한글 FreeBSD 핸드북

Korea FreeBSD Users Group – http://www.kr.FreeBSD.org/

Security: 보안

이 영 옥 <rick@rickinc.com> 역.

2004 년 8 월 7 일.
문서갱신번호 1.1

본 문서는 FreeBSD Documentation Project 에서 발행하는 FreeBSD Handbook 을 번역한 것으로 2004 년 8 월 7 일자 수정본을 기준으로 합니다. FreeBSD Handbook 은 지속적으로 갱신되기 때문에, 본 문서의 발간 이후에 갱신된 내용이 무엇인지 알고자 한다면, CVS Repository 를 조회하여 해당일 이후에 갱신된 내용만을 쉽게 찾아볼 수 있습니다. 또한 표지에 표시되는 날짜와 문서갱신번호는 이 문서의 갱신상태를 나타냅니다. 날짜는 번역의 대상이 되는 오리지날 핸드북의 최종 수정일자를 표시하며, 문서갱신번호는 문서에 수정이 있을 때 증가하므로 사용자가 이 문서의 갱신상태를 쉽게 파악할 수 있도록 합니다. 본 문서는 이영옥이 관리, 배포하므로 내용에 오탈자 및 오역이 발견될 경우엔 역자에게 알려주어 수정될 수 있도록 해 주십시오.

14 장 보안

14.1 개요

이번 장에서는 일반적인 정책 중 가장 최우선인 시스템 보안 개념과 FreeBSD 의 몇 가지 발전된 주제를 소개한다. 여기서 다루는 많은 주제는 시스템에 적용할 수 있고 일반적인 인터넷 보안에도 적용할 수 있다. 인터넷의 모든 사람들이 여러분들의 친절한 이웃이기를 원하겠지만 더 이상 안전한 곳이 아니다. 시스템 보안은 데이터, 지적 재산, 시간 등을 해커 같은 사람들로부터 지키는 필수요소다.

FreeBSD 는 시스템과 네트워크의 보안을 강화하기 위해 여러 유틸리티와 메커니즘을 제공한다.

이번 장을 읽고 다음 사항을 알 수 있다:

- FreeBSD 에 고려할 기본적인 시스템 보안개념
- FreeBSD 에서 이용할 수 있는 DES 와 MD5 같은 다양한 암호 메커니즘
- one-time 패스워드 인증은 어떻게 설정하는가
- FreeBSD 5.0 이전 릴리즈에 KerberosIV 는 어떻게 설정하는가
- FreeBSD 5.0 이후 릴리즈에 Kerberos5 를 어떻게 설정하는가
- IPFW 을 사용한 방화벽 생성
- IPsec 를 설정하고 FreeBSD/Windows 머신 사이에 VPN 은 어떻게 생성하는가
- FreeBSD 에서 SSH 를 사용하기 위해 OpenSSH 는 어떻게 설정하는가
- 파일시스템 ACL 은 무엇이고 어떻게 사용하는가

- FreeBSD 보안 권고를 어떻게 이용하는가

이번 장을 읽기 전에 다음 내용을 알고 있어야 된다:

- FreeBSD 와 인터넷의 기본.

14.2 소개

보안은 시스템 관리자가 해야 될 업무의 시작과 끝이다. 모든 BSD 유닉스와 멀티유저 시스템이 고유한 보안 메커니즘을 가지고 있지만 유저들이 "정직"하도록 추가적인 보안 메커니즘을 구축하고 관리하는 일이 시스템 관리자의 가장 큰 업무일 것이다. 머신은 보안에 관심을 가진 만큼 안전해지지만 보안은 편리함과 상반된다. 보통 유닉스 시스템은 동시에 수많은 프로세스를 운용할 수 있고 이들 프로세스 중 대부분은 서버로 운용된다. 이 말은 외적인 요소와 연결되어 통신할 수 있음을 의미한다. 과거의 미니 컴퓨터와 메인 프레임은 오늘날의 데스크톱이 되고 컴퓨터가 네트워크와 인터넷에 이용됨으로 보안 문제가 더욱 커지고 있다.

보안은 층으로 이루어진 양파껍질 같은 접근을 통하여 가장 완벽하게 수행된다. 간결하게 말해서 해야 될 일은 사용하기 편할 정도의 많은 보안계층을 만들고 침입에 대비하여 주의깊게 시스템을 살펴보는 것이다. 지나치게 보안사항을 많이 만들거나 보안 메커니즘의 가장 중요한 관점 중 하나인 탐색에 방해가 되기를 원하지 않을 것이다. 예를 들어 각각의 시스템 바이너리에 schg 플래그(chflags(1))를 설정하면 안 된다. 왜냐하면 일시적으로는 바이너리들을 보호하겠지만 공격자들에 의한 변화를 쉽게 잡아내지 못하여 결국 보안시스템이 공격자들을 탐지하지 못한다.

다 계층 보안 레이어

시스템 보안은 시스템다운이나 시스템을 이용할 수 없도록 시도하는 것을 포함하여 다양한 공격 양상과 관련이 있다. 보안문제는 몇 개의 카테고리로 나누어진다:

① 서비스 거부 공격

② 사용자 계정 획득

③ 접근할 수 있는 서버를 통해 root 권한 획득

④ 유저 권한을 통한 root 권한 획득

⑤ 백 도어 생성

서비스 거부 공격은 머신에 필요한 리소스를 빼앗는 행위다. 전형적인 DoS 공격은 머신이 다운되도록 하거나 다른 말로 서버나 네트워크 스택이 반응할 수 없도록 하여 머신을 사용할 수 없도록 하는 맹목적인 메커니즘이다. 어떤 DoS 공격은 네트워크 스택에서 싱글 패킷으로 머신이 다운되는 버그를 이용한다. 후자는 커널의 버그를 패치하여 해결할 수 있다. 서버에 대한 공격은 종종 시스템에 악의적인 상황을 초래하는 서버 부하를 적절히 제한하는 옵션으로 해결할 수 있다. 맹목적인 네트워크 공격은 해결하기 어렵다. 예를 들어 스푸핑(spoof) 공격은 멈추기가 거의 불가능하여 인터넷으로부터 시스템을 계속해서 끊어놓는다. 머신을 다운시킬 수 없겠지만 인터넷 연결을 집중적으로 공격할 수 있다.

유저 권한 획득은 DoS 공격보다 더 빈번하다. 많은 시스템 관리자는 아직도 표준 telnetd, rlogind, rshd 와 ftp 서버를 머신에서 운용한다. 기본적으로 이러한 서버는 연결을 암호화해서 운용하지 않는다. 그 결과 다소간의 유저를 가지고 있다면 원격에서(시스템에 로그인하는 아주 일반적이고 편리한 방법) 시스템에 로그인하는 유저 중 한 명 이상의 유저가 패스워드를 스니프(sniff) 당할 것이다. 주의 깊은 시스템 관리자는 로그인에 성공한 유저라도 의심스러운 소스를 찾으려고 원격접근 로그를 분석할 것이다.

한번이라도 공격자가 유저 계정으로 접근했다면 공격자가 항상 root 권한을 획득할 수 있다고 가정할 수 있다. 그러나 사실은 보안상 잘 관리되는 시스템이라도 root 권한을 획득하기 위해 유저 계정에 접근해야 되는 것은 아니다. root 에 접근하지 못한 공격자는 그의 흔적을 숨기지 못하고 자신이 획득한 유저 파일을 삭제하는 외에 머신을 다운시키지 못하기 때문에 이 차이는 중요하다. 유저는 시스템 관리자의 경고를 무시하는 경향이 있기 때문에 유저 계정 획득은 매우 빈번하다.

