 나의 귀중한 시간을 소모하고 있다.
패킷 필터로 하여금 doubleclick.net으로부터 오는 모든 패킷을 막게 하여 문제를 해결하였다.(물론 이 문제에 대해서는 보다 좋은 방법이 있다.)

### • 보안(Security):

리눅스 박스가 무법천지의 인터넷과 깔끔하고 잘 정돈된 여러분만의 네트워크 사이에 놓인 유일한 컴퓨터인 경우, 여러분에게 방문할 수 있는 권한을 제한하는 방법을 익혀두는 것이 좋다. 예를 들어 여러분의 네트워크로부터 밖으로 나가는 것은 별 문제가 아니지만 '죽음의 핑' 같은 것이 악의를 품은 외부인으로부터 오는 것에 대하여 걱정하고 있을지 모른다.

또는 계정과 패스워드를 제대로 갖고 있다 하더라도 외부에서는 리눅스 박스로 텔넷하여 들어오는 것을 바라지 않을 수 있다. 여러분은 대부분의 사람들과 마찬가지로 인터넷을 구경하는 방문객이 되고 싶을 뿐, 방문객을 받아야 하는 서버가 되고 싶지는 않다고 생각할지 모른다.
이 때는 간단히 접속에 사용되는 패킷이 들어오는 것만 막아버리면 그만이다.

### • 감시(Watchfulness):

몇몇 잘못 설정된 내부 네트워크 상의 머신들이 외부로 패킷을 유출하는 것도 가능하다. 이 때는 패킷 필터로 하여금 비정상적인 일이 벌어지는 것에 대하여 통보하도록 할 수 있다. 이에 대한 조치를 취할 수도 있고 또는 그냥 궁금해서 해 볼 수도 있다.

## ③ 어떻게?

커널 안에 새로운 일반 IP 방화벽 체인(Generic IP 침입차단시스템 체인) 기능을 갖춰야 한다. 지금 현재 작동 중인 커널이 이 기능을 갖추고 있는지 알아보려면 '/proc/net/ip_fwchains' 파일이 있는지 확인하라. 있다면 이미 커널안에 기능이 포함되어 있는 것이다.

그렇지 않다면 일반 IP 방화벽 체인 기능을 갖도록 커널을 컴파일해야 한다.
커널을 다운로드 받고 위에서 언급한 웹 페이지로부터 가져온 패치를 적용하고 다음과 같은 설정을 해주어야 한다. 커널 컴파일에 대하여 잘 모른다면 커널 하우투 문서를 읽어보기 바란다.

필요한 설정 옵션은 다음과 같다:

```
CONFIG_EXPERIMENTAL=y
CONFIG_침입차단시스템=y
CONFIG_IP_침입차단시스템=y
CONFIG_IP_침입차단시스템_CHAINS=y
```

ipchains라는 도구를 사용하여 커널과 대화하면서 어떠한 패킷을 필터링할 것인지 말한다. 프로그래머가 아닌 한, 그리고 필요 이상으로 호기심을 갖지 않는 사람들이라면 ipchains 도구를 사용하여 패킷 필터링을 제어한다.

## ④ ipchains

이 도구는 구식의 IP 방화벽 코드에 사용되었던 ipfwadm를 대체한다.
패키지에는 ipfwadm-wrapper라는 이름의 쉘 스크립트가 들어있는데 이것을 사용하면 예전의 패킷 필터링 명령을 그대로 사용할 수 있다. ipfwadm(느린데다 매개변수를 제대로 점검하지 않는 결점을 가지고 있다)를 사용하던 시스템으로부터 일단은 빨리 업그레이드한 상태에서 방화벽 기능을 잠시라도 중단하고 싶지 않은 경우에만 사용하라. 이 스크립트를 사용한다면 이 하우투 문서를 더 이상 읽을 필요가 없다. 별첨 'ipchains와 ipfwadm 간의 차이점' 과 별첨 'ipfwadm-wrapper 스크립트 사용하기' 를 읽어보면 ipfwadm에 관한 문제를 조금 더 자세히 알 수 있다.

# 3 일반 IP 방화벽 체인(Generic IP 침입차단시스템 체인)

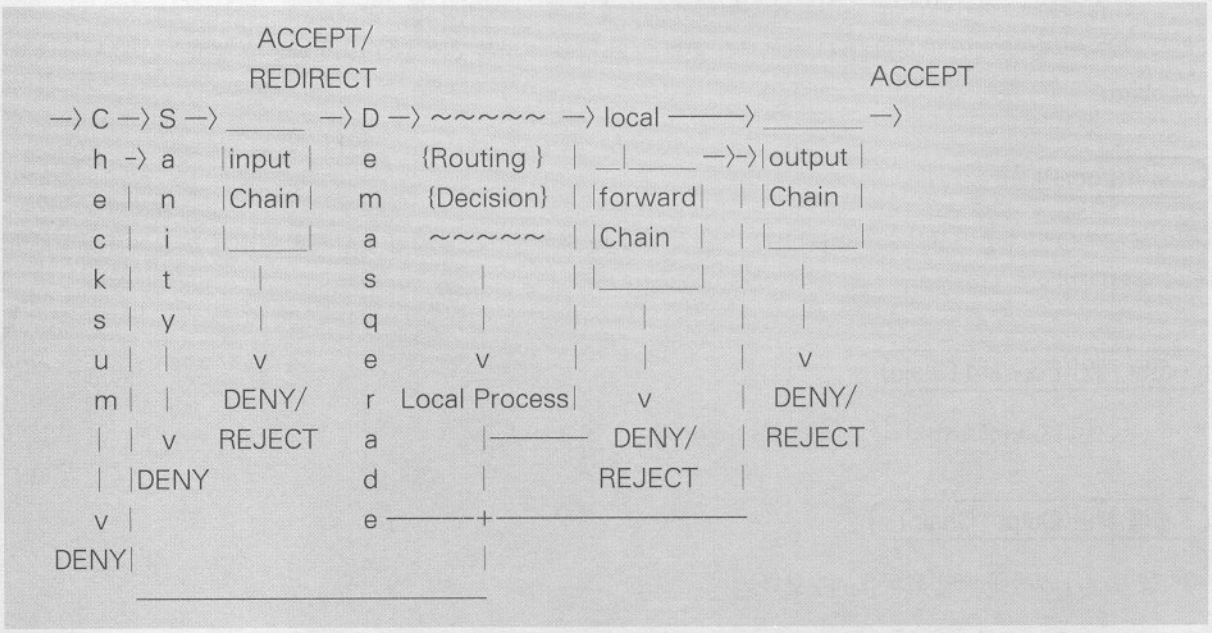
이번 섹션에서는 여러분이 원하는 패킷 필터를 구축하는 데 있어 정말로 알아둬야 할 모든 것을 설명한다.

## ① 어떻게 패킷이 필터를 지나가는가?

커널은 3가지의 규칙 목록(방화벽 체인 또는 간단히 체인이라고 부른다)을 가지고 시작한다. 이 3가지 체인은 입력(input), 출력(output), 그리고 전달(forward)이다. 패킷이 들어오면(예를 들어 이더넷 카드를 통해) 커널은패킷의 운명을 결정짓기 위해 input 체인을 사용한다. 만약 이 단계에서 살아남았다면 이번에는 패킷을 어디로 보내야 할지 결정한다. 만약 패킷이 다른 머신으로 가야 한다면 forward 체인을 살펴본다.
마지막으로 패킷이 떠나려 하는 순간, output 체인을 점검한다.

체인이란 '규칙'의 적용 항목이다. 각 규칙은 '만약 패킷 헤더가 이러이러하다면 패킷을 이렇게 처리하시오'라고 말한다. 만약 패킷이 규칙에 맞지 않으면 체인에 놓인 그 다음 규칙을 적용한다. 마지막으로 더 이상 적용할 규칙이 없으면 체인의 '정책'을 따른다. 보안 문제에 민감한 시스템에서는 거절 또는 무시를 기본 정책으로 삼는다.

ASCII 아트 팬들을 위해 머신으로 들어온 패킷의 완전한 여행 경로를 그려 보았다.




각 단계 하나하나에 대한 설명을 해보자면 다음과 같다:

**• 체크섬(Checksum):**

패킷이 전송 중간에 손상되었는지 여부를 점검한다. 만약 손상되었다면 무시된다.

**• 정상성 테스트(Sanity):**

각각의 방화벽 체인에 앞서 행해지는 정상성 테스트 중 하나이다. 하지만 가장 중요한 것은 입력 체인 전에 이뤄지는 정상성 테스트이다. 몇몇 잘못 만들어진 패킷은 규칙 점검 코드를 혼란에 빠지게 할 가능성이 있으므로 여기서 무시해 버린다.(이런 일이 발생하면 syslog를 통해 메시지가 출력된다)

• 입력 체인(Input Chain):

패킷이 제일 먼저 테스트를 거쳐야 할 방화벽 체인이다. 만약 체인 점검 결과가 DENY 또는 REJECT가 아니라면 패킷은 남은 길을 가야 한다.

• 디매스커레이드(Demasquerade):

만약 패킷이 앞서 매스커레이드된 패킷에 대한 답신 패킷이라면 일단 디매스커레이드하고 출력 체인으로 곧장 진출한다. IP 매스커레이드를 사용하지 않는 사람은 도표에서 디매스커레이드 부분을 지우고 생각해도 좋다.

• 경로 결정(Routing Decision):

라우팅 코드를 거쳐 목적지 필드를 조사하여 패킷이 지역 프로세스(뒤에 나오는 지역 프로세스 섹션을 참고)에게 갈 것인지 아니면 원격 머신에 전달될 것인지 결정한다.(뒤에 나오는 전달 체인 섹션을 참고)

• 지역 프로세스(Local Process):

머신에서 작동 중인 프로세스는 경로 결정 단계 이후 패킷을 받거나 또는 패킷을 보낼 수 있다.(보낸 패킷이 지역 프로세스를 향해 가는 것이라면 log로 설정된 인터페이스의 출력, 입력 체인을 통과한다.
그렇지 않다면 출력 체인만 지나간다) 프로세스 간의 체인 경유 과정은 흥미롭지 않기 때문에 도표에서 충분히 표현하지 않았다.

• 지역(Local):

지역 프로세스에 의해 생성된 것이 아니라면 전달 체인에 대하여 점검하고 지역 프로세스에 의해 만들어진 것이라면 곧장 출력 체인으로 나아간다.

• 전달 체인(Forward Chain):

이 머신에서 다른 머신으로 나아가는 패킷에 대하여 체인을 적용한다.

• 출력 체인(Output Chain):

패킷을 보내기 직전에 체인 규칙을 적용한다.

## ● ipchains 사용하기

우선 이 문서에서 언급하고 있는 ipchains 버전을 가지고 있는지 점검한다:

```
$ ipchains –version
ipchains 1.3.3, 16-May-1998
```

ipchains에는 매우 상세한 설명을 담고 있는 맨 페이지가 들어있다.(man ipchains) 만약 특정 분야에 대한 좀더 자세한 정보를 원한다면 프로그래밍 인터페이스를 살펴보거나(man 4 ipfw) 커널 소스 중에서 net/ipv4/ip_ fwtrees.c 파일을 살펴보라. 이 파일의 내용이 가장 확실한 정보를 제공할 것임에 틀림없다.

ipchains를 통해 할 수 있는 일에는 다음과 같다. 우선 전체 체인을 관리하는 동작이 있다. 우선 삭제할 수 없는 input, output, forward라는 3가지 내장 체인부터 시작해본다.

1. 새로운 체인을 만든다.(-N)
2. 빈 체인을 지운다.(-X)
3. 내장 체인에 대한 기본 정책을 변경한다.(-P)
4. 체인 속에 든 규칙을 나열한다.(-L)
5. 체인으로부터 규칙을 모두 방출한다.(-F)
6. 체인 속의 모든 규칙에 대한 패킷, 바이트 카운터 값을 0으로 설정한다.(-Z)

체인 속의 규칙을 관리하는 방법은 다음과 같다:

1. 체인에 새로운 규칙을 추가한다.(-A)
2. 체인 속 어딘가에 새로운 규칙을 삽입한다.(-I)
3. 체인 속 특정 위치의 규칙을 교체한다.(-R)
4. 체인 속의 특정 규칙을 삭제한다.(-D)
5. 체인 속에서 첫번째 부합되는 규칙을 삭제한다.(-D)

매스커레이딩을 위한 몇 가지 동작이 ipchains에 들어있다.

1. 현재 매스커레이드된 접속 규칙을 나열한다.(-M -L)
2. 매스커레이딩 타임아웃 값을 설정한다.(-M -S)

마지막(그리고 어쩌면 가장 유용한) 기능으로는 여러분으로 하여금 어떤 패킷이 주어진 체인을 통과할 때 어떤 일이 벌어지는지 점검하는 기능이 있다.

## ● 하나의 규칙에 대한 동작

규칙을 자유자재로 관리하는 것, 이것이야말로 ipchains의 핵심이다.
대부분의 경우 여러분은 추가(-A), 삭제(-D) 명령을 사용한다.
다른 것들(삽입을 위한 -I, 교체를 위한 -R)은 기초 개념에 대한 단순한 확장에 불과하다.

각 규칙은 패킷이 만족시켜야 하는 조건들을 나타내고 조건을 충족하면 어떤 행위(target)를 해야 하는지 말한다. 예를 들어 127.0.0.1이라는 IP 주소로부터 오는 모든 ICMP 패킷을 무시하려 한다고 하자. 이 때 프로토콜은 ICMP여야 하고 발신 주소는 127.0.0.1이어야 한다는 것이 조건이다. 목표(target)는 DENY가 된다.

127.0.0.1은 '루프백' 인터페이스로서 실질적인 네트워크 연결이 없어도 항상 가지고 있는 인터페이스이다. ping 프로그램을 사용하여 패킷을 발생시킬 수 있다. (ping은 단순히 ICMP 타입 8(echo request)를 보내며 이에 협조하는 모든 호스트는 ICMP 타입 0(echo reply)를 보내주도록 되어 있다.) 테스트해 보기에 좋은 예이다.

```
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2 ms

— 127.0.0.1 ping statistics —
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
```

```
# ipchains -A input -s 127.0.0.1 -p icmp -j DENY
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes

— 127.0.0.1 ping statistics —
1 packets transmitted, 0 packets received, 100% packet loss
#
```

위의 예에서 첫번째 ping은 성공한다.( '-c 1' 이란 패킷 하나만 보내라는 지시이다.)

그리고 나서 우리는 'input' 체인에 127.0.0.1로부터 오며( '-s 127.0.0.1' ) 프로토콜은 ICMP인( '-p ICMP' ) 패킷에 대하여 DENY로 점프하라고( '-j DENY' ) 규칙을 추가하였다.(-A)

두번째 ping을 가지고 규칙을 점검해본다. 결코 오지 않을 응답을 기다리느라 프로그램이 기다리는 시간이 있기 때문에 잠시 지연된다.

규칙을 지우는 방법은 두 가지이다. 우선 입력 체인에 규칙이 하나 뿐이라는 사실을 알고 있기 때문에 숫자를 지정하여 지울 수 있다.

```
# ipchains -D input 1
#
```

위의 예는 입력 체인의 1번 규칙을 지운다.

두번째 방법은 -A 명령과 똑같이 하되 -A를 -D로 바꾸는 방법이다.
매우 복잡한 규칙의 체인이 있고 삭제해야 할 규칙 번호가 무엇인지 찾고 싶지 않을 때 사용한다. 다음과 같다:

```
# ipchains -D input -s 127.0.0.1 -p icmp -j DENY
#
```

-D 문구는 -A 명령(또는 -I, -R)에서와 완전히 똑같은 옵션을 가져야 한다.
만약 같은 체인에 동일한 규칙이 여러 개 있다면 첫번째 것만 지워진다.

## ● 필터링 규칙 표현(Specification)

앞서 프로토콜을 명시하기 위해 '-p' 를 사용하고 발신 주소를 명시하기 위해 '-s' 를 사용한 것을 보았다. 하지만 이것 이외에도 패킷의 특성을 표현하는 다른 옵션들이 많다. 약간 지루하게 느껴질지도 모르는 요약 설명을 나열하고자 한다.

### 발신, 도착 IP 주소 명시하기

발신(-s), 도착(-d) IP 주소는 4가지 방법으로 표현할 수 있다.
가장 일반적인 방법은 'localhost', 'www.linuxhq.com' 과 같은 완전한 이름을 사용하는 것이다. 두번째 방법은 '127.0.0.1' 과 같이 IP 주소를 적는 것이다.

세번째, 네번째 방법은 '199.95.207.0/24' 또는 '199.95.207.0/255.255.255.0' 와 같이 IP 주소의 그룹을 표현하는 것이다. 이 둘 다 192.95.207.0부터 192.95.207.255까지의 IP 주소를 표현한다.

'/' 다음의 숫자는 IP 주소 중 어느 부분이 중요한가를 나타낸다. '/32' 또는 '/255.255.255.255'이 기본값이다.(IP 주소의 모든 부분이 중요) '/0'을 사용하면 모든 IP 주소를 나타낼 수 있다.

```
# ipchains -A input -s 0/0 -j DENY
#
```

위의 방법은 거의 사용되지 않는다. 왜냐하면 아예 '-s' 옵션을 쓰지 않은 것이나 다름없는 결과이기 때문이다.

## 역규칙(Inversion) 명시하기

'-s', '-d' 플래그를 포함하는 많은 플래그 앞에 '!'를('not'을 뜻한다) 붙이면 주어진 주소와 같지 않는 것을 표현한다. 예를 들어 '-s ! localhost'는 localhost에서 오지 않는 모든 패킷과 일치한다.

## 프로토콜 명시하기

'-p' 플래그를 사용하여 프로토콜을 명시할 수 있다. 프로토콜은 번호(IP에 대한 프로토콜 번호를 알고 있는 경우) 또는 'TCP', 'UDP', 'ICMP'와 같은 이름을 사용할 수 있다. 대소문자는 중요하지 않으므로 'tcp', 'TCP'는 똑같이 처리된다.

역표현을 하기 위해 '-p ! TCP' 처럼 '!'을 붙일 수 있다.

### (1) UDP, TCP 포트 명시하기

TCP, UDP 프로토콜의 경우에는 TCP, UDP 포트 또는 포트의 범위를 추가로 지정할 수 있다.('조각 처리하기' 섹션을 꼭 읽어보기 바람) 범위는 '6000:6010' 처럼 콜론(:) 문자를 사용하여 표현한다. 이 때 '6000:6010'의 의미는 6000부터 시작하여 6010까지 11개의 포트 번호이다.
만약 범위 시작값이 생략되면 0으로 간주한다. 만약 범위 끝값이 생략되면 65535로 간주한다. 따라서 1024 포트 이하로부터 오는 TCP 접속은 '-p TCP -s 0.0.0.0/0 :1024'라고 표현한다. 포트 번호는 예를 들어 'www'와 같이 이름으로 표기할 수도 있다.

포트를 명시함에 있어 '!'를 사용하여 역규칙 표현이 가능하다.
WWW 패킷 이외의 모든 TCP 패킷을 표현하고자 할 때는 다음과 같이 한다.

```
-p TCP -d 0.0.0.0/0 ! www
-p TCP -d ! 192.168.1.1 www
```

위의 명시 방법과

```
-p TCP -d 192.168.1.1 ! www
```

표현은 서로 전혀 다른 것이라는 사실을 구별할 수 있어야 한다.

첫번째 것은 192.168.1.1을 제외한 모든 머신의 WWW 포트를 사용하는 TCP 패킷을 가리키는 반면 두번째 것은 192.168.1.1 머신으로 오는 WWW 포트 이외의 모든 TCP 접속을 나타낸다.

마지막으로 다음 표현은 192.168.1.1이 아니고 WWW 포트가 아님을 나타낸다:

```
-p TCP -d ! 192.168.1.1 ! www
```

### (2) ICMP 타입과 코드 명시하기

ICMP 역시 부가 옵션을 가지고 있다. 하지만 ICMP에 있어 포트란 없기 때문에 그 의미는 전혀 다르다.

'-s' 옵션 다음에 ICMP 이름(이름을 알아보려면 'ipchains -h icmp' 라고 명령한다)을 적거나 또는 숫자로 된 ICMP 유형과 코드로 적을 수 있다. ICMP 유형는 '-s' 옵션 뒤에 나오고 코드는 '-d' 옵션 뒤에 나올 수 있다.

ICMP 이름은 상당히 길다. 따라서 다른 것과 구별할 수 있을 정도로만 적어주면 된다.

가장 흔한 ICMP 패킷 중 일부이다.

| 번호 | 이름 | 필요로 하는 곳 |
|---|---|---|
| 0 | echo-reply | ping |
| 3 | destination-unreachable | 모든 TCP/UDP 자료 교환 |
| 5 | redirect | 라우팅 데몬을 사용하지 않을 때의 라우팅 |
| 8 | echo-request | ping |
| 11 | time-exceeded | traceroute |

지금 현재로서는 ICMP 이름 앞에 '!' 를 붙일 수 없는 상태이다.

절대로 절대로 ICMP 유형 3번 메시지를 막아선 안된다.( 'ICMP 패킷' 참고)

## 인터페이스 명시하기

'-i' 옵션을 사용하여 적용할 인터페이스 이름을 명시할 수 있다.
들어오는 패킷에 대한 인터페이스(즉, input 체인을 통과하는 패킷에 대하여)는 패킷이 들어온 인터페이스를 간주한다. 논리적으로 볼 때 나가는 패킷에 대한 인터페이스(output 체인을 통과하는 패킷에 대하여)는 그들이 실제로 나갈 인터페이스를 가리킨다. forward 체인을 가로지를 패킷에 대한 인터페이스 또한 패킷이 나갈 인터페이스를 뜻한다. 매우 다양한 조합이 가능하다고 본다.

지금 현재 존재하지 않는 인터페이스를 명시해도 아무 문제 없다.
장치가 작동하기 전까지는 그 규칙에 어떠한 경우도 해당되지 않을 것이기 때문이다. 이 특징은 다이얼 업 PPP 연결(일반적으로 'ppp0' )과 같은 경우에 매우 쓸모있다.

특별한 경우로서 인터페이스 이름이 '+' 로 끝나면 그 문자열로 시작하는 모든 인터페이스(현재 존재하든 않든 관계없이)에 적용된다. 예를 들어 '-i ppp+' 옵션을 사용하면 모든 PPP 인터페이스를 가리킨다.

주어진 인터페이스를 제외한 인터페이스를 표현하기 위해 '!' 를 인터페이스 이름 앞에 붙일 수 있다.

## TCP SYN 패킷만 명시하기

때로는 TCP 접속을 한 방향으로만 허용할 필요가 있다. 예를 들어 외부 WWW 서버에 대한 접속을 허용하면서도 지금 서버 상에 작동 중인 서버에는 접속하지 못하도록 했으면 할 때가 있다.

자연스런 접근 방식은 서버로부터 오는 TCP 패킷을 봉쇄하는 것이다.
하지만 불행하게도 TCP 접속은 양방향으로 패킷이 오갈 수 있어야만 한다.

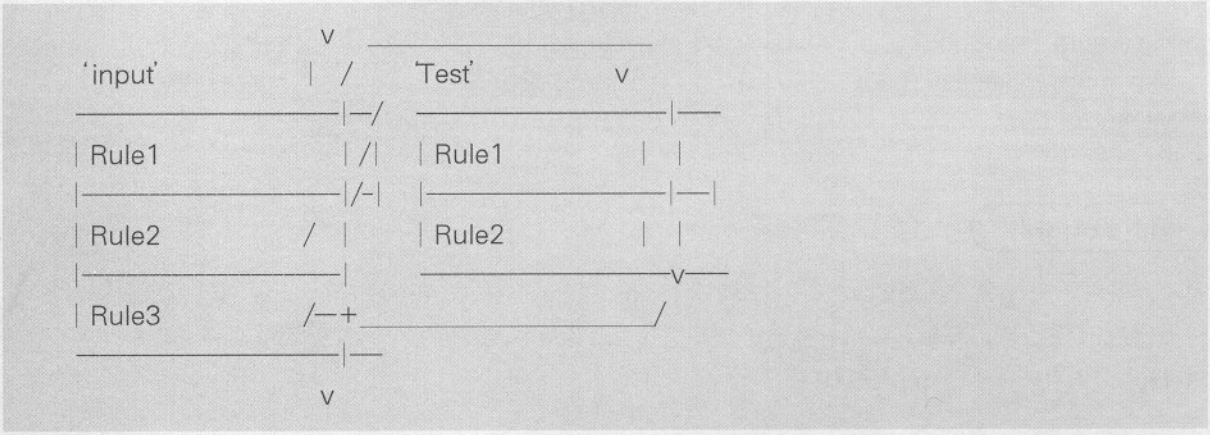
이에 대한 해결책으로서 접속을 요청하는 패킷만 봉쇄하는 방법을 사용한다.
이 패킷을 SYN 패킷이라 부른다.(기술적인 설명으로는 SYN 플래그가 설정되어 있고 FIN, ACK 플래그는 설정되지 않은 패킷을 가리킨다) 이 패킷을 허용하지 않음으로써 접속 요청을 막을 수 있다.