시스템 관리자는 잠재적으로 머신에서 root 를 깰 수 있는 다양한 방법이 있음을 생각해야 된다. 공격자가 root 패스워드를 알 수 있고 root 계정으로 실행 중인 서버의 버그를 찾아서 네트워크를 통해 그 서버에 연결하여 root 를 깰 수도 있다. 또한 공격자가 유저 계정을 획득하였다면 root 를 깰 수 있는 suid root 프로그램의 버그를 알고 있을 것이다. 공격자가 머신에서 root 를 깨는 방법을 찾았다면 공격자는 백 도어를 설치할 필요가 없을 것이다. 그러나 root 취약점들은 공격자가 자신의 흔적을 지우기 위해 많은 노력이 필요하기 때문에 대부분의 공격자들은 백 도어를 설치한다. 백 도어는 공격자가 쉽게 root 로 다시 시스템에 접근할 수 있는 방법을 제공하지만 영리한 시스템 관리자에게는 그들의 침투를 쉽게 알 수 있는 방법을 제공한다. 공격자가 처음 공격한 취약점을 막을 수도 있기 때문에 백 도어를 설치하지 못하게 하는 것이 실제로 보안에 나쁜 영향을 미칠 수도 있다.

보안 개선 사항은 다양한 층으로 된 양파 껍질처럼 접근할 수 있는 방법이어야 되고 다음과 같이 분류할 수 있다:

① root 와 스태프(staff) 계정 보안

② root 로 실행되는 서버와 suid/sgid 바이너리 보안

③ 유저 계정 보안

④ 패스워드 파일 보안

⑤ 커널 코어, raw 장치와 파일시스템 보안

⑥ 부적절한 시스템 변화를 빨리 감지한다.

⑦ 편집증

이번 장의 다음 섹션은 위에서 열거한 내용에 대해 심도 있게 다룬다.

14.3 FreeBSD 보안 강화

명령어 대 프로토콜: 이 문서에서는 명령어와 어플리케이션에 **bold** 체를 사용했다. ssh 는 명령어일 뿐만 아니라 프로토콜에 가깝기 때문에 ssh 와 같은 보기에도 사용했다.

이 섹션은 위에서 언급한 FreeBSD 시스템 보안 개선사항에 대해 설명한다.

14.3.1 root 계정과 스텝 계정 보안

첫째로 root 계정의 보안을 강화하지 않았다면 스텝 계정 사용자들을 귀찮게 하지 않는다. 대부분의 시스템은 root 계정에 패스워드를 할당한다. 첫째로 패스워드는 *항상* 노출될 수 있음을 생각한다. 이 의미는 패스워드를 삭제하라는 뜻이 아니다. 패스워드는 콘솔을 통해 머신에 접속하기 위해 항상 필요하다. 이 말은 패스워드 없이 콘솔 밖에서 su(1) 명령을 사용할 수 있게 해서는 안된다. 예를 들어 /etc/ttys 파일의 pty 를 insecure 지정해서 telnet 이나 rlogin 을 사용하여 직접 root 로 로그인할 수 없도록 한다. **sshd** 같은 다른 로그인 서비스도 root 로 직접 로그인하는 것을 막는다. /etc/ssh/sshd_config 파일에서 *PermitRootLogin* 을 *NO* 로 설정해서 직접 로그인을 막을 수 있다. FTP 같은 접근 방법은 종종 크랙(cracks) 될 수 있다. 따라서 직접 root 로그인은 시스템 콘솔로만 접속하도록 해야 된다.

물론 시스템 관리자는 root 가 될 수 있어야 하기 때문에 몇 가지 관문을 열어 놓는다. 그러나 이러한 관문을 사용하기 위해 추가적인 패스워드 인증이 필요하도록 한다. root 가 될 수 있도록 하는 한가지 방법은 적당한 스텝 계정을 wheel 그룹에(/etc/group 에서) 추가한다. wheel 그룹에 있는 스텝 멤버만 su 로 root 가 될 수 있다. 절대 일반적인 스텝 멤버의 패스워드 엔트리에 wheel 그룹을 넣어서 root 로 접근하는 권한을 주지 않는다. 스텝 계정은 staff 그룹에 있어야 하고 /etc/group 파일에서 wheel 그룹을 추가한다. 실제로 root 접근이 필요한 스텝 멤버만 wheel 그룹에 넣어야 된다. 또한 Kerberos 같은 인증방법을 사용한다면 Kerberos 를 사용하는 root 계정의 .k5login 파일은 유저가 wheel 그룹에 있지 않아도 ksu(1)로 root 가 될 수 있다. 침입자가 패스워드 파일을 가지고 스텝 계정으로 침입했다면 wheel 메커니즘으로 침입자는 root 를 파괴할 수 있기 때문에 Kerberos 가 더 좋은 방법이다. 그러나 wheel 메커니즘이라도 가지고 있는 것이 아무런 보안 옵션을 가지고 있지 않은 것보다 좋다.

최후의 root 접근은 다른 로그인 접근 방법을 사용하고, 스텝 계정을 간접적으로 안전하게 만들기 위해 "*"로 알려진 암호화된 패스워드를 스텝 계정에 사용한다. vipw(8) 명령을 사용

하면 하나의 "*" 문자로 암호화된 패스워드의 각 인수를 바꿀 수 있다. 이 방법은 /etc/master.passwd 파일을 업데이트해서 user/password 데이터베이스가 패스워드 인증 로그인을 비활성한다.

스텝 계정 엔트리는 다음과 같다:

```
foobar:R9DT/Fa1/LV9U:1000:1000::0:0:Foo Bar:/home/foobar:/usr/local/bin/tcsh
```

이것을 다음과 같이 바꾼다:

```
foobar:*:1000:1000::0:0:Foo Bar:/home/foobar:/usr/local/bin/tcsh
```

암호화된 패스워드는 절대 "*"와 일치하지 않기 때문에 이렇게 변경하면 일반적인 로그인을 막을 수 있다. 이렇게 되면 스텝 멤버는 인증을 받기 위해 kerberos(1)나 공인/개인 키 쌍을 사용하는 ssh(1) 같은 다른 메커니즘을 사용해야 된다. Kerberos 같은 것을 사용할 때 보통 Kerberos 서버를 실행하는 머신과 데스크톱 워크스테이션은 안전해야 된다. ssh 에 공인/개인키 쌍을 사용할 때 일반적인 로그인에 사용되는 머신도(보통 누군가의 워크스테이션) 보안에 안전해야 된다. 보안의 추가적인 계층은 ssh-keygen(1)로 키 쌍을 생성할 때 패스워드 보호 키 쌍으로 키 쌍을 추가할 수 있다. 스텝 계정의 패스워드를 "*"로 하는 것도 설정해둔 안전한 접근방법으로 스텝 맴버만 로그인하는 것을 보장한다. 이 방법은 모든 스텝 멤버가 많은 침입자들이 사용하는 중요한 취약점을 닫고 안전하고 암호화된 연결을 강제로 사용하게 한다: 안전하지 않은 머신에서 네트워크가 탐지되기 때문이다.