ASCII 아트 시간이 돌아왔다. 다음 2개의 (약간은 멍청한) 체인을 생각해 보자. 하나는 내장 체인인 input이며 하나는 사용자 정의 체인인 Test이다.

```
input                          Test
-------------------------      ----------------------
| Rule1: -p ICMP -j REJECT |   | Rule1: -s 192.168.1.1 |
|--------------------------|   |-----------------------|
| Rule2: -p TCP -j Test    |   | Rule2: -d 192.168.1.1 |
|--------------------------|   ----------------------
| Rule3: -p UDP -j DENY    |
-------------------------
```

192.168.1.1로 부터 와서 1.2.3.4로 가는 TCP 패킷을 생각해본다. 패킷이 입력 체인으로 들어가서 규칙 1번에 적용해본다. 해당되지 않는다.
규칙 2번이 맞고 그 목표는 Test이다. 따라서 조사할 다음 규칙은 Test의 시작 부분이다. Test의 규칙 1은 목표를 명시하고 있지 않으므로 그 다음 규칙 2 를 적용한다. 맞지 않으므로 체인의 끝에 도달하게 된다. 여기서 우리는 마치 규칙 2를 실행하고 나서 규칙 3을 실행하게 된 것처럼 원래의 input 체인으로 돌아간다. 규칙 3 역시 적용되지 않는다.

따라서 패킷의 경로는 다음과 같다:




사용자 체인을 효과적으로 사용하기 싶은 사람은 '여러분만의 방화벽 규칙을 체계화하기' 섹션을 참고하라.

## ● 패킷 기록하기

규칙에 맞을 때 가질 수 있는 부수 효과이다. 일치하는 패킷에 대하여 '-l' 플래그를 가지고 기록할 수 있다. 일상적인 패킷을 기록하기 보다는 예외적인 상황을 발견하고자 할 때 사용하게 될 것이다.
('man klogd' 또는 'man dmesg' 참고)

## ● 서비스 유형 처리하기

IP 헤더에서 자주 사용되지는 않는 4개의 비트를 서비스 유형(Type of Service, TOS) 비트라고 부른다. 이 비트들은 패킷의 처리 방식에 영향을 준다. 4개의 비트는 "최소 지연(Minimum Delay)", "최대 전송률(Maximum Throughput)", "최대 안정성(Maximum Reliability)", "최소 경로(Minimum Cost)"이다. 한 번에 하나만 설정할 수 있다. TOS 처리 코드의 저자인 Rob van Nieuwkerk씨의 말을 들어보겠다.

무엇보다도 내게 있어선 "최소 지연(Minimum Delay)"이 중요하다.
나의 상위 라우터(리눅스이다)로 가는 '대화형' 패킷에 대하여 이 비트를 켜둔다. 나는 336k 모뎀 뒷쪽에 있다. 리눅스는 3개의 큐를 통해 패킷의 우선권을 조정한다. 이런 식으로 동시에 대량의 다운로드 작업을 하면서도 대화형 작업의 효율을 높일 수 있다.(시리얼 장치 드라이버의 큐도 역시 작았더라면 더 향상되었을 수 있겠지만 어찌 되었든 지연 시간을 1.5초로 낮출 수 있었다.)

가장 일반적인 사용 방법은 TELNET & FTP 제어 접속에 대해서는 "최소 지연(Minimum Delay)"을 사용하고 FTP 자료에 대해서는 "최대 전송률(Maximum Throughput)"을 사용하는 것이다. 다음과 같이 하면 된다.

```
ipchains -A output -p tcp -d 0.0.0.0/0 telnet -t 0x01 0x10
ipchains -A output -p tcp -d 0.0.0.0/0 ftp -t 0x01 0x10
ipchains -A output -p tcp -s 0.0.0.0/0 ftp-data -t 0x01 0x08
```

'-t' 플래그는 2개의 전달 변수를 16진수로 받아들인다. 이렇게 함으로써 복잡한 TOS 비트 조작이 가능하다. 첫번째 매스크는 패킷의 현재 TOS 값에 AND 되고 두번째 매스크는 XOR 된다. 너무 복잡하게 여겨진다면 다음 도표를 사용하라.

| TOS 이름 | 값 | 전형적인 용도 |
|---|---|---|
| Minimum Delay | 0x01 0x10 | ftp, telnet |
| Maximum Throughput | 0x01 0x08 | ftp-data |
| Maximum Reliability | 0x01 0x04 | snmp |
| Minimum Cost | 0x01 0x02 | nntp |

## 패킷 표시하기

현재 사용되고 있지 않지만 커널 v2.1 시리즈에서 Traffic Shaper 코드를 Alexey Kuznetsov씨의 새로운 서비스 품질(Quality of Service) 구현으로 바뀌게 되면 더욱 복잡하고 강력한 상호 작용이 가능할 것이라 예상하고 있다.
따라서 2.0 시리즈에서도 역시 무시된다.

## 전체 체인에 대한 동작

ipchains의 유용한 기능 중 하나는 연관된 규칙을 묶어 체인 속에 넣을 수 있다는 것이다. 내장 체인(input, output, forward)과 내장 목표(MASQ, REDIRECT, ACCEPT, DENY, REJECT, RETURN)과 이름 충돌만 없다면 이 체인을 어떻게 부르든 상관없다. 본인이 앞으로의 확장 기능에 대하여 사용할 지 모르므로 대문자 이름을 피하길 권장한다. 체인 이름은 8자까지 가능하다.

## 새로운 체인 만들기

새로운 체인을 만들어보자. 본인으로 말할 것 같으면 매우 상상력이 풍부한(?) 사람이기 때문에 체인 이름을 'test'라 하겠다 (^^)

```
# ipchains -N test
#
```

매우 간단하다. 이제부터 규칙을 추가할 수 있다.

## 체인 지우기

체인 지우기 또한 간단하다.

```
# ipchains -X test
#
```

왜 '-X' 인가? 아~ 이미 좋은 글자들을 다 써버렸기 때문이다.

체인을 지우는데는 몇 가지 제한 사항이 있다. 일단 체인이 빈 상태여야 한다.( '체인 비우기' 참고) 그리고 체인이 다른 규칙의 목표가 되어서는 안된다. 3가지 내장 체인은 지울 수 없다.

## 체인 비우기

'-F' 명령을 사용하면 체인으로부터 모든 규칙을 간단히 비울 수 있다.

```
# ipchains -F forward
#
```

체인을 명시하지 않으면 모든 체인 내용이 비워진다.

## 체인 내용 보기

'-L' 명령을 사용하여 체인 속에 든 모든 규칙을 나열할 수 있다.

```
# ipchains -L input
Chain input (refcnt = 1): (policy ACCEPT)
target     prot opt     source               destination          ports
ACCEPT     icmp ———— anywhere             anywhere             any
# ipchains -L test
Chain test (refcnt = 0):
target     prot opt     source               destination          ports
DENY       icmp ———— localnet/24          anywhere             any
#
```

test 항목에 표시된 'refcnt' 란 test를 목표로 갖고 있는 규칙의 갯수를 나타낸다. 체인이 지워기 전에 이 값이 0이어야 한다.(그리고 체인 자체도 빈 상태여야 한다.)

체인 이름이 생략되면 빈 것이라 할지라도 모든 체인이 표시된다.

'-L' 명령과 사용할 수 있는 3개의 옵션이 있다. '-n' (숫자 표현) 옵션은 ipchains로 하여금 IP 주소 살펴보기를 하지 않게 하기 때문에 DNS가 제대로 설정되지 않아 상당한 지연이 생기거나 DNS 요청을 필터링한 경우에 유용하다. 포트에 대해서는 포트 이름이 아닌 숫자로 표시되도록 한다.

'-v' 옵션을 사용하면 패킷, 바이트 카운터, TOS 매스크, 인터페이스 그리고 패킷 표식과 같은 규칙의 세부 사항을 볼 수 있다. 옵션을 붙이지 않으면 이 값들은 표시되지 않는다. 예를 보자.

```
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
 pkts  bytes  target      prot opt     tosa tosx    ifname      mark          source
   destination             ports
    10          840   ACCEPT                  icmp  ------     0xFF   0x00      lo
anywhere            anywhere            any
```

패킷, 바이트 카운터의 값이 1000, 1,000,000, 1,000,000,000에 대하여 'K', 'M', 'G' 라는 접미사를 사용하여 표시된다는 것을 눈여겨 보자.
'-x' 옵션을 사용하면 그 값이 매우 크다할지라도 완전한 숫자로 표시해준다.

## 카운터 재설정하기(0으로 만들기)

카운터 값을 0으로 만들고 싶을 때가 있다. '-Z' (제로 카운터) 옵션을 사용하면 된다. 예를 보자:

```
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
  pkts bytes target      prot opt     tosa tosx    ifname      mark          source
    destination             ports
    10   840 ACCEPT      icmp ------ 0xFF 0x00  lo                        anywhere
    anywhere            any
# ipchains -Z input
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
  pkts bytes target      prot opt     tosa tosx    ifname      mark          source
    destination             ports
     0     0 ACCEPT      icmp ------ 0xFF 0x00  lo                        anywhere
    anywhere            any
#
```

이런 접근 방식의 문제점은 종종 재설정 바로 직전에 카운터 값을 알아야 할 필요가 있다는 것이다. 위의 예에서 어떤 패킷이 '-L' 과 '-Z' 명령 중간에 지나가는 경우가 발생한다. 이런 이유 때문에 값을 읽으면서 동시에 값을 재설정하기 위해 '-L' 과 '-Z' 를 같이 사용한다. 하지만 이 방법으로는 하나의 체인에 대하여 명령할 수 없으므로 한 번에 모든 체인을 보면서 동시에 0으로 만들어야 한다.

```
# ipchains -L -v -Z
Chain input (policy ACCEPT):
  pkts bytes target      prot opt     tosa tosx    ifname      mark          source
    destination             ports
    10   840 ACCEPT      icmp ------ 0xFF 0x00  lo                        anywhere
    anywhere            any

Chain forward (refcnt = 1): (policy ACCEPT)
Chain output (refcnt = 1): (policy ACCEPT)
Chain test (refcnt = 0):
     0     0 DENY        icmp ------ 0xFF  0x00    ppp0                    localnet/24
   anywhere             any
```

'-y' 플래그를 이런 용도로 사용한다. 오로지 TCP 프로토콜에만 해당한다.
예를 들어 192.168.1.1로부터 오는 TCP 접속을 표현할 때는 다음과 같다.

```
-p TCP -s 192.168.1.1 -y
```

역시 '!'를 붙임으로써 접속 초기화 패킷 이외의 패킷을 표현할 수 있다.

## 조각(Fragments) 처리하기

때때로 하나의 패킷이 한 번에 한 회선을 통과하기에는 너무 큰 경우가 발생한다.
이 때는 패킷이 '조각'으로 나뉘어 여러 개의 패킷으로 전송된다. 받는 쪽에서는 이 조각을 모아 하나의 패킷으로 재구성한다.

조각과 관련된 문제는 다음과 같다. 앞서 나열한 몇 가지 명시 방법(특히 발신지 포트, 목적지 포트, ICMP 유형, ICMP 코드 또는 TCP SYN 플래그)은 커널이 패킷의 시작 부분을 보고 판별하는데, 이 정보가 오로지 첫번째 조각에만 존재한다는 사실로부터 유래한다.

여러분의 머신이 외부 네트워크로 나가는 유일한 연결 창구라면 커널 컴파일시 'IP: always defragment'를 Y로 설정하고 컴파일함으로써 리눅스 머신을 지나가는 모든 조각을 모으도록 할 수 있다. 이렇게 함으로써 문제를 산뜻하게 처리할 수 있다.

그렇지 않을 때는 조각이 필터링 규칙에 의해 어떻게 처리되는지 이해하는 일이 중요하다. 우리가 갖고 있지 못한 정보를 요구하는 어떤 필터링 규칙도 일치하지 않을 것이다. 이는 첫번째 조각이 마치 다른 패킷과 마찬가지로 처리됨을 뜻한다. 하지만 두번째 이후부터는 문제가 발생한다.
따라서 '-p TCP -s 192.168.1.1 www'라는 규칙(발신 포트가 'www'인 것)은 첫번째 조각을 제외하고 나머지 조각에 대하여 일치하지 않는다.
그 반대 규칙인 '-p TCP -s 192.168.1.1 ! www'도 마찬가지다.

그렇지만 '-f' 플래그를 사용하여 두번째 이후의 조각에 대하여 규칙을 명시할 수 있다. 이 조각들에 대하여 TCP, UDP 포트, ICMP 유형, ICMP 코드 또는 TCP SYN 플래그를 명시하는 것은 분명히 적절지 않다.

'!'를 '-f' 앞에 붙임으로써 두번째 이후의 조각이 아닌 것에 대한 규칙을 명시하는 것이 가능하다.

일반적으로 두번째 이후의 조각은 그냥 놔두는 것이 안전하다. 왜냐하면 필터링 규칙이 첫번째 조각에 영향을 줄 것이고 목적지 호스트에서 제대로 조각이 모이지 못하게 할 것이기 때문이다. 하지만 이런 점을 이용하여 조각들을 보내게 되면 머신이 다운되는 버그가 발견된 적 있다. 여러분의 판단에 맡긴다.

네트워크 관리자가 주의할 점 : 잘못 형성된 패킷(TCP, UDP, ICMP 패킷이 포트나 ICMP 코드, 유형을 방화벽 코드가 읽기에는 너무 작은 경우) 또한 조각으로 간주한다. 8번째 위치에 있는 TCP 조각만이 명시적으로 방화벽 코드에 의해 버려진다.(이 때 syslog에 오류 메시지가 나타난다 )

예를 들어 다음 규칙은 192.168.1.1로 가는 모든 조각을 버린다.

```
# ipchains -A output -f -D 192.168.1.1 -j DENY
#
```

## ● 필터링의 부차적인 효과

지금까지 규칙을 사용하여 패킷을 잡아내는 모든 방법을 배웠다.
만약 패킷이 규칙에 부합하면 다음과 같은 일이 일어난다:

1. 규칙에 대한 바이트 카운터가 패킷의 크기(헤더오 모든 자료)만큼 증가한다.
2. 규칙에 대한 패킷 카운터가 증가한다.
3. 규칙에서 요구하는 경우 패킷을 기록한다.
4. 규칙에서 요구하는 경우 패킷의 서비스 유형(Type Of Service) 필드를 변경한다.
5. 규칙에서 요구하는 경우 패킷을 표시한다.(2.0 커널 시리즈에서는 안됨)
6. 규칙 목표를 조사하여 패킷에 대하여 어떤 일을 할 것인지 결정한다.

중요성의 순서대로 이 문제를 언급하도록 하겠다.

## ● 목표(target) 명시하기

'target' 은 커널이 규칙에 맞는 패킷을 어떻게 처리할 것인지 말해주는 것이다.
ipchains는 '-j' (어디어디로 점프한다고 생각하라)을 써서 목표를 명시한다.

가장 간단한 경우는 아무런 목표가 없는 경우이다. 이런 규칙(보통 '회계(accounting) 규칙' 이라 부른다)은 특정 유형의 패킷에 대하여 갯수를 셀 때 사용된다. 규칙에 맞든 틀리든 커널은 체인 속의 다음 규칙으로 향한다.
예를 들어 192.168.1.1로 부터 오는 패킷의 갯수를 세고자 할 때는 다음과 같이 한다.

```
# ipchains -A input -s 192.168.1.1
#
```

( 'ipchains -L -v' 를 사용하여 각 규칙에 연관된 바이트, 패킷 카운터를 볼 수 있다.)

6개의 특별한 목표가 있다. 처음 3가지는 ACCEPT, REJECT, DENY로서 매우 간단하다. ACCEPT는 패킷이 통과하도록 허용한다. DENY는 마치 패킷을 받지 않은 것처럼 버린다. REJECT는 패킷을 버리지만 패킷의 발신지에 ICMP 답신을 보내어 목적지에 도착할 수 없음을 통보한다.(ICMP 패킷이 아닌 경우)

그 다음은 MASQ로서 커널로 하여금 패킷을 매스커레이드하게 한다. 제대로 작동하기 위해서는 IP 매스커레이딩이 가능하도록 커널이 컴파일되어 있어야 한다. 자세한 사항은 Masquerading-HOWTO와 별첨 'ipchains와 ipfwadm 간의 차이점' 을 참고하라. 이 목표는 forward 체인을 통과하는 패킷에만 유효하다.

또다른 중요한 특별 목표로는 커널로 하여금 패킷이 향하고 어디로 향하고 있었든 상관없이 지역 포트로 패킷의 방향으로 변경해버리는 REDIRECT가 있다. TCP, UDP와 같은 프로토콜에서만 가능하다. 선택적으로 포트(이름 또는 번호)를 '-j REDIRECT' 다음에 적음으로써 어떤 특정 포트로 향하고 있던 패킷을 다른 포트로 방향 전환시킬 수 있다. 이 목표는 input 체인을 통과하는 패킷에 유효하다.

특별한 목표의 마지막은 RETURN으로서 즉시 체인의 마지막 항목을 떠나도록 한다.( '정책 정하기' 를 참고)

이외의 다른 목표는 사용자 정의 체인( '전체 체인에 대한 동작' 참고)이다.
패킷은 체인의 규칙을 통과하기 시작한다. 만약 체인에서 패킷의 운명이 결정되지 않았고 체인 통과를 마쳤다면 현재 진행 중이었던 체인의 바로 다음 규칙에서 패킷 통과 작업이 재개된다.

```
# ipchains -L -v
Chain input (policy ACCEPT):
  pkts bytes target          prot opt    tosa tosx    ifname        mark            source
     destination               ports
    10   840 ACCEPT          icmp ---- 0xFF 0x00  lo                                anywhere
     anywhere                  any

Chain forward (refcnt = 1): (policy ACCEPT)
Chain output (refcnt = 1): (policy ACCEPT)
Chain test (refcnt = 0):
       0       0 DENY          icmp ---- 0xFF 0x00  ppp0                          localnet/24
    anywhere                  any
#
```

### 정책 설정하기

앞서 '목표 명시하기'에서 어떻게 패킷이 체인을 통과하는지 이야기할 때 패킷이 내장 체인의 마지막을 통과할 때 어떤 일이 벌어지는지 간략히 알아보았다.
이 때 체인의 정책이 패킷의 운명을 결정한다. 오로지 내장 체인(input, output, forward)만이 정책을 갖는다. 왜냐하면 패킷이 사용자 정의 체인을 통과하고 나면 원래의 체인에서 다시 시작하기 때문이다.

정책은 다음과 같은 특수한 4가지 대상 중 하나를 선택할 수 있다.
ACCEPT, DENY, REJECT, MASQ. 여기서 MASQ는 오로지 forward 체인에만 유효하다.

내장 체인에 포함된 규칙에 RETURN 목표를 두면 패킷이 규칙에 맞을 때 명시적으로 체인의 정책을 따르도록 하는데 쓸모 있다. 이 점도 알아두는 것이 좋다.

## ● 매스러케이딩(Masquerading)에 관련된 동작

IP 매스커레이딩과 관련하여 써먹을 수 있는 몇 가지 매개변수가 있다.
이들을 위한 별도의 도구를 만들 이유가 없다고 생각했기에(이 생각은 앞으로 바뀔 수 있다) ipchains에 그 기능을 포함시켰다.

IP 매스커레이드 명령은 '-M'이다. '-L'과 같이 쓰면 현재 매스커레이드되어 있는 접속 현황을 볼 수 있고 '-S'를 사용하여 매스커레이드 매개변수를 정할 수 있다.

'-L' 명령에 '-n'을 붙이면 호스트 이름과 포트 이름 대신 모두 숫자로 보여주며 '-v' 옵션을 주면 매스커레이드된 접속 상황에 대하여 순차 번호(sequence #)의 델타(delta)값을 보여준다.

'-S' 명령 뒤에는 3개의 타임아웃 값을 초 단위로 적어야 한다.
하나는 TCP 세션, 또 하나는 FIN 패킷 이후의 TCP 세션, 그리고 하나는 UDP 패킷에 대한 값이다. 값을 변경하고 싶지 않을 때는 간단히 '0'을 적는다.

기본값은 '/usr/include/net/ip_masq.h'에 적혀 있으며 현재는 각각 15분, 2분, 5분이다.

일반적으로 변경하는 값은 첫번째 값으로서 FTP를 위해서이다.('FTP 악몽'을 참고)

## ● 패킷 점검하기

때로는 방화벽 체인을 디버깅하기 위해 어떤 패킷이 머신에 들어오면 어떤 일이 발생하는지 알고 싶을 때가 있다. ipchains의 '-C' 명령을 사용하면 된다. 커널이 실제 패킷을 검사할 때 사용하는 동일한 루틴을 사용한다.

'-C' 인수 다음에 패킷을 테스트해 볼 체인 이름을 적는다. 커널은 언제나 input, output, forward 체인에서 시작하지만 테스팅을 목적으로 할 때는 어떤 체인에서든 시작할 수 있다.

패킷에 대한 세부 설명은 방화벽 규칙을 명시할 때 사용했던 문법 그대로 사용한다. 특히 프로토콜('-p'), 발신지 주소('-s'), 목적지 주소('-d'), 인터페이스('-i') 옵션을 꼭 명시해야 한다. 프로토콜이 TCP 또는 UDP 인 경우 발신지 주소 하나와 목적지 포트를 명시해야 하며 ICMP 프로토콜인 경우에는 ICMP 유형과 코드를 꼭 명시해야 한다. (조각 규칙을 가리키기 위해 '-f' 플래그를 사용한 경우가 아닐 때 그러하다. 만약 '-f' 옵션이 사용된 경우에는 발신지 주소, 목적지 포트 옵션은 쓸 수 없다.)

프로토콜이 TCP라면(그리고 '-f' 플래그가 없는 상태) '-y' 플래그를 사용하여 패킷에 SYN 비트가 설정되어 있음을 표현할 수 있다.

다음은 192.168.1.1의 60000포트에서 192.168.1.2의 www 포트로 향하는 TCP SYN 패킷이 input 체인에 들어가서 어떤 결과를 내놓는지 테스트하는 예이다.(전형적인 WWW 접속 시작 과정의 예이다)

```
# ipchains -C input -p tcp -y -s 192.168.1.1 60000 -d 192.168.1.2 www
packet accepted
#
```

## ● 한 번에 여러 규칙 만들기와 사건 감시하기

때로는 하나의 명령으로도 여러 규칙에 영향을 미칠 수 있다. 다음과 같은 두 가지 방법이 있다. 우선 DNS를 통하여 여러 개의 IP 주소로 해석되는 호스트 이름을 명시하게 되면 ipchains는 마치 여러분이 각 IP 주소마다 한 번씩 똑같은 명령을 내린 것처럼 처리한다.

따라서 만약 'www.foo.com' 이라는 호스트 이름이 3개의 IP 주소를 가리키고 'www.bar.com' 호스트 이름이 2개의 IP 주소를 가리키게 된다고 하고 'ipchains -A input -j reject -s www.bar.com -d www.foo.com' 라 명령하게 되면 마치 6개의 규칙을 input 체인에 넣은 것과 같다.

ipchains가 여러 행동을 취하게 하는 다른 방법은 양방향 플래그('-b')를 사용하는 방법이다. 이 플래그를 사용하면 ipchains는 명령을 두 번 내린 것처럼 동작한다. 두번째 명령은 첫번째 것과 '-s', '-d'가 뒤바뀐 명령이다.
따라서 192.168.1.1로 향하거나 그로부터 들어오는 모든 패킷 전달을 막기 위해서는 다음과 같이 할 수 있다.

```
# ipchains -b -A forward -j reject -s 192.168.1.1
#
```

개인적으로는 '-b' 옵션을 좋아하지 않는다. 이게 편하다고 생각하는 사람은 'ipchains-save 사용하기' 섹션을 참고하라.

'-b' 옵션은 삽입('-I'), 삭제('-D')(규칙 번호를 적는 경우 제외), 추가('-A'), 점검('-C') 명령과 함께 사용할 수 있다.

유용한 플래그로는 여러분의 명령에 따라 ipchains가 어떤 일을 하고 있는지 보여주도록 하는 '-v' (장황하게)가 있다. 다수의 규칙에 영향을 미칠 것이라 생각하는 명령에서 쓸모있다.

예를 들어 192.168.1.1과 192.168.1.2 사이의 조각들의 상태를 점검하는 경우를 보겠다.

```
# ipchains -v -b -C input -p tcp -f -s 192.168.1.1 -d 192.168.1.2 -i lo
  tcp opt    ---f- tos 0xFF 0x00  via lo    192.168.1.1  -> 192.168.1.2    * -> *
packet accepted
  tcp opt    ---f- tos 0xFF 0x00  via lo    192.168.1.2  -> 192.168.1.1    * -> *
packet accepted
#
```

## ② 쓸만한 예제

나는 전화접속 PPP 접속( '-i ppp0' )을 하고 있다. 전화접속을 할 때마다 뉴스를 긁어오며( '-p TCP -s news.virtual.net.au nntp' ) 메일을 가져온다.
( '-p TCP -s mail.virtual.net.au pop-3' ) 리눅스 박스를 정기적으로 갱신하기 위해 데비안 FTP 갱신 방법을 사용한다.( '-p TCP -s ftp.debian.org ftp-data' ) 그 동안 ISP의 프록시를 통해 웹 서핑을 즐긴다.
( '-p TCP -d proxy.virtual.net.au 8080' ) 하지만, Dilbert Archive 사이트에 보이는 doubleclick.net으로부터 광고를 싫어한다.
( '-p TCP -y -d 199.95.207.0/24' & '-p TCP -y -s 199.95.208.0/24' )

접속 중에 다른 사람들이 내 머신으로 ftp해 들어오는 것은 게의치 않는다.
( '-p TCP -d $LOCALIP ftp' ) 하지만 내 네트워크 외부의 사람들이 내 IP 주소를 가장하여 사용하길 원치는 않는다.( '-s 192.168.1.0/24' )

내부 네트워크에 다른 머신이 없기 때문에 설정은 매우 쉽게 이뤄진다.