더욱 간접적인 보안 메커니즘도 보안이 강화된 서버에서 보안설정이 별로 없는 서버로 로그인 하는 것이다. 예를 들어 메인 박스에 모든 종류의 서버가 운용 중이라면 워크스테이션에는 운용하는 것이 없어야 된다. 워크스테이션이 안전하도록 운용하는 서버가 가능한 없어야 되고 패스워드 화면보호기를 운용해야 된다. 물론 워크스테이션에 물리적인 접근이 가능하면 공격자는 설치한 모든 종류의 보안을 깨뜨릴 수 있다. 이 사항을 고려해야 되지만 워크스테이션이나 서버에 물리적으로 접근할 수 없는 사람들로부터 주로 네트워크를 통해 원격지에서 공격이 이루어진다는 사실 또한 고려해야 된다.

Kerberos 같은 것을 사용해서 한곳에서 스텝 계정의 패스워드 변경을 허용하거나 방지할 수 있고 스텝 멤버의 계정을 가지고 있는 모든 머신에 즉시 적용된다. 스탭 멤버 계정이 획득당했다면 획득 당한 계정의 패스워드가 모든 머신에서 변경됨을 고려해야 된다. 따로 운영하는 패스워드를 N 대의 머신에서 변경하는 것은 엄청난 작업일 것이다. Kerberos 로 패스워드를 다시 만들 수 있다: Kerberos 는 일정 시간이 지나면 타임아웃 되도록 할 뿐 아니라

Kerberos 시스템은 특정 시간이 경과하면(한 달에 한번 정도) 유저가 새로운 패스워드로 변경하도록 할 수 있다.

14.3.2 root 로 실행되는 서버와 SUID/SGID 바이너리 보안 강화

신중한 시스템 관리자는 많지도 적지도 않게 오직 필요한 서버만 운용한다. 추가적인 서버는 종종 상당한 버그를 가지고 있다. 예를 들면 예전 버전의 **imapd** 나 **popper** 를 운용하는 것은 모든 사람들에게 완벽한 root 티켓을 주는 것과 같다. 주의 깊게 체크하지 않은 서버는 절대 운용하지 않는다. 많은 서버는 root 로 작동하지 않아도 된다. 예를 들어 **ntalk**, **comsat** 와 **finger** 데몬은 특수한 유저 *sandboxes* 로 운용할 수 있다. sandbox 는 완벽하지 않아서 수많은 문제를 수반하겠지만 양파 껍질처럼 안전한 접근방식을 가지고 있다: 누군가 sandbox 로 운용 중인 서버를 깰 수 있다면 그들은 sandbox 만 파괴할 수 있다. 더 높은 층에 대한 공격은 성공하기 어렵기 때문에 공격자의 성공은 보잘것 없어진다. root 보안 취약점은 사실상 기본적인 서버를 포함해서 root 로 운용중인 모든 서버에서 역사적으로 발견되어 왔다. **telnetd**, **rshd** 나 **rlogind** 를 통해 로그인할 수 없고 **sshd** 로만 사람들이 로그인할 수 있다면 이들 서비스를 종료시킨다.

FreeBSD 는 이제 기본적으로 **ntalkd**, **comsat** 과 **finger** 를 sandbox 에서 운용한다. sandbox 로 운용할 다른 서버는 아마 named(8)가 될 것이다. /etc/defaults/rc.conf 에는 **named** 를 sandbox 로 운용하기 위한 인수가 주석 처리되어 있다. 새로운 시스템을 설치하였거나 기존 시스템을 업그레이드 함에 따라 이렇게 sandboxes 로 사용되는 특수 유저 계정이 설치되지 않았을 것이다. 신중한 시스템 관리자는 가능하면 서버에 sandboxes 를 사용하려고 할 것이다.

일반적으로 sandboxes 로 운용되지 않는 다른 서버들이 있다: **sendmail**, **popper**, **imapd**, **ftpd** 와 그 외 다른 서버들이다. 이들 서버 중 어떤 서버를 설치하는 것은 시스템 관리자가 일반적으로 하는 작업보다 더 많은 작업이 필요하다. 이러한 서버는 root 로 운용해야 되고 보안상 문제가 될만한 곳을 찾는 다른 메커니즘을 사용해야 될 것이다.

시스템에서 가능성이 가장 큰 다른 root 보안 취약점은 시스템에 설치되어 있는 suid-root 와 sgid 바이너리이다. 대부분의 **rlogin** 처럼 이들 바이너리는 /bin, /sbin, /usr/bin 또는 /usr/sbin 에 존재한다. 반면 100% 안전하지 안지만 시스템에 기본적으로 설치된 suid 와 sgid 바이너리는 신뢰할만하다. 아직도 root 보안 취약점이 한번씩 이들 바이너리에서 발견

되고 있다. xterm 을(전형적인 suid) 공격하기 쉽게 만드는 root 보안 취약점이 1998 년에 *Xlib* 에서 발견되었다. 편안함을 추구하기 보다 안전이 중요하기 때문에 신중한 시스템 관리자는 suid 바이너리를 스텝들만 실행하고 스텝들만 접근할 수 있도록 제한하여 nobody 가 사용하는 suid 바이너리는 제거(chmod 000)한다. 화면에 디스플레이가 필요가 없는 서버는 보통 xterm 바이너리가 필요 없다. sgid 바이너리는 대부분 위험하다. 침입자가 sgid-kmem 바이너리를 깰 수 있다면 침입자는 /dev/keme 을 읽을 수 있기 때문에 암호화된 패스워드 파일을 읽어서 잠재적으로 계정 패스워드를 획득할 수 있다. 다른 방법으로 kmem 을 파괴할 수 있는 침입자는 안전한 방법으로 로그인하는 유저가 사용하는 pty 를 포함하여 pty 를 통해 입력되는 모든 키 입력을 모니터 할 수 있다. tty 그룹을 깨뜨린 침입자는 대부분의 유저 tty 를 사용할 수 있다. 유저가 터미널 프로그램이나 키보드 시뮬레이션으로 에뮬레이터를 사용한다면 침입자는 그 유저처럼 유저 터미널의 명령을 보여 줄 수 있는 데이터 스트림(stream) 발생기를 만들 수 있다.

14.3.3 유저 계정 보호

유저 계정은 일반적으로 보호하기 매우 힘들다. 스텝에 엄격한 접근 제한을 강요하고 그들의 패스워드를 "*"로 할 수 있지만 일반적인 유저 계정에 이렇게 하는 것은 불가능하다. 충분한 제어를 하고 있다면 유저 계정을 적절하게 보호할 수 있을 것이다. 그렇지 않다면 단순히 이들 계정을 잠도 자지 못하고 모니터링 해야 된다. 유저 계정에 ssh 와 kerberos 를 사용하는 것은 추가적인 관리와 기술 지원이 필요하기 때문에 더욱 문제가 되지만 아직은 암호화된 패스워드 파일과 비교하면 매우 좋은 솔루션이다.

14.3.4 패스워드 파일 보호

유일하게 완벽한 방법은 가능한 많은 패스워드를 *로 하고 이들 계정 접근에 ssh 나 kerberos 를 사용한다. 그리고 암호화된 패스워드 파일(/etc/spwd.db)을 root 만 읽을 수 있고 공격자가 root 의 쓰기 권한을 획득할 수 없더라도 침입자가 이 파일의 읽기 권한을 획득하는 것도 가능하다.

보안 스크립트는 패스워드 파일이(아래의 파일 무결성 체크를 본다) 변경되었는지 항상 체크하고 보고해야 된다.

14.3.5 커널 Core, Raw 장치와 파일시스템 보안

공격자가 root 를 깼다면 원하는 모든 것을 할 수 있지만 특별한 방법이 있다. 예를 들면 대부분의 현대 커널에는 패킷 스니핑(sniffing) 장치 드라이버라는 것이 커널에 내장되어 있다. FreeBSD 에서는 bpf 장치라고 부른다. 침입자는 보통 획득하려는 머신에 패킷 스니퍼를 실행한다. 대부분의 시스템은 bpf 장치를 컴파일 할 필요가 없기 때문에 침입자에게 패킷 스니퍼를 사용할 수 있는 기회를 줄 필요가 없다.

그러나 bpf 장치를 꺼 두었더라도 /dev/mem 과 /dev/kmem 을 가지고 있기 때문에 신경써야 된다. 문제는 침입자가 raw 디스크 장치에 쓰기를 할 수 있다는 것이다. 또한 모듈 로더라고(kldload(8)) 부르는 다른 커널 기능도 있다. 고난도의 침입자는 운용 중인 커널에 자신만의 bpf 장치나 다른 스니핑(sniffing) 장치를 설치하기 위해 KLD 모듈을 사용할 수 있다. 이러한 문제를 피하기 위해 최소한 보안 레벨 1 이상의 높은 보안 레벨에서 커널을 운용해야 된다. 보안 레벨은 *kern.securelevel* 변수에 sysctl 로 설정할 수 있다. 보안 레벨을 1 로 설정하면 raw 장치에 쓰기 접근이 거부되고 *schg* 같은 특수 chflags 플래그가 강제로 설정된다. 또한 중요한 시작 바이너리 디렉터리와 스크립트 파일에 *schg* 플래그를 설정한다; 보안 레벨이 설정된 채로 모든 것이 실행된다. 너무 심한 보안 레벨이 설정되면 시스템을 업그레이드할 때 더욱 어려울 것이다. 높은 보안 레벨에서 시스템을 운용하더라도 모든 시스템 파일과 디렉터리에 *schg* 플래그를 설정하지 않는다. 다른 방법은 단순히 /와 /usr 를 읽기 전용으로 마운트하는 것이고 너무 엄격한 보호는 침입자의 중요한 흔적을 찾지 못하게 할 수 있다.

14.3.6 파일, 바이너리, 설정파일 등의 무결성 체크

일단 침입이 발생하면 심각해지기 전에 주요 시스템구성이나 제어파일들만 보호할 수 있다. 예를 들면 chflags 로 /와 /usr 의 대부분의 파일에 *schg* 비트를 설정하는 것은 역효과를 초래 할 것이다. 왜냐하면 파일을 보호하는 동안 탐지창도 닫기 때문이다. 다 계층 보안의 마지막 층이자 가장 중요한 것은 탐지다. 침입을 탐지하지 못했다면 보안계층의 나머지는 별로 유용하지 않다(또는 안전 불감증으로 더 나쁠 것이다). 양파 보안계층의 반은 공격자를 막기보단 속도를 줄이는 것이고 그가 움직이는 만큼 잡을 수 있는 탐지기회를 주기 위해서다.

침입을 탐지하기 위해 가장 좋은 방법은 변경되거나 실수로 만들어진 또는 예상하지 못한 파일을 찾는 것이다. 변경된 파일을 찾는 가장 좋은 방법은 접근이 제한된(가끔 중앙화 된) 다른 시스템에서다. 추가적으로 보안이 강화되어 제한된 시스템에 보안 스크립트를 추가하

는 것은 가능한 공격을 대부분 차단하기 때문에 중요하다. 최대의 효과를 얻으려면 중요한 머신에 접근할 때 머신의 NFS 를 읽기 전용으로 공유하거나 접근이 제한된 머신에서 다른 머신에 ssh 로 접근할 수 있도록 ssh 키 쌍을 설정해야 된다. 이것이 네트워크 트래픽을 제외하고 NFS 가 최소한으로 노출되는 방법이다. 또한 사실상 탐색되지 않는 각 클라이언트 박스에서 파일시스템을 모니터 할 수 있다. 접근이 제한된 서버가 스위치를 통하여 클라이언트 머신에 연결되어 있다면 NFS 를 사용하는 것이 가끔 더 좋은 선택이 된다. 접근이 제한된 서버가 허브를 통하여 클라이언트와 연결되어있거나 몇 개의 라우팅 계층을 통하여 연결된다면 NFS 를 사용하는 것은 매우 보안에 취약하기 때문에 ssh 를 이용하는 것이 더 좋은 선택일 것이다.

접근이 제한된 머신이 있다면 모니터 하기 위해 클라이언트 시스템에 최소한 읽기 권한이 필요하고 실제 모니터링은 스크립트를 작성해야 된다. 주어진 NFS 마운트에서 find(1)와 md(1) 같은 단순한 시스템 유틸리티로 스크립트를 작성할 수 있다. 클라이언트 머신의 파일을 md5 로 체크하는 것은 최소 하루에 한번 그리고 /etc 와 /usr/local/etc 같은 곳에서 찾을 수 있는 제어 파일테스트는 더욱 자주하는 것이 좋다. 의심스러운 것이 발견되었고 접근이 제한된 머신의 md5 기반 관련정보를 믿을 수 있다면 시스템 관리자가 이것을 체크했을 때 긴장하게 될 것이다. 좋은 보안 스크립트는 적절하지 않은 suid 바이너리와 /, /usr 같은 시스템 파티션에서 추가되거나 삭제된 파일을 체크하는 것이다.

NFS 가 아닌 ssh 를 사용할 때 보안 스크립트를 작성하는 것이 더 어렵다. 기본적으로 이들 보안 스크립트를 실행해서 결과를 보여주고 안전하게 이들 스크립트를 사용하려면 scp 바이너리가(find 같은) 클라이언트 머신에 필요하다. 클라이언트 머신의 **ssh** 클라이언트가 이미 공격 당했을 수도 있다. 취약한 연결에 **ssh** 를 사용해야 되겠지만 역시 쉬운 일은 아니다.

좋은 보안 스크립트는 유저와 스텝 멤버의 접근 설정파일의 변화를 체크하는 것이다: .rhosts, .shosts, .ssh/authorized_keys 와 더 많은 파일은 *MD5* 체크 범위를 벗어나면 실패할 것이다.

거대한 유저 디스크 공간을 가지고 있다면 이들 파티션의 모든 파일까지 체크하기에는 너무 많은 시간이 필요할 것이다. 이 경우 suid 바이너리와 장치를 이들 파티션에 마운트하지 못하도록 마운트 플래그를 설정하는 것도 좋은 생각이다. *nodev* 와 *nosuid* 옵션(mount(8))을 눈 여겨 보아야 된다. 어쨌든 최소 일주일에 한번씩 스캔 해야 된다. 왜냐하면 이 계층의 목적은 침입할 수 있는지 침입할 수 없는지 효과적으로 탐지하는 것이기 때문이다.

프로세스 계정(accton(8)을 본다)은 침입이 있은 후 분석 메커니즘을 돕는 운영체제의 비교

적 낮은 오버헤드 기능이다. 침입이 발생한 후 파일이 손상되지 않았다면 침입자가 실제로 어떻게 시스템에 침입했는지 추적하는데 특히 유용하다.

마지막으로 보안 스크립트는 로그파일을 처리해야 되고 로그는 가능한 보호된 상태에서 생성해야 한다. 이러한 목적에 원격 syslog 가 매우 유용할 것이다. 침입자는 그의 자취를 없애려고 하고 로그파일은 시스템 관리자가 최초 침입 시간과 방법을 추적하는데 중요하다. 로그파일을 영구적으로 기록하는 방법은 시스템 콘솔을 시리얼 포트로 운용하고 안전한 머신의 모니터링 콘솔을 통해 영구적으로 정보를 모은다.