지역 프로세스 어떤 것도(예를 들어 넷스케이프, Gozilla 등등) doubleclick.net에 접속하지 못하도록 하고 싶다.

```
# ipchains -A input -d 199.95.207.0/24 -j REJECT
# ipchains -A input -d 199.95.208.0/24 -j REJECT
#
```

지역적으로 생성되어 input 체인을 통과하는 패킷에 대하여 인터페이스를 'lo' 로 설정하기 위해 '-i lo' 를 명시할 수도 있다.

이제 나는 외부로 나가는 패킷에 대하여 우선권을 조정하려 한다.(들어오는 패킷에 대하여 할 수 있는 일이라곤 별로 없다. ) 적지 않은 규칙을 가지고 있기 때문에 이 모두를 'ppp-out' 이라는 이름의 체인에 넣어두는 것이 좋다고 생각한다.

```
# ipchains -N ppp-out
# ipchains -A output -i ppp0 -j ppp-out
#
```

웹, 텔넷에 대해서는 최소 지연을 원한다.

```
# ipchains -A ppp-out -p TCP -d proxy.virtual.net.au 8080 -t 0x00 0x10
# ipchains -A ppp-out -p TCP -d 0.0.0.0 telnet -t 0x00 0x10
#
```

ftp 자료와 nntp, pop-3에 대해서는 우선권을 낮춘다:

```
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 ftp-data -t 0x00 0x02
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 nntp -t 0x00 0x02
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 pop-3 -t 0x00 0x02
#
```

ppp0 인터페이스를 통해 들어오는 패킷을 제한한 몇 가지 규칙이 있다:
'ppp-in' 이라는 체인을 만들어보자:

```
# ipchains -N ppp-in
# ipchains -A input -i ppp0 -j ppp-in
#
```

ppp0를 통해 192.168.1.*이라는 발신 주소를 갖고 들어오는 패킷은 모두 기록하고 처리한다.

```
# ipchains -A ppp-in -s 192.168.1.0/24 -l -j DENY
#
```

DNS(모든 요청을 203.29.16.1으로 전달하기 때문에 DNS TCP에 대해서는 응답을 허가한다), ftp, 반환하는(return) ftp-data 접속을 허가한다.(return ftp-data란 1023 포트 위로 나가는 ftp-data를 말한다.)

```
# ipchains -A ppp-in -p TCP -s 203.29.16.1 -d $LOCALIP dns -j ACCEPT
# ipchains -A ppp-in -p TCP -s 0.0.0.0/0 ftp-data -d $LOCALIP 1024: -j ACCEPT
# ipchains -A ppp-in -p TCP -d $LOCALIP ftp -j ACCEPT
#
```

마지막으로 그 밖에 지역적으로 생성된 패킷은 모두 허가한다:

```
# ipchains -A input -i lo -j ACCEPT
#
```

input 체인에 대한 기본 정책은 DENY이다. 따라서 명시된 것 이외의 모든 것은 거부한다.

```
# ipchains -P input DENY
#
```

**주의** 실제로 이 순서대로 체인 설정을 하지는 않는다. 왜냐하면 설정 중에 패킷들이 지나갈 수 있기 때문이다. 가장 안전한 방법은 우선 정책을 DENY로 한 후 규칙을 삽입해나가는 것이다. 당연히 규칙에서 DNS 찾아보기를 필요로 하는 경우에는 문제가 된다.

## ipchains-save 사용하기

여러분이 원하는 대로 방화벽 체인을 설정해 놓은 후, 다음 번에 어떻게 했는지 기억하며 끙끙거리는 일은 정말 고통스럽다.

바로 이 문제를 위해 'ipchains-save'라는 스크립트가 현재의 체인 설정을 읽어들여 파일로 저장하는 일을 해준다. 당분간은 ipchains-restore가 어떤 일을 하는지에 대해서만 말해두겠다.

```
$ ipchains-save > my_침입차단시스템
Saving 'input'.
Saving 'output'.
Saving 'forward'.
Saving 'ppp-in'.
Saving 'ppp-out'.
$
```

## ipchains-restore 사용하기

ipchains-save를 사용하여 저장한 체인을 복구하는 스크립트가 ipchains-restore이다. 두 개의 옵션을 가지고 있다. '-v'는 어떤 규칙이 추가되고 있는지 보여준다. '-f' 옵션은 앞서 설명한 대로 체인이 존재하는 경우 사용자 정의 체인의 내용을 일단 비우도록 한다.

만약 스크립트의 입력 내용에 사용자 정의 체인이 있다면 'ipchains-restore'는 체인이 이미 존재하고 있는지 점검한다. 존재하는 경우 일단 체인의 내용을 모두 비울 것인지(규칙을 모두 지우는 일) 아니면 그냥 이 부분은 아무런 일도 하지 않고 넘어갈 것인지 질문을 받는다. '-f' 옵션을 명령행에 주면 질문 과정은 없다. 무조건 체인의 내용이 일단 비워진다.

스크립트를 실행하기 위해서는 root여야 한다. 규칙을 복구시켜놓기 위해 ipchains를 사용하기 때문이다.

예를 보자:

```
# ipchains-restore < my_침입차단시스템
Restoring 'input'.
Restoring 'output'.
Restoring 'forward'.
Restoring 'ppp-in'.
Chain 'ppp-in' already exists. Skip or flush? [S/f]? s
Skipping 'ppp-in'.
Restoring 'ppp-out'.
Chain 'ppp-out' already exists. Skip or flush? [S/f]? f
Flushing 'ppp-out'.
#
```

# 4 기타 등등

이 섹션에서는 문서의 앞 부분 구조에 넣을 수 없었던 기타 정보와 FAQ를 다룬다.

## ① 어떻게 원하는 방화벽 규칙을 체계화할 수 있는가?

이 질문에 앞서 약간의 생각이 필요하다. 여러분은 속도를 최적화할 수도 있고(대부분의 패킷에 대하여 점검 규칙의 수를 최소화하는 쪽) 또는 관리를 편하게 하는 쪽으로 선택할 수 있다.

PPP처럼 간헐적으로 연결되는 회선을 가지고 있는 경우에는 부팅하면서 제일 먼저 input 체인에 '-i ppp0 -j DENY' 를 설정하고 싶을 것이다. 그리고 'ip-up' 스크립트에 다음과 같은 내용을 넣어 사용한다.

```
# Re-create the 'ppp-in' chain.
ipchains-restore -f < ppp-in.침입차단시스템

# Replace DENY rule with jump to ppp-handling chain.
ipchains -R input 1 -i ppp0 -j ppp-in
```

ip-down 스크립트의 내용은 다음과 같다:

```
ipchains -R input 1 -i ppp0 -j DENY
```

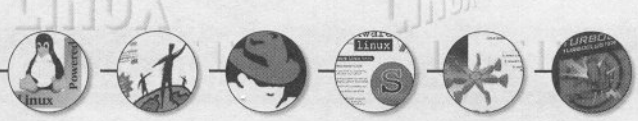
## ① 걸러내서는 안될 것

원하지 않는 모든 것을 걸러내기 시작하기 전에 알아두어야 할 몇 가지 중요한 점들이 있다.

### ● ICMP 패킷

(다른 많은 것들 중에서도) ICMP는 다른 프토토콜(TCP, UDP)의 실패를 통보할때 사용된다. '목적지에 도달할 수 없음(destination-unreachable)' 패킷이ICMP 패킷의 예이다. 이 패킷을 막게 되면 '호스트에 접근할 수 없음(Host unreachable)' 또는 '호스트로의 도달 경로 없음(No route to host)' 오류를 받을 수 없게 된다. 답신이 오지 않을 경우에도 무작정 기다리게 된다. 짜증나게 하는 일이지만 그렇게 심각한 것은 아니다.

더 중요한 문제는 ICMP 패킷이 MTU 찾기에 쓰인다는 사실 때문에 비롯된다.
잘 되어 있는 TCP 구현체(리눅스 포함)들은 패킷이 조각으로 나뉨없이 목적지에 도착할 수 있는 가장 큰 패킷의 크기를 알아내기 위해 MTU 찾기를 사용한다.(조각현상(fragmentation)은 성능을 저하시키기 때문이다. 특히 조각 중 일부를 잃게 되면 성능 저하는 매우 심각해진다.) MTU 찾기는 "조각 나누지 말 것(Don't Fragment)" 비트를 설정한 패킷을 보내고 나서 "조각 나누기가 필요하지만 DF(조각 나누지 않았음) 비트를 설정함(Fragmentation needed but DF set)"이라는 메시지를 받을 때까지 조그마한 패킷을 보냄으로써 이뤄진다.
이 패킷이 바로 '목적지에 도달할 수 없음(destination-unreachable)' 류의 패킷이다. 만약 이 패킷을 받지 못하면 지역 호스트는 MTU 값을 줄일 수 없게 되고 성능은 형편없이 떨어지게 된다.



### ● DNS 답신

모든 TCP 접속을 막고 싶을 때가 있겠지만 이렇게 하면 몇 가지 중요한 것이 작동하지 않을 경우가 있는데 그 첫번째가 바로 DNS이다. 여러분의 리눅스 머신은 호스트 이름을 IP 주소로 바꾸기 위해 DNS를 사용한다.
정상적인 경우 DNS는 UDP를 사용하지만 답신이 클 때는 TCP 답신을 사용한다.
이러한 접속 요구를 막게 되면 DNS 결과를 신뢰할 수 없다.

DNS 요청을 항상 똑같은 외부 머신이 처리하게 하고 있다면(/etc/resolv.conf에서 nameserver 설정을 해주거나 캐슁 네임서버를 전달 모드로 사용하고 있는 경우) 바로 그 머신에 대해서만 domain 포트로의 TCP 접속을 허용하면 된다.

### ● FTP에 관련한 악몽

또 하나의 고전적인 문제가 FTP이다. FTP에는 두 가지 모드가 있다.
전통적인 것을 active mode라고 부르고 근래에 도입된 것을 passive mode라고 부른다. 웹 브라우저는 기본적으로 수동 모드를 사용하고 명령행 ftp 프로그램들은 일반적으로 능동 모드를 사용한다.
능동 모드에서는 원격 머신이 파일을 보내고자 할 때(또는 ls나 dir 명령의 결과) 우선 이쪽 지역 머신으로 TCP 연결을 시도한다. 따라서 여기서 사용하는 TCP 접속을 막게 되면 능동 모드의 FTP가 작동하지 않게 된다.
만약 수동 모드를 선택할 수 있다면 좋다. 수동 모드는 자료 받기의 경우에도 클라이언트로부터 서버로 자료 접속이 이뤄지기 때문이다. 선택권이 없다면 1024 이상의 포트 그리고, 6000과 6010 사이의 포트는 제외한 포트로 TCP 접속이 들어오는 것에 대하여 허가하는 것이 좋다.(6000은 X 윈도우가 사용한다)

## ③ 죽음의 핑(Ping of Death) 걸러내기

리눅스 박스는 현재 그 유명한 죽음의 핑(Ping of Death)에 대하여 면역을 가지고 있다. 이 핑은 비정상적으로 큰 ICMP 패킷을 보내어 TCP 스택 안에서 오버플로우가 일어나게 하고 따라서 수신 측에 해를 끼치도록 되어 있다.

죽음의 핑에 면역 기능을 갖지 않은 리눅스 박스를 보호하기 위해서는 간단히 ICMP 조각을 막아버리면 그만이다. 정상적인 ICMP 패킷은 조각 나누기가 필요할 정도로 크지 않기 때문에 거대한 핑 말고 다른 부분에 영향을 없을 것이다. 본인이 듣기로는(확실한 것은 아니다) 몇몇 시스템의 경우 정상 크기 이상의 ICMP 패킷의 마지막 조각만으로도 시스템이 망쳐질 수 있기 때문에 첫번째 조각만 막는 것은 권하지 않는다.

## ④ 티어드롭(Teardrop)과 봉크(Bonk) 걸러내기

티어드롭과 봉크는 겹치기 조각(overlapping fragment) 방법을 사용하는 두 가지 공격 방법으로서 주로 마이크로소프트 윈도우 NT 머신에 사용된다.
여러분의 리눅스 라우터가 조각모으기를 하게 하거나 이 공격에 무력한 윈도우 NT와 같은 머신에 조각이 가지 못하도록 막으면 된다.