14.3.7 편집증

약간의 불신은 절대 해가 되지 않는다. 일반적으로 시스템 관리자는 편리함에 영향을 미치지 않는다면 개수에 관계없이 보안관련 사항을 추가할 수 있으며 편리함에 영향을 미치는 것과 동시에 보안적인 요소가 증가하는 것도 추가할 수 있다. 보안 관리자는 이러한 것을 섞어서 사용해야 된다. 여기서 알려준 방법을 사용한다면 여러분의 방법은 이 문서를 읽은 공격자에게 알려지게 된다.

14.3.8 서비스 거부 공격

이번 섹션은 서비스 거부 공격에 대해 다룬다. DoS 공격은 전형적인 페킷 공격이다.
현재 페킷 속임수로 네트워크를 소모시키는 공격을 방어할만한 것은 없고 보통 서버가 공격으로 다운되지 않도록 피해를 감소시킬 수는 있다.

① 서버 forks 제한

② 공격 수단의 제한(ICMP 응답 공격과 핑 브로드케스트 등)

③ 커널 라우트 케시

보통 DoS 공격은 머신이 죽을 때까지 서버 프로세스 및 파일기술자(file descriptor: 흔히 파일핸들(file handle)이라고 부르며 유닉스 운영체제는 프로세스가 사용할 수 있는 파일기술자 개수를 제한한다)와 메모리를 소비하게 한다. inetd(inetd(8))는 이런 공격을 제한하는 몇 가지 옵션을 가지고 있지만 머신이 다운되는 것을 막을 수 없고 공격으로부터 서비스 중단을 막는 것도 보통 불가능하다. inetd 매뉴얼 페이지를 주의 깊게 읽고 *-c*, *-C* 와 *-R* 옵

션을 눈 여겨본다. init *-C* 옵션은 변조된 IP 공격을 피할 수 있기 때문에 보통 이런 옵션을 조합하여 사용한다. 어떤 단독 서버는 자체적으로 fork 제한 매개변수를 가지고 있다.

Sendmail 은 sendmail 의 로드제한 옵션을 사용하는 것보다 더 효과적으로 로드를 줄여주는 *-OMaxDaemonChildren* 옵션을 가지고 있다. sendmail 을 실행할 때 예상한 부하를 충분히 제어하도록 *MaxDaemonChildren* 매개변수를 지정하고 컴퓨터가 제어하지 못하도록 너무 높게 지정하지 않는다. sendmail 을 큐 모드에서(*-ODeliveryMode=queued*) 운용하고 데몬을(sendmail -bd) 큐의 운용에서(sendmail -q15m) 분리하여 실행하는 것도 현명한 생각이다. 실시간 배달을 원한다면 *-q1m* 처럼 더욱 낮은 간격으로 큐를 운용할 수 있지만 sendmail 의 단계별 실패를 막기 위해 *MaxDaemonchildren* 옵션을 적당한 값으로 지정한다.

Syslogd 는 직접 공격당 할 수 있기 때문에 가능하면 언제나 *-s* 옵션을 이용하기를 강력히 권장하고 그렇지 않으면 *-a* 옵션을 사용한다.

또한 직접 공격 받을 수 있는 tcpwrapper 의 역방향 identd 같은 역방향 연결 서비스도 (connect-back services) 아주 조심한다. 보통 이런 이유로 tcpwrappers 의 역방향 ident 기능을 사용하지 않을 것이다.

> Tcp wrapper: TCP 를 기반으로 한 네트워크 서비스(finger, ftp, telnet 등) 요청을 받아 그 서비스를 실행하기 전에 요청한 호스트에 대해 보안상으로 필요한 검사를 해서 서비스가 실행되기 이전에 공격을 막을 수 있도록 해주는 프로그램이다.

라우터 끝에 방화벽을 두어 외부 접근에서 내부 서비스를 보호하는 것은 아주 좋은 생각이다. 이 아이디어는 LAN 밖에서의 공격을 방지하지만 네트워크 기반의 root 획득으로부터 내부 서비스를 보호하기에는 적합하지 않다. 항상 방화벽을 배타적으로 설정한다; 예를 들어 "방화벽은 포트 A, B, C, D 와 M-Z 를 제외한 모든 포트를 열어둔다". 이 방법으로 named(존의 메인 서버라면), ntalkd, sendmail 과 인터넷으로 접근할 수 있는 다른 서비스처럼 특정 서비스를 제외한 낮은 포트(1024 안의 포트)를 열어줄 수 있다. 포용적이고 호의적으로 방화벽을 설정하려면 몇 개의 서비스를 막는 것을 잊거나 새로운 내부 서비스를 추가하고 방화벽 업데이트를 잊으면 된다. 낮은 포트를 열어주지 않고 유연하게 방화벽을 운용하려면 높은 번호의 포트범위를 계속 열어줄 수 있다. FreeBSD 는 복잡한 방화벽을 쉽게 설정할 수 있도록 *net.inet.ip.portrange* sysctl 을(sysctl -a | fgrep portrange) 이용하여 동적으로 포트범위를 제어할 수 있다. 예를 들어 일반적으로 4000 에서 5000 사이의 첫 번째/마지막 범위와 49152 에서 65535 의 높은 포트범위를 사용하고 방화벽에서(물론 확실히 지정된 인터넷 서비스 포트를 제외하고) 4000 번 아래는 막는다.

일반적인 다른 DoS 공격은 스프링보드(springboard) 공격이라고 한다; 서버를 공격하기 위한 방법으로 서버가 응답하도록 하여 서버나 로컬 네트워크 또는 다른 머신의 부하를 증가시킨다. 이들 중 가장 보편적인 공격은 ICMP 핑 브로드케스트 공격이다. 공격자는 실제로 공격하려는 소스 IP 주소를 핑 패킷에 설정하여 LAN 의 브로드케스트 주소에 보낸다. 외곽의 라우터가 브로드케스트 주소 핑에 대해 stomp 설정이 되어있지 않으면 LAN 은 공격 당하는 머신을 마비시키도록 변조된 소스 주소로 충분한 응답을 생성한다. 특히 공격자가 한 번에 여러 개의 다른 네트워크에서 여러 개의 다른 브로드케스트에 같은 트릭을 사용한다면 공격하려는 머신을 마비시킬 수 있다. 백 개 이상의 브로드케스트와 20Mbit 공격이 적당하다. 두 번째로 일반적인 스프링보드 공격은 ICMP 에러 리포팅 시스템이다. ICMP 에러 응답으로 생성되는 패킷을 짜맞춰서 공격자는 서버의 입력 네트워크를 포화상태로 만들 수 있기 때문에 ICMP 응답으로 나가는 네트워크도 포화상태가 된다. 이런 종류의 공격은 mbuf 의 (네트워크 버퍼에 사용되는 구조체 이름으로 초기 크기는 kern.maxusers 에 의해 결정된다) 고갈로 서버가 다운될 수 있다. 특히 서버가 ICMP 응답을 처리하지 못한다면 쉽게 다운될 수 있다. FreeBSD 커널은 이런 종류의 피해를 줄이기 위해 *ICMP_BANDLIM* 이라는 새로운 커널 컴파일 옵션을 가지고 있다. 마지막으로 주요 스프링보드 공격은 udp 에코 서비스처럼 특정 내부 inetd 와 관련된 것이다. 서버 A 와 B 가 같은 LAN 에 있으면 공격자는 단순히 소스 주소를 서버 A 의 에코 포트에 그리고 목적지 주소를 서버 B 의 에코 포트로 설정하여 UDP 패킷을 변조한다. 두 서버는 이 패킷을 각자에게 되돌려 준다. 공격자는 이 방법에서 간단히 약간의 패킷을 추가하여 두 서버와 그 서버들의 LAN 을 과부화 상태로 만들 수 있다. 비슷한 문제가 내부 **chargen** 포트에서 나타난다. 유능한 시스템 관리자는 이런 inetd 내부 테스트 서비스를 끌 것이다.