## ⑤ 조각 폭탄(Fragment Bombs) 걸러내기

Some less-reliable TCP stacks are said to have problems dealing with 몇몇 신뢰성 없이 구현되어 있는 TCP 스택은 전체 조각을 받지 못한 상태에서 거대한 양의 조각을 처리할 때 문제를 일으킨다고 한다. 리눅스에는 이러한 문제가 없다. 조각을 그냥 걸러내버리거나(이 경우에는 정상적인 패킷도 걸러내는 효과를 갖는다) 커널을 컴파일할 때 'IP: always defragment' 에서 Y라고 한다.(물론 리눅스 박스가 모든 패킷이 드나드는 유일한 통로일 때만 유효)

## ⑥ 방화벽 규칙 변경하기

방화벽 규칙을 변경하는 데 있어 타이밍에 관련된 문제가 있다. 만약 주의하지 않는다면 규칙을 변경하는 도중 패킷이 지나가도록 놓치는 수가 있다.
이를 막는 가장 간단한 방법은 다음과 같다.

```
# ipchains -I input 1 -j DENY
# ipchains -I output 1 -j DENY
# ipchains -I forward 1 -j DENY
```

... 규칙을 변화시킨다 ...

```
# ipchains -D input 1
# ipchains -D output 1
# ipchains -D forward 1
#
```

변경하는 동안은 모든 패킷을 버리도록 한다.

만약 한 개의 체인에 대해서만 변경 작업을 하고 있다면 새로운 규칙으로 새로운 체인을 만들어 채운 뒤 교체한 예전 체인을 가리키고 있는 규칙이 새롭게 만들 체인을 가리키도록 교체할('-R') 수 있다. 그 다음 예전의 체인을 삭제한다. 이렇게 하여 교체 작업이 한 번에 이뤄지게 할 수 있다.

## ⑦ 보다 나은 계획들

본인이 작성한 코딩한 'libfw'라는 이름의 사용자 영역 라이브러리를 소스 코드에 포함시켜 두었다. IP 체인 1.3의 기능을 사용하여 패킷을 사용자 영역으로 복사해준다.(IP_침입차단시스템_NETLINK 옵션을 사용해야 함)

이 라이브러리를 사용하여 '상태유지 검사(stateful inspection)'와 같은 것을 사용자 영역에서 구현할 수 있다.(본인은 상태 유지 검사보다는 동적 방화벽이라는 용어를 더 선호한다) 또 다르게 응용할 수 있는 좋은 분야는 사용자별로 패킷을 제어하는 것인데 이는 사용자 영역의 데몬 살펴보기를 하면 가능하다. 이 작업은 매우 쉬운 작업이다.

방화벽의 표시(mark) 기능은 많이 사용되고 있지 못한 상태이다.
패킷의 우선권을 간단히 조절할 수 있도록 서비스 품질(Quality of Service) 코드에 대한 순위를 나타내도록 사용할 수 있다.

## ⑧ 앞으로의 보강 계획

앞으로는 모든 방화벽 제어를 /proc/sys/net/ipv4 밑에 두고자 한다.
스크립트를 사용하여 관리하기 쉽고 일반적으로 더 깨끗하게 보이기 때문이다.

사용자 영역으로부터 특정 체인으로 패킷을 다시 주입하는 좋은 방법을 고안하고 있다. 이렇게 되면 libfw 설계가 좀 더 간단해진다.

## 5 별첨 - ipchains과 ipfwadm 간의 차이점

변화된 것 중 일부는 커널 변화의 결과이며 다른 부분은 ipchains와 ipfwadm의 차이로부터 유래한 것이다.

1. 많은 인수들이 다시 재배치되었다:대문자는 명령을 나타내며 소문자는 옵션을 나타내는 데 사용되고 있다.
2. 임의의 체인을 만들 수 있으며 내장 체인이라 할 지라도 플래그 방식이 아니라 이름을 갖고 있다.(예. '-I' 라고 하지 않고 'input' 이라고 부름)
3. '-k' 옵션이 사라졌다. 대신 '! -y' 를 사용하라.
4. '-b' 옵션은 실제로 두 개의 규칙을 삽입하거나 추가하거나 삭제한다. 하나로 된 '양방향' 규칙을 만들지 않는다.
5. 두 가지를 점검하기 위해(양 방향에 대하여 한 번씩) '-C' 에 '-b' 옵션을 사용할 수 있다.
6. '-I' 에 사용하던 '-x' 옵션이 '-v' 로 교체되었다.
7. 여러 개의 발신지 포트는 더 이상 지원되지 않는다. 포트 범위 앞에 부정을 하는 방법으로 대신할 수 있으리라 생각한다.
8. 인터페이스는 오로지 이름으로만 명시할 수 있다.(주소는 안됨) 구식 방법이 2.1 커널 시리즈에서 말없이 변화되어 있다.
9. 조각을 검사하며 자동으로 허용하지는 않는다.
10. 명시적인 회계 체인은 없다.
11. IP 에 대한 모든 프로토콜을 지원한다.
12. SYN, ACK 매칭에 대한 구식 방법(TCP 아닌 패킷에 대해서는 무시함)이 바뀌었다. SYN 옵션은 TCP 관련 규칙이 아닌 곳에서는 사용할 수 없다.
13. 카운터는 x86 에서 32비트가 아니라 64비트이다.
14. 반대 옵션이 지원된다.
15. ICMP 코드가 지원된다.
16. 와일드 카드 인터페이스가 지원된다.
17. TOS 처리에서 정상성 점검이 이뤄진다. 예전 커널 코드에서는 0이어야 함(Must Be Zero) TOS 비트를 (불법적으로) 처리하는 경우 그냥 아무 말 없이 무시했다. ipchains는 기타 다른 불법적인 접근의 경우와 마찬가지로 오류를 발생시킨다.

## ① 간략한 참조 도표

〔주로 명령 인수는 대문자이고 옵션 인수는 소문자로 되어 있다〕

한 가지 잘 알아둬야 할 것으로서 매스커레이딩은 '-j MASQ' 라고 적는다.
'-j ACCEPT' 와는 전혀 다른 것으로서 ipfwadm과 같이 단순한 2차적 기능으로 간주하지 않는다.

| ipfwadm | ipchains | 주의 사항 |
|---|---|---|
| -A (both) | -N acct<br>& -I 1 input -j acct<br>& -I 1 output -j acct<br>& acct | 'acct' 체인을 만들고 들어오고 나가는 패킷이 이 체인을 통과하도록 한다. |
| -A in | input | 목표가 없는 규칙 |
| -A out | output | 목표가 없는 규칙 |
| -F | forward | 〔체인〕으로 사용하라. |
| -I | input | 〔체인〕으로 사용하라. |
| -O | output | 〔체인〕으로 사용하라. |
| -M -I | -M -L | |
| -M -s | -M -S | |
| -a policy | -A (chain) -j POLICY | (-r 와 -m 참고). |
| -d policy | -D (chain) -j POLICY | (-r 와 -m 참고). |
| -i policy | -I 1 (chain) -j POLICY | (-r 와 -m 참고). |
| -l | -L | |
| -z | -Z | |
| -f | -F | |
| -p | -P | |
| -c | -C | |
| -P | -p | |
| -S | -s | 포트 하나 또는 범위만 유효. 다수 지정 불가 |
| -D | -d | 포트 하나 또는 범위만 유효. 다수 지정 불가 |
| -V | 〈none〉 | -i 〔이름〕을 사용할 것. |
| -W | -i | |
| -b | -b | 실제로는 2 개의 규칙 생성 |
| -e | -v | |
| -k | ! -y | -p tcp 를 꼭 같이 사용해야 한다. |
| -m | -j MASQ | |
| -n | -n | |
| -o | -l | |
| -r (redirpt) | -j REDIR (redirpt) | |
| -t | -t | |
| -v | -v | |
| -x | -x | |
| -y | -y | -p tcp 를 꼭 같이 사용해야 한다. |

## ② ipfwadm 명령 번역의 예

```
Old command: ipfwadm -F -p deny
New command: ipchains -P forward DENY
Old command: ipfwadm -F -a m -S 192.168.0.0/24 -D 0.0.0.0/0
New command: ipchains -A forward -j MASQ -s 192.168.0.0/24 -d 0.0.0.0/0
Old command: ipfwadm -I -a accept -V 10.1.2.1 -S 10.0.0.0/8 -D 0.0.0.0/0
New command: ipchains -A input -j ACCEPT -i eth0 -s 10.0.0.0/8 -d 0.0.0.0/0
```

(Note that there is no equivalent for specifying interfaces by
(주의: 주소를 이용하여 인터페이스를 명시하는 방법은 없다. 인터페이스 이름만을 사용하라. 이 머신의 예에서 10.1.2.1은 eth0에 해당한다.)

# 6 별첨 - ipfwadm-wrapper 스크립트 사용하기

ipfwadm-wrapper 쉘 스크립트는 ipfwadm 2.3a와 하위 호환성을 갖추기 위해 만든 ipfwadm에 대한 플러그-인 교체용이다.

다룰 수 없는 유일한 기능은 '-V' 옵션이다. 이 옵션이 사용되면 경고 메시지가 출력될 것이다. '-W' 옵션을 사용하면 '-V' 옵션을 무시한다. '-W' 옵션이 없는 경우에는 ifconfig 명령을 사용하여 주소와 연관되어 있는 인터페이스 이름을 찾으려고 시도한다. 만약 실패한다면(예를 들어 인터페이스가 다운되어 있는 상태인 경우) 오류 메시지를 내보이고 종료한다.

'-V'를 '-W'로 바꾸거나 스크립트의 표준 출력을 /dev/null로 리다이렉트하여 경고 메시지를 피할 수 있다.

스크립트에서 실수한 것을 발견하거나 실제 ipfwadm과 스크립트 간의 차이점이 있다면 버그 보고를 꼭 해주기 바란다 : 메일 주소는 Paul.Russell@rustcorp.com.au이며 주제란에 BUG-REPORT라고 적어 보내주길 바란다. 여러분이 갖고 있는 ipfwadm 구버전( 'ipfwadm -h' )과 사용 중인 ipchains의 버전( 'ipchains —version' ), ipfwadm 랩퍼 스크립트의 버전( 'ipfwadm-wrapper —version' )을 나열하고 또한 'ipchains-save'의 출력도 보내주었으면 한다. 우선 감사의 말을 하고 싶다.

ipchains와 ipfwadm-wrapper 스크립트를 섞어쓰는 것은 좋지 않으나 여러분의 자유이다.

For fixing my leftover debug messages in 2.0.33.

## ● IPCHAINS FRONTEND - GFCC

### ① 소개

gfcc (GTK+ Firewall Control Center) - 구교선님(icarus@autostock.co.kr)
http://icarus.autostock.co.kr

GTK+ 기반의 리눅스 방화벽 설정 프로그램으로 내부적으로는 ipfwc를 사용한다.
GTK+ 기반의 프로그램답게 상당히 깔끔하며 별다른 조작없이 방화벽을 설정할 수 있다.

### ② 설치

http://www.freshmeat.net에서 검색어로 gfcc를 입력하거나 http://icarus.autostock.co.kr에 접속해 gzipped tar ball이나 rpm을 받는다. 여기서는 tarball 설치만 다룬다.

설치를 위해서는 커널 버전 2.1.102 이상 버전(2.2에서 무난하게 작동), gtk 1.2.0 이상, 그리고 libipfwc가 필요하다. 이는 tar ball 자체에 포함되어 있으며 ipchains 패키지에도 역시 들어가 있다.

```
# 파일 확인
[root@bluebird src]# ls
fwt126.tar gfcc-0.7.3.tar.gz

# 압축 및 tar 풀기
[root@bluebird src]# tar -zxvf gfcc-0.7.3.tar.gz
gfcc-0.7.3/
gfcc-0.7.3/Pics/
gfcc-0.7.3/Pics/expand.xpm
gfcc-0.7.3/Pics/apply.xpm
gfcc-0.7.3/Pics/apply-grey.xpm
gfcc-0.7.3/Pics/open.xpm
gfcc-0.7.3/Pics/open-system.xpm
gfcc-0.7.3/Pics/open-grey.xpm
gfcc-0.7.3/Pics/masq.xpm
gfcc-0.7.3/Pics/save.xpm
gfcc-0.7.3/Pics/save-grey.xpm
gfcc-0.7.3/Pics/inverse.xpm
gfcc-0.7.3/Pics/newchain.xpm
gfcc-0.7.3/Pics/export.xpm
gfcc-0.7.3/Pics/export-grey.xpm
gfcc-0.7.3/Pics/normal.xpm
gfcc-0.7.3/README
gfcc-0.7.3/stamp-h.in
gfcc-0.7.3/AUTHORS
gfcc-0.7.3/COPYING
gfcc-0.7.3/ChangeLog
gfcc-0.7.3/INSTALL
gfcc-0.7.3/Makefile.am
gfcc-0.7.3/Makefile.in
gfcc-0.7.3/NEWS
gfcc-0.7.3/TODO
gfcc-0.7.3/aclocal.m4
gfcc-0.7.3/config.guess
gfcc-0.7.3/config.h.in
gfcc-0.7.3/config.sub
gfcc-0.7.3/configure
gfcc-0.7.3/configure.in
gfcc-0.7.3/install-sh
gfcc-0.7.3/ltconfig
gfcc-0.7.3/ltmain.sh
gfcc-0.7.3/missing
gfcc-0.7.3/mkinstalldirs
gfcc-0.7.3/texinfo.tex
```

```
gfcc-0.7.3/main.c
gfcc-0.7.3/file.c
gfcc-0.7.3/window.c
gfcc-0.7.3/firewall.c
gfcc-0.7.3/chain.c
gfcc-0.7.3/host.c
gfcc-0.7.3/masq.c
gfcc-0.7.3/gtkutil.c
gfcc-0.7.3/misc.c
gfcc-0.7.3/gfcc.h
gfcc-0.7.3/libipfwc.tar.gz
gfcc-0.7.3/README.ko
gfcc-0.7.3/rules/
gfcc-0.7.3/rules/Makefile.in
gfcc-0.7.3/rules/Makefile.am
gfcc-0.7.3/rules/configure
gfcc-0.7.3/rules/configure.in
gfcc-0.7.3/rules/masquerade.rule
gfcc-0.7.3/rules/standalone.rule
gfcc-0.7.3/rules/routable.rule
gfcc-0.7.3/rules/ppp-masquerade.rule
gfcc-0.7.3/data/
gfcc-0.7.3/data/Makefile.in
gfcc-0.7.3/data/Makefile.am
gfcc-0.7.3/data/configure
gfcc-0.7.3/data/configure.in
gfcc-0.7.3/data/gtkrc
gfcc-0.7.3/data/host_networks
```

**# 디렉토리 이동**

```
[root@bluebird src]# cd gfcc-0.7.3
```

**# 파일 확인**

```
[root@bluebird gfcc-0.7.3]# ls
AUTHORS      README      configure     install-sh      mkinstalldirs
COPYING      README.ko   configure.in  libipfwc.tar.gz rules
ChangeLog    TODO        data          ltconfig        stamp-h.in
INSTALL      aclocal.m4  file.c        ltmain.sh       texinfo.tex
Makefile.am  chain.c     firewall.c    main.c          window.c
Makefile.in  config.guess gfcc.h      masq.c
NEWS         config.h.in gtkutil.c    misc.c
Pics         config.sub  host.c       missing
```

```
# libipfwc 압축 및 tar 풀기
[root@bluebird gfcc-0.7.3]# tar -zxvf libipfwc.tar.gz
libipfwc/
libipfwc/libipfwc.c
libipfwc/libipfwc.h
libipfwc/ipfwc_kernel_headers.h
libipfwc/Makefile
[root@bluebird gfcc-0.7.3]# cd libipfwc
[root@bluebird libipfwc]# ls
Makefile ipfwc_kernel_headers.h libipfwc.c libipfwc.h
[root@bluebird libipfwc]# make
gcc -Wall -Wunused -g -O   -c libipfwc.c -o libipfwc.o
ar rv libipfwc.a libipfwc.o
a - libipfwc.o
[root@bluebird libipfwc]# cd ..

# configure 이용
[root@bluebird gfcc-0.7.3]# ./configure —with-ipfwc=./libipfwc
creating cache ./config.cache
checking for a BSD compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking whether make sets ${MAKE}... yes
checking for working aclocal... found
checking for working autoconf... found
checking for working automake... found
checking for working autoheader... found
checking for working makeinfo... found
checking host system type... i686-pc-linux-gnu
checking build system type... i686-pc-linux-gnu
checking for ranlib... ranlib
checking for gcc... gcc
checking whether the C compiler (gcc  ) works... yes
checking whether the C compiler (gcc  ) is a cross-compiler... no
checking whether we are using GNU C... yes
checking whether gcc accepts -g... yes
checking for ld used by GCC... /usr/bin/ld
checking if the linker (/usr/bin/ld) is GNU ld... yes
checking for BSD-compatible nm... /usr/bin/nm -B
checking whether ln -s works... yes
updating cache ./config.cache
checking for object suffix... o
checking for gcc option to produce PIC... -fPIC
checking if gcc PIC flag -fPIC works... yes
checking if gcc supports -c -o file.o... yes
```

```
checking if gcc supports -c -o file.lo... yes
checking if gcc supports -fno-rtti -fno-exceptions ... yes
checking if gcc static flag -static works... -static
checking if the linker (/usr/bin/ld) is GNU ld... yes
checking whether the linker (/usr/bin/ld) supports shared libraries... yes
checking command to parse /usr/bin/nm -B output... yes
checking how to hardcode library paths into programs... immediate
checking for /usr/bin/ld option to reload object files... -r
checking dynamic linker characteristics... Linux ld.so
checking if libtool supports shared libraries... yes
checking whether to build shared libraries... yes
checking whether to build static libraries... yes
checking for objdir... .libs
creating libtool
loading cache ./config.cache
checking for gcc... (cached) gcc
checking whether the C compiler (gcc -g -O2 ) works... yes
checking whether the C compiler (gcc -g -O2 ) is a cross-compiler... no
checking whether we are using GNU C... (cached) yes
checking whether gcc accepts -g... (cached) yes
checking for a BSD compatible install... /usr/bin/install -c
checking whether ln -s works... (cached) yes
checking for gawk... gawk
checking for gtk-config... /usr/bin/gtk-config
checking for GTK - version >= 1.2.0... yes
checking for libipfwc... yes
checking for Linux 2.1.102 or higher... yes
checking for main in -lX11... no
checking for main in -lXext... no
checking for main in -ldl... no
checking for main in -lgdk... no
checking for main in -lglib... no
checking for main in -lgmodule... no
checking for main in -lgtk... no
checking for main in -lm... no
checking how to run the C preprocessor... gcc -E
checking for ANSI C header files... no
checking for unistd.h... yes
checking for working const... no
checking for 8-bit clean memcmp... no
updating cache ./config.cache
creating ./config.status
creating rules/Makefile
creating data/Makefile
```

```
creating Makefile
creating config.h
[root@bluebird gfcc-0.7.3]# make
make  all-recursive
make[1]: Entering directory `/root/firewall/src/gfcc-0.7.3'
Making all in rules
make[2]: Entering directory `/root/firewall/src/gfcc-0.7.3/rules'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/root/firewall/src/gfcc-0.7.3/rules'
Making all in data
make[2]: Entering directory `/root/firewall/src/gfcc-0.7.3/data'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/root/firewall/src/gfcc-0.7.3/data'
make[2]: Entering directory `/root/firewall/src/gfcc-0.7.3'
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c main.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c file.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c window.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c firewall.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c chain.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c host.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c masq.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c gtkutil.c
gcc -DHAVE_CONFIG_H -I. -I. -I.     -I/root/firewall/src/gfcc-0.7.3/./libipfwc -
g -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/shar
e/gfcc\" -c misc.c
/bin/sh ./libtool --mode=link gcc  -I/root/firewall/src/gfcc-0.7.3/./libipfwc -g
 -O2 -I/usr/X11R6/include -I/usr/lib/glib/include -DGFCC_HOME=\"/usr/local/share
/gfcc\"  -o gfcc main.o file.o window.o firewall.o chain.o host.o masq.o gtkuti
```

```
l.o misc.o  -L/root/firewall/src/gfcc-0.7.3/./libipfwc  -L/usr/lib -L/usr/X11R6/
lib -lgtk -lgdk -rdynamic -lgmodule -lglib -ldl -lXext -lX11 -lm -lipfwc
gcc -I/root/firewall/src/gfcc-0.7.3/./libipfwc -g -O2 -I/usr/X11R6/include -I/us
r/lib/glib/include -DGFCC_HOME=\"/usr/local/share/gfcc\" -o gfcc main.o file.o w
indow.o firewall.o chain.o host.o masq.o gtkutil.o misc.o -L/root/firewall/src/g
fcc-0.7.3/./libipfwc -L/usr/lib -L/usr/X11R6/lib -lgtk -lgdk -rdynamic -lgmodule
 -lglib -ldl -lXext -lX11 -lm -lipfwc
make[2]: Leaving directory `/root/firewall/src/gfcc-0.7.3'
make[1]: Leaving directory `/root/firewall/src/gfcc-0.7.3'

# 설치
[root@bluebird gfcc-0.7.3]#make istall
```

# 7 IPCHAINS WEB INTERFACE - FWT

http://home.swipnet.se/rodac/doc/fwt.html
Current version: 1.2.6

## ① 소개

FWT는 리눅스의 방화벽 기능을 웹 기반 환경에서 설정하게 해주는 툴이다. FWT는 일반적으로 CGI-BIN 형태로 작동되고 root 권한이 필요하기 때문에 SUID root를 사용한다. FWT로 작성한 룰은 저장되고, 부팅시 /etc/rc.d/init.d/fwt(레드햇 기준)에 의해 작동된다.

## ② 설치

**요구되는 소프트웨어:**

```
Linux kernel with Firewall and Masquerading support.
Apache web server
Ipchains
Ipmasqadm
Iroute2
You also need a 2.2.x (x>=12) kernal recompiled to use advanced routing.
```

**루트 권한으로 할 일:**

fwt라는 이름의 사용자를 만든다. FWT의 대부분 cgi는 SUID 실행 파일 형태로 사용된다.

Note: 중요한 몇 개의 cgi는 root SUID가 사용된다.

```
useradd -c "FWT user" -d /dev/null -g nobody -s /dev/null fwt

# tar ball을 풀고 컴파일 한다. 그전에 Makefile을 손봐야 한다.
tar xvfz fwt_v1_2_1.tgz
cd fwt
make

# 설치
make install

# 비밀번호 설정
echo "password" >/home/httpd/cgi-bin/fwt/login/password
chown fwt:nobody /home/httpd/cgi-bin/fwt/login/password
chmod 400 /home/httpd/cgi-bin/fwt/login/password

# 웹 클라이언트에서 다음과 같은 형식의 URL을 입력한다 http://ip_of_firewall/fwt/fwt.html
```

# 8 실제 IP를 가지는 IP 포워딩 머신 구축

## ① 들어가기에 앞서

Linux가 중소형 네트워크 환경에서 훌륭하게 작동하는 것은 리눅스를 사용하고 있지 않은 사람도 익히 들어 알고 있겠지만, 실제 문서자료를 아무리 찾아봐도 머스커레이딩방식을 이용한 방화벽 구성에 대한 글만 있을 뿐 이미 구성된 실제 IP를 가지는 네트워크에서 방화벽 및 포워딩 머신을 구축하는 것에 대해서는 자료가 부족하여 본인은 이렇게 Mini-HOWTO를 감히 쓰게 되었다.

## ② 요구사항

- 간단한 네트워크 개념(routing , netmask , forwarding , etc.)
- 실제 Router랑 연결되어 있거나 Wan 카드가 부착된 리눅스 머신
- 담배(비흡연자는 콜라)

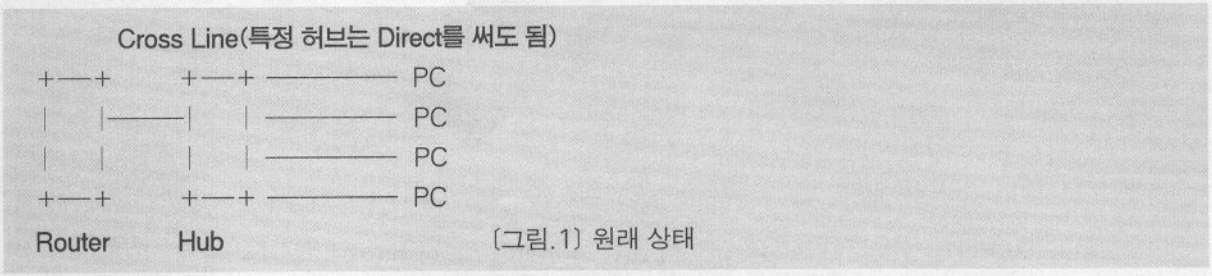
## ③ 본인의 장비 및 환경

- Router: CISCO 4500M + 1 Fastethernet module + 4 serial module
- CSU : 보라넷 임대 장비
- Linux1: Intel pentium III 500 + 128M ram + 9G HDD
- NIC: 3com 905 , 905b
- HUB: Intel 405T standalone switch HUB * 3
- IP 대역: 211.50.38.0/25 (255.255.255.128)
- 디스 한 보루 + 지포 라이터
- Network Bandwidth: T1

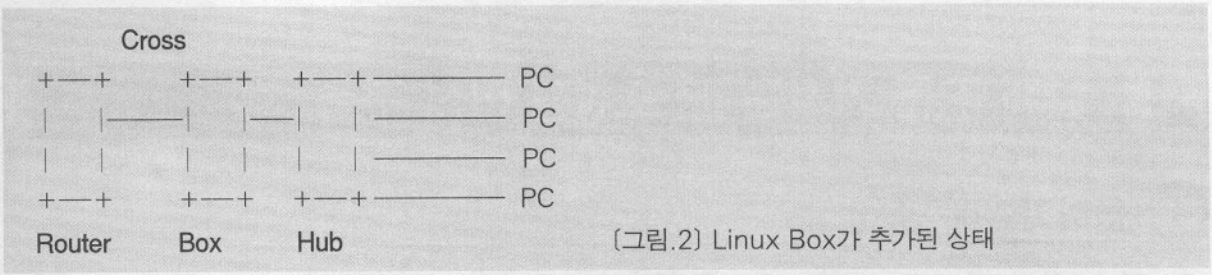
## ④ 닭질의 시작

우선, 자신의 라우터나 랜카드에 맞는 크로스 케이블을 만든다. 크로스 케이블을 만들줄 모르는 사람은 잘하는 사람에게 배우거나 인터넷에서 검색해본다. 아니면 용산이나 테크노마트 등지에서 크로스된 라인을 사든지 크로스시키는 잭을 사면 된다.