패킷 변조 공격은 커널 라우트 케시의 부하를 높일 때도 사용된다. *net.inet.ip.rtexpire*, *rtminexpire* 와 *rtmaxcache* sysctl 매개변수를 참고한다. 무작위로 소스 IP 를 사용하는 패킷 거부공격은 라우트 테이블에서 임시로 케시된 라우트를 커널이 생성하도록 하고 **netstat -rna | fgrep W3** 로 볼 수 있다. 이런 라우트는 전형적으로 1600 초안에 타임아웃 된다. 커널이 케시된 라우트 테이블이 너무 크다는 것을 발견했다면 rtexpire 로 동적으로 감소시키지만 rtminexpire 보다 절대 적어지지 않는다. 여기 두 가지 문제가 있다:

① 커널은 가벼운 부하상태인 서버가 갑자기 공격 당했을 때 빠르게 대응하지 못한다.

② 커널이 일관된 공격에서 살아남도록 *rtminexpire* 를 충분히 낮춰야 한다.

서버가 T3 나 더 좋은 인터넷에 연결되어있다면 직접 sysctl(8)로 *rtexpire* 와 *rtminexpire* 를

설정하는 것이 좋을 수 있다. 매개변수를 절대 0 으로(머신이 다운되기를 원하지 않는다면) 설정하지 않는다. 두 매개변수를 2 로 설정하면 공격으로부터 라우트 테이블을 보호하기에 충분할 것이다.

14.3.9 kerberos 와 SSH 접근 문제

Kerberos 와 SSH 를 사용하려고 하면 Kerberos 와 ssh 를 사용한다고 알려야 되는 몇 가지 문제가 있다. Kerberos V 는 훌륭한 인증 프로토콜이지만 kerberized 된 telnet 과 rlogin 어플리케이션에서 바이너리 스트림과 맞지 않는 버그가 있다. 또한 기본 Kerberos 는 *-x* 옵션을 사용하지 않으면 세션을 암호화하지 않는다. ssh 는 기본적으로 모든 것을 암호화한다.

기본적으로 암호 키를 포워드하는 것을 제외하고 ssh 를 사용한 작업은 상당히 안전하다. 이 의미는 나머지 시스템에 접근할 수 있도록 암호 키를 가지고 있는 안전한 워크스테이션이 있고 불안전한 머신에 ssh 를 사용한다면 암호 키는 유용하다. 실제 키는 노출되지 않지만 로그인이 지속되는 동안 ssh 는 포워딩 포트를 설치하고 공격자가 불안전한 머신에서 root 를 획득했다면 다른 머신에 접속하기 위해 키를 사용하려고 포트를 사용할 수 있다.

가능하다면 스텝(staff) 로그인에 ssh 와 Kerberos 를 같이 사용하도록 권장한다. ssh 가 Kerberos 를 지원하도록 컴파일 할 수 있다. 이 방법은 Kerberos 로 패스워드를 보호하는 동안 잠재적으로 ssh 키가 노출되는 것을 줄인다. ssh 키는 안전한 머신에서(Kerberos 는 이 일에 적당하지 않다) 자동화된 태스크에 사용해야 된다. 또한 ssh 설정에서 키 포워딩을 끄거나 키를 특정 머신에서 로그인할 때만 사용하도록 authorized_keys 파일에서 *from=IP/DOMAIN* 옵션을 사용하기를 권장한다.

14.4 DES, MD5 와 암호화

유닉스 시스템의 모든 유저는 계정에 맞는 패스워드를 가지고 있다. 이런 패스워드는 유저와 실제 운영체제만 알고 있어야 한다. 이런 패스워드를 비밀리에 유지하기 위해 쉽게 암호화만 될 수 있고 복호화는 될 수 없는 일방적인 해시(hash)로 암호화된다. 바꿔 말하면 우리가 방금 설명한 내용은 정확히 진실은 아니다: 운영 체제는 실제로 패스워드를 알지 못하고 단지 패스워드로부터 암호화된 것을 알고 있다. 평범한 텍스트 패스워드를 아는 유일한 방법은 가능한 패스워드를 강제로 검색하는 것이다.

불행히 패스워드를 암호화하는 유일하게 안전한 방법은 유닉스가 데이터 암호화 표준인 DES 기반일 때였다. DES 소스 코드를 미국 이외로 수출할 수 없었기 때문에 미국에 살고 있는 사람들에게는 문제가 되지 않았지만 FreeBSD 는 미국 수출법과 DES 를 계속 사용하는 다른 모든 유닉스와 호환을 유지하는 방법을 찾아야 했다.

해결책은 암호화 라이브러리를 나누어서 미국 유저는 DES 라이브러리를 설치하여 DES 를 사용할 수 있지만 다국적 유저는 수출할 수 있는 암호화 방법을 가지고 있다. 여기서 어떻게 FreeBSD 가 기본 암호화 기법으로 MD5 를 사용하게 되었는지 설명한다. MD5 가 DES 보다 좀더 안전하다고 생각되었기 때문에 DES 를 설치하는 것은 주로 호환성 때문이다.

14.4.1 보안 메커니즘 확인

FreeBSD 4.4 이 전에 libcrypt.a 는 암호화에 사용되는 라이브러리를 심볼릭으로 링크하였다. FreeBSD 4.4 에서는 libcrypt.a 가 인증해시 라이브러리에 설정할 수 있는 패스워드를 제공한다. 현재 이 라이브러리는 DES, MD5 와 Blowfish 해시기능을 지원한다. 기본적으로 FreeBSD 는 패스워드 암호화에 MD5 를 사용한다.

어떤 암호화 기법이 FreeBSD 에 설정되고 사용되는지 확인하는 방법은 상당히 쉽다. /etc/master.passwd 파일에서 암호화된 패스워드를 확인하는 것도 하나의 방법이다. MD5 해시로 암호화된 패스워드는 DES 해시로 암호화된 것보다 길고 1로 시작된다. 2로 시작하는 패스워드는 Blowfish 해시기능으로 암호화된 것이다. DES 패스워드 문자열은 확인할 수 있는 어떠한 특수 문자도 가지고 있지 안지만 MD5 패스워드보다 짧고 $ 문자를 포함하지 않은 64 개 알파벳이 코드 되어있어서 달러 문자로($) 시작하지 않고 짧은 문자열은 대부분 DES 패스워드다.

새로운 패스워드에 사용하는 패스워드 포맷은 /etc/login.ocnf 에서 "des", "md5" 또는 "blf" 의 값을 갖는 “passwd_formt” 로그인 설정으로 제어된다. 로그인 설정에 대한 더 많은 정보는 login.conf(5) 매뉴얼 페이지를 본다.

14.5. 일회용 패스워드(One-Time Password)

S/Key 는 단 방향 해시기능 기반의 일회용 패스워드다. FreeBSD 는 호환성을 위해 MD4 해시를 사용하지만 다른 시스템은 MD5 와 DES-MAC 를 이용한다. S/Key 는 FreeBSD 1.1.5

부터 FreeBSD 기본 시스템에 편입되었고 수많은 운영체제에 채택되고 있다. S/Key 는 벨 통신 연구소의 등록상표다.

FreeBSD 버전 5.0 부터 S/Key 가 OPIE(모든 것에서 일회용 패스워드)와 기능적으로 동일하게 되었다. OPIE 는 기본적으로 MD5 해시를 사용한다.

아래에서 3 가지 종류의 패스워드를 설명한다. 첫째는 보통 유닉스 스타일 또는 Kerberos 패스워드다; 우리는 이것을 유닉스 패스워드라고 부른다. 두 번째 종류는 S/Key 키 프로그램이나 OPIE opiekey(1) 프로그램이 생성하고 keyinit 또는 opiepaswd(1) 프로그램과 로그인 프롬프트가 인증한다; 우리가 “일회용 패스워드(one-time 패스워드)"라고 부르는 일회용 패스워드다. 마지막 패스워드는 일회용 패스워드를 만들 때 사용되는 key/opiekey 프로그램을 사용하는 비밀 패스워드다; 우리는 이것을 "비밀 패스워드" 또는 그냥 "패스워드"라고 부른다.

비밀 패스워드는 유닉스 패스워드와 아무런 관련이 없다; 두 개를 동일하게 설정할 수 있지만 권장하지 않는다. S/Key 와 OPIE 비밀 패스워드는 예전 유닉스 패스워드(FreeBSD 표준 로그인 패스워드는 128 개의 문자 길이를 가지고 있을 것이다)처럼 8 개 문자의 한계가 없이 원하는 길이만큼 사용할 수 있다. 6 이나 7 개 문자의 긴 어구로 된 패스워드가 일반적이다. 대부분 S/Key 나 OPIE 시스템은 유닉스 패스워드와 완벽하게 독립적으로 운영된다.

패스워드와 달리 S/Key 와 OPIE 에 중요한 두 가지 다른 데이터가 있다. 하나는 두 개의 문자와 5 개의 숫자로 이루어진 "seed" 또는 "key”로 알려진 것이다. 다른 하나는 "반복 카운트"라고 하는 1 에서 100 까지의 번호다. S/Key 는 seed 와 비밀 패스워드를 연결하여 일회용 패스워드를 생성하고 반복 카운트에 지정된 만큼 MD4/MD5 해시에 적용하여 6 개의 짧은 영어단어를 결과로 출력한다. 이 6 개의 영어단어가 일회용 패스워드다. 인증 시스템은 (PAM 이 메인 인증 시스템) 마지막으로 일회용 패스워드를 사용한 흔적을 가지고 유저에 제공된 해시 패스워드가 이전 패스워드와 같다면 유저는 인증에 통과한다. 왜냐하면 성공적으로 사용된 패스워드가 저장되어있다면 여기에 사용된 단 방향 해시는 나중에 일회용 패스워드를 만드는 것이 불가능하기 때문이다; 반복 카운터는 성공적인 각각의 로그인후 유저와 로그인 프로그램을 동기화하기 위해 감소된다. 반복 카운트가 감소되어 1 이 되면 S/Key 와 OPIE 는 다시 초기화된다.

아래에서 설명할 각 시스템과 관련된 3 개의 프로그램이 있다. key 와 opiekey 프로그램은 반복 카운트, seed, 비밀 패스워드 그리고 일회용 패스워드를 생성하거나 일회용 패스워드의 연속적인 리스트를 허용한다. keyinit 와 opiepasswd 프로그램은 각각 S/Key 와 OPIE 초기화에 쓰이고 패스워드, 반복 카운트 또는 seed 변경에 사용된다; 이들은 비밀 패스워드

구문이나 반복 카운트, seed 및 일회용 패스워드 중 하나를 사용한다. keyinfo 와 opieinfo 프로그램은 관련된 증명파일을 확인해서(/etc/skeykeys 또는 /etc/opiekeys) 유저의 현재 반복 카운트와 seed 를 출력한다.

우리가 설명할 네 가지 종류의 사용법이 있다. 첫째는 안전한 연결을 통하여 처음으로 일회용 패스워드를 설정하거나, 패스워드나 seed 를 변경하는데 keyinit 또는 opiepasswd 를 사용하는 것이다. 두 번째 사용법은 안전한 연결의 key 나 opiekey 를 사용하여 안전하지 않은 연결에 keyinit 또는 opiepasswd 를 사용하는 것과 같다. 세 번째는 안전하지 않은 연결에서 key/opiekey 를 사용하는 것이다. 네 번째는 안전하지 않은 연결로 어떤 곳에 연결할 때, 적어둘 수 있거나 출력할 수 있는 여러 개의 키를 생성하기 위해 key 나 opieky 를 사용하는 것이다.

14.5.1 보안 연결에서(ssh 처럼 암호화된 연결) 초기화하기

처음 S/Key 를 초기화하려면 안전한 연결로 로그인이 되어있는 동안 패스워드를 바꾸거나 seed 를 변경하고(예: 머신의 콘솔이나 ssh 를 통해) 아니면 직접 로그인 되어있는 동안 아무런 인수 없이 keyinit 명령을 사용한다:

```
% keyinit
Adding unfurl:
Reminder - Only use this method if you are directly connected.
If you are using telnet or rlogin exit with no password and use keyinit -s.
Enter secret password:                    ❶
Again secret password:

ID unfurl s/key is 99 to17757             ❸
DEFY CLUB PRO NASH LACE SOFT
```

OPIE 에서는 opiepasswd 를 대신 사용한다:

```
% opiepasswd -c
[grimreaper] ~ $ opiepasswd -f -c
Adding unfurl:
Only use this method from the console; NEVER from remote. If you are using
telnet, xterm, or a dial-in, type ^C now or exit with no password.
Then run opiepasswd without the -c parameter.
Using MD5 to compute responses.
Enter new secret pass phrase:      ❷
Again new secret pass phrase:
ID unfurl OTP key is 499 to4268    ❸
MOS MALL GOAT ARM AVID COED
```

❷ Enter new secret pass phrase: 또는 ❶ Enter secret password: 프롬프트에 패스워드를 입력한다. 이것은 로그인할 때 사용하는 패스워드가 아니고 일회용 로그인 키를 생성하기 위해 사용된다. ❸ ID 라인에는 특별한 인스턴스의 매개변수를 보여준다; 로그인 이름, 반복 카운트와 seed. 로그인할 때 시스템이 이들 매개변수를 기억하기 때문에 따로 기억할 필요는 없다. 마지막 라인에는 이들 매개변수 및 비밀 패스워드와 일치하는 특별한 일회용 패스워드를 입력한다; 즉시 다시 로그인 했다면 이 일회용 패스워드를 사용할 것이다.