중요한 것은 이미 구성되어 있는 네트워크에 리눅스 박스를 추가함으로써 방화벽 및 포워딩 기능을 제공하는 것이기 때문에 기존 라우터에서 허브로 연결되는 라인을 중간에서 가로채야만 한다. 그림으로 도식해보겠다.




〔그림.1〕 원래 상태




〔그림.2〕 Linux Box가 추가된 상태

그럼 중간에서 가로채기만 하면 되느냐? 절대로 될 리가 없을것이다 :) 우선 리눅스 박스의 세팅부터 해보자. 편의를 위해 본인의 실제 IP를 사용하겠다. 착오없기 바란다. 중요한 것은 리눅스 박스에 랜카드 두 장 이상이 붙어 있어야 한다는 것이다. 그리고 포워딩시 신뢰성과 속도를 위해 버스 마스터링이 잘되는 100Mbps PCI 랜카드를 사용하기를 권장한다. 3Com이나 Intel의 100Mbps 랜카드라면 무난할 것이다. 여기서는 3Com 3c905와 905b를 사용한다. 하드웨어적인 준비가 끝났으면 커널 컴파일을 한다.

네트워크 부분에서 꼭 IP_FORWARDING을 설정해야 한다. ipchains 사용을 위해 IP_FIRE WALL을 설정하는것도 잊지말자. 이렇게 만들 커널로 재부팅후 랜카드가 잘 인식된다면 이제는 약간의 작업이 필요하다. 우선, 라우터와 바로 물린 랜카드의 디바이스를 알아야 한다(본인은 eth1). 이제 다음부터가 중요한데, 수많은 시행착오끝에 얻은 결론을 말하자면 라우터의 IP와 리눅스의 라우터와 물린 랜카드 IP를 private IP - 즉 실제로 라우팅용으로 사용하지 않는 사설망용 IP - 로 배정해야 한다는 것이다.

본인의 경우 Router에는 10.0.0.1을, eth1에는 10.0.0.2를 배정했다. 실제 IP로 라우터와 랜카드에 배정을 할 경우에는 서브넷으로 다시 나누지 않은 네트워크의 경우(즉 211.50. 38.0/128을 다시 더 나누지 않는 이상) 라우터에서 라우팅을 제대로 해줄수가 없기 때문에 두 장치에는 라우팅의 보증을 위해 사설 IP를 사용한 것이다. 실제 이런 문제 때문인지 ISP에서는 시리얼(라우터간 통신 IP)에는 netmask 255.255.255.252를 사용한다.

IP를 배정한 다음, 나머지는 그다지 중요한 게 없다. 이 글의 목적이 원래 존재하는 네트워크에 영향을 미치지 않고 조용히 리눅스를 집어넣는 것이기 때문에 나머지 랜카드에는 원래 라우터가 쓰던 IP(일반적으로 게이트웨이 IP)를 배정한다.

```
Network: 211.50.38.0/25

     Cross          +————————+
+—+                 |+—+ +—+|          +—+ ———————— PC  211.50.38.2
|  |—————————————— |  | |  |—————————|  | ———————— PC  211.50.38.3
|  |                ||  | |  ||        |  | ———————— PC  211.50.38.4
+—+                 |+—+ +—+|          +—+ ———————— PC
                    +————————+
serial:             NIC0:               HUB
211.50.1.202/30     10.0.0.2
ip:                 NIC1:
10.0.0.1                 211.50.38.1
```

일단 이렇게 구성되면 IP_FORWARDING이 알아서 이루어지기 때문에 허브쪽에 물린 PC에서 외부로 나가는 네트워크를 쓸 경우 무조건 우리가 만든 리눅스 포워딩 머신을 지나가게 된다. 자, 이제는 포워딩 머신에서 ipchains를 이용해 각종 방화벽 설정을 할 수 있다. 회사 내부에서 업무시간중 채팅을 막기 위해 irc 포트로 나가는 모든 네트워크를 막을 수도 있을 것이고, 지금 활성화된 네트워크도 끊을 수가 있다 :)

실례를 위해 본인의 설정 몇 가지를 보이겠다.

**- 라우터 설정**

```
〔root@unixian /root〕# ztelnet 10.0.0.1
Trying 10.0.0.1...
Connected to 10.0.0.1.
Escape character is '^]'.
WyzSoft Research & Development Lab. Access-control Router

User Access Verification

Password:
router>en
Password:
router#show running-config
Building configuration...

Current configuration:
!
version 11.2
no service password-encryption
no service udp-small-servers
no service tcp-small-servers
!
hostname router
!
enable secret 5 ****************************
!
```

```
ip subnet-zero
ip domain-name wyzlab.com
ip name-server 210.205.2.52
!
interface Serial0
 ip address 211.50.1.202 255.255.255.252
!
interface Serial1
 no ip address
 shutdown
!
interface Serial2
 no ip address
 shutdown
!
interface Serial3
 no ip address
 shutdown
!
interface FastEthernet0
 ip address 10.0.0.1 255.0.0.0
!
no ip classless
ip route 0.0.0.0 0.0.0.0 211.50.1.201
ip route 211.50.38.0 255.255.255.128 10.0.0.2
logging buffered informational
logging console informational
logging monitor informational
logging 211.50.38.2
snmp-server community wyzlab RO
snmp-server trap-authentication
banner motd ^CWyzSoft Research & Development Lab. Access-control Router^C
!
line con 0
line aux 0
line vty 0 4
 password ******************
 login
!
end
router#
```

### 방화벽 리눅스 박스의 설정

```
[root@bluebird jhjung]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
211.50.38.1     *               255.255.255.255 UH    0      0        0 eth0
10.0.0.2        *               255.255.255.255 UH    0      0        0 eth1
10.0.0.0        *               255.255.255.252 U     0      0        0 eth1
211.50.38.0     *               255.255.255.128 U     0      0        0 eth0
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         10.0.0.1        0.0.0.0         UG    0      0        0 eth1
```

### 일반 PC의 설정(Linux 기준)

```
[root@unixian /root]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
211.50.38.38    *               255.255.255.255 UH    0      0        0 eth1
211.50.38.0     *               255.255.255.128 U     0      0        0 eth1
127.0.0.0       *               255.0.0.0       U     0      0        0 lo
default         211.50.38.1     0.0.0.0         UG    0      0        0 eth1
[root@unixian /root]#
```

### 일반 PC에서의 traceroute 결과

```
[root@unixian /root]# traceroute linux.sarang.net
traceroute to linux.sarang.net (210.122.59.30), 30 hops max, 38 byte packets
 1  211.50.38.1 (211.50.38.1)  0.241 ms  0.192 ms  0.149 ms
 2  10.0.0.1 (10.0.0.1)  2.350 ms  0.884 ms  0.777 ms
 3  211.50.1.201 (211.50.1.201)  4.102 ms  3.571 ms  3.443 ms
 4  anybbb185-fe4-1-0.rt.bora.net (210.120.252.102)  4.060 ms  3.680 ms  3.895 ms
 5  anyg4-ge2-0.rt.bora.net (210.120.193.145)  3.775 ms  3.728 ms  3.692 ms
 6  selg2-pos8-0.rt.bora.net (210.120.192.117)  4.423 ms  3.924 ms  3.943 ms
 7  ysng12kix4-ge5-0.rt.bora.net (210.120.192.69)  4.049 ms  4.053 ms  4.061 ms
 8  210.107.53.66 (210.107.53.66)  4.911 ms  5.091 ms  5.216 ms
 9  203.255.117.248 (203.255.117.248)  5.077 ms  5.759 ms  5.190 ms
10  10.241.1.178 (10.241.1.178)  12.136 ms  11.571 ms  11.424 ms
11  linux.sarang.net (210.122.59.30)  11.841 ms  10.977 ms  10.900 ms
```

- 외부 네트워크에서의 traceroute 결과

```
[jhjung@www jhjung]$ /usr/sbin/traceroute 211.50.38.38
traceroute to 211.50.38.38 (211.50.38.38), 30 hops max, 40 byte packets
 1  210.118.74.1 (210.118.74.1)  2.944 ms  2.871 ms  2.908 ms
 2  210.118.73.1 (210.118.73.1)  0.914 ms  0.905 ms  0.877 ms
 3  210.118.49.97 (210.118.49.97)  3.146 ms  3.460 ms  2.889 ms
 4  dacomkix-sds-s2-0.rt.bora.net (203.233.37.221)  5.048 ms  5.653 ms  4.491 ms
 5  selg2-ge5-0.rt.bora.net (210.120.192.65)  4.608 ms  5.121 ms  4.654 ms
 6  anyg4-pos8-0.rt.bora.net (210.120.192.118)  5.543 ms  5.481 ms  4.817 ms
 7  anybbb185-ge1-0-0.rt.bora.net (210.120.193.147)  5.259 ms  69.667 ms  5.210 ms
 8  anyaba74-fe1-0-0.rt.bora.net (210.120.252.74)  5.457 ms  6.135 ms  5.139 ms
 9  211.50.1.202 (211.50.1.202)  9.027 ms  9.660 ms  8.556 ms
10  10.0.0.2 (10.0.0.2)  8.838 ms  9.265 ms  8.908 ms
11  211.50.38.38 (211.50.38.38)  17.076 ms  8.672 ms  8.902 ms
```

### ⑤ 사용후기

외부 네트워크가 T1이기 때문에 아직 폭주를 경험해 보지는 못했지만 상당히 포워딩/방화벽 머신으로는 고사양이기 때문에 패킷 로스율이 거의 없이 잘 작동한다. 사실 포워딩 자체로는 의미가 없고 방화벽 설비를 아주 염가로 구축할 수 있기 때문에 사랑받는 관리자가 될 수 있을 것이다. 포워딩 머신에서 네트워크 분석툴들을 돌리면(IPtraf, ntop 등) 모든 네트워크를 감시할 수 있기때문에 관리가 한결 수월해진다.

아직 해결하지 못한 점이 있다면 포워딩 머신 자체에서는 외부 네트워크로 연결할 수가 없다. 다른 PC에서 가지고 나가는 IP는 단지 포워딩만 되기 때문에 상관없지만, 실제 포워딩 머신의 기본 IP는 10.0.0.2로 나가기 때문에 외부로 라우팅될 수 없는 문제인 것 같다. IP tunneling을 쓰면 가능하겠지만 수익체감의 법칙상 이정도는 큰 문제가 아니기 때문에 죽어도 포워딩 머신에서 네트워크를 써야겠다는 분은 IP tunnell-ing을 연구해 보기 바란다. 참고로, IPchains를 쉽게 설정해 주는 도구는 http://www.freshmeat.net에서 ipchains로 검색해 보기 바란다. 본인이 가장 즐겨 쓰는 도구는 gtk front-end인 gfcc이다. 그럼, 흡연을 위해 이만 줄입니다.

## 9 결론

네트워크는 날이 갈수록 복잡해지고 있고 그 반작용으로 네트워크 침입기술이 더욱 고도화, 지능화되고 있다. 하지만 이 기술들은 웹과 그밖 인터넷 컨텐츠의 발달로 인해 소위 스크립트 키디(Script kiddy)라 불리는 비전문적인 침입자들에게 쉽게 그 길에 들어 설 수 있게 해주고 있기 때문에 이제는 정확한 목적이 없는 대량 공격(Massive Attack) 시대가 되었고, 네트워크 연결된 호스트나 그밖의 장비들은 이제는 더 이상 그런 위협에서 안전하다고는 말할 수 없다. 오늘 이 세미나에서 설명한 기술들은 리눅스에서 구현할 수 있는 아주 간단하면서도 강력한 차단기술이기 때문에 그런 위협에서 조금이나마 벗어날 수 있고, 관리자 자신의 네트워크 이해도 향상에 조금이나마 도움이 될 것이라 생각한다.