14.5.2 안전하지 않은 연결에서(telnet 처럼 암호화되지 않은 연결) 초기화하기

안전하지 않은 연결에서 초기화하거나 비밀 패스워드를 변경하려면 key 나 opiekey 를 실행할 수 있는 공간을 위해 안전한 연결이 필요하다; 이것은 Macintosh 의 데스크 액세서리 형식이거나 신뢰된 머신의 쉘 프롬프트일 것이다. 또한 반복 카운트를(100 정도의 값이 좋을 것이다) 만들어야 되고 자신만의 seed 를 구성하거나 무작위로 만들어진 것을 사용할 것이다. 안전하지 않은 연결에서(초기화하려는 머신) keyinit -s 명령을 사용한다:

```
% keyinit -s
Updating unfurl:
Old key: to17758
Reminder you need the 6 English words from the key command.
Enter sequence count from 1 to 9999: 100
Enter new key [default to17759]:
```

```
s/key 100 to 17759
s/key access password:
s/key access password:CURE MIKE BANE HIM RACY GORE
```

OPIE 에는 opiepasswd 를 사용해야 된다:

```
% opiepasswd

Updating unfurl:
You need the response from an OTP generator.
Old secret pass phrase:
        otp-md5 498 to4268 ext
        Response: GAME GAG WELT OUT DOWN CHAT
New secret pass phrase:
        otp-md5 499 to4269
        Response: LINE PAP MILK NELL BUOY TROY

ID mark OTP key is 499 gr4269
LINE PAP MILK NELL BUOY TROY
```

기본 seed 를(keyinit 프로그램을 혼동되게 key 라고 부르는) 사용하려면 Enter 를 누른다. 그리고 접근 패스워드를 입력하기 전에 안전한 연결이나 S/Key 데스크 액세서리로 이동해서 같은 매개변수를 입력한다:

```
% key 100 to17759
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password: <비밀번호 입력>
CURE MIKE BANE HIM RACY GORE
```

OPIE 는 다음 명령을 사용한다:

```
% opiekey 498 to4268
Using the MD5 algorithm to compute response.
Reminder: Don't use opiekey from telnet or dial-in sessions.
Enter secret pass phrase:
GAME GAG WELT OUT DOWN CHAT
```

이제 안전하지 않은 연결로 되돌아와서 관련된 프로그램이 생성한 일회용 패스워드를 복사한다.

14.5.3 일회용 패스워드 하나 생성하기

S/Key 나 OPIE 를 초기화하면 로그인할 때 다음과 같은 프롬프트가 나타난다:

```
% telnet example.com
Trying 10.0.0.1...
Connected to example.com
Escape character is '^]'.

FreeBSD/i386 (example.com) (ttypa)

login: <username>
s/key 97 fw13894
Password:
```

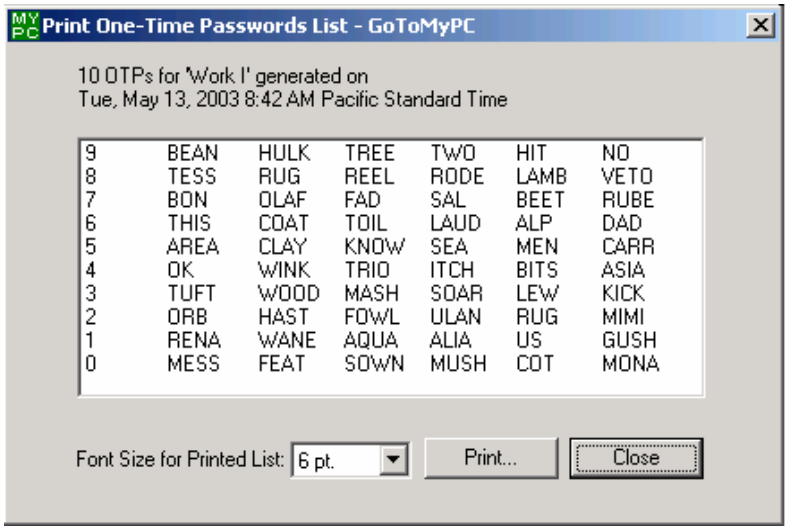
OPIE 는 다음과 같이 나타난다:

```
% telnet example.com
Trying 10.0.0.1...
Connected to example.com
Escape character is '^]'.

FreeBSD/i386 (example.com) (ttypa)

login: <username>
otp-md5 498 gr4269 ext
Password:
```

S/Key 와 OPIE 프롬프트는 유용한 기능을(여기서는 설명하지 않는다) 가지고 있다: 패스워드 프롬프트에서 Return 을 누르면 프롬프트는 에코로 다시 보여주기 때문에 무엇을 입력했는지 볼 수 있다. 이것은 직접 패스워드를 입력했다면 printout 처럼 아주 유용할 수 있다.

여기서 이 로그인 프롬프트에 대한 응답으로 일회용 패스워드를 생성해야 된다. 신뢰된 시스템에서 key 나 opiekey 를 실행하면 된다(DOS, 윈도우와 MacOS 에서도 다양한 버전이 있다).

윈도우 버전 일회용 패스워드 생성기



여기서 반복 카운트와 seed 가 명령어 라인 옵션으로 필요하다. 로그인 프롬프트에서 잘라내서 로그인 하려는 머신에 붙일 수 있다.

신뢰할 수 있는 연결이나 시스템에서 다음 명령을 실행한다:

```
% key 97 fw13894
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password:
WELD LIP ACTS ENDS ME HAAG
```

OPIE 는 다음 명령을 사용한다:

```
% opiekey 498 to4268
Using the MD5 algorithm to compute response.
Reminder: Don't use opiekey from telnet or dial-in sessions.
Enter secret pass phrase:
GAME GAG WELT OUT DOWN CHAT
```

이제 계속 로그인할 수 있는 일회용 패스워드를 갖게 된다:

```
login: <username>
s/key 97 fw13894
Password: <echo 를 활성화하기 위해 Enter 를 누른다>
s/key 97 fw13894
Password [echo on]: WELD LIP ACTS ENDS ME HAAG
Last login: Tue Mar 21 11:56:41 from 10.0.0.2 ...
```

14.5.4 여러 개의 일회용 패스워드 생성

가끔 신뢰관계가 아닌 머신이나 보안상 안전한 연결에 접근해야 될 때가 있다. 이러한 경우 key 와 opiekey 명령을 사용하여 일회용 패스워드를 미리 생성해서 사용할 수 있다. 예를 들면 다음과 같이 실행한다:

```
% key -n 5 30 zz99999
Reminder - Do not use this program while logged in via telnet or rlogin.
Enter secret password: <secret password>
26: SODA RUDE LEA LIND BUDD SILT
27: JILT SPY DUTY GLOW COWL ROT
28: THEM OW COLA RUNT BONG SCOT
29: COT MASH BARR BRIM NAN FLAG
30: CAN KNEE CAST NAME FOLK BILK
```

OPIE 는 다음 명령을 사용한다:

```
% opiekey -n 5 30 zz99999
Using the MD5 algorithm to compute response.
Reminder: Don't use opiekey from telnet or dial-in sessions.
Enter secret pass phrase: <secret password>
26: JOAN BORE FOSS DES NAY QUIT
27: LATE BIAS SLAY FOLK MUCH TRIG
28: SALT TIN ANTI LOON NEAL USE
29: RIO ODIN GO BYE FURY TIC
30: GREW JIVE SAN GIRD BOIL PHI
```

*-n 5*는 순서대로 5 개 키를 요청하고 *30*은 반복해야 될 마지막 숫자를 지정한다. 이들은 마지막으로 사용된 날짜의 역순으로 출력된다. 의심된다면 결과를 직접 적어둘 수 있고 아니면 잘라내서 lpr 에 붙일 수 있다. 각 라인은 반복 카운트와 일회용 패스워드를 보여준다; 이런 방법을 사용하거나 패스워드를 직접 찾을 것이다.

14.5.5 유닉스 패스워드 사용제한

S/Key 는 호스트 이름, 유저 이름, 터미널 포트 또는 로그인 세션의 IP 주소에 따라 유닉스 패스워드 사용을 제한할 수 있다. 이들 제한은 /etc/skey.access 설정파일에서 찾을 수 있다. skey.access(5) 매뉴얼 페이지에 파일의 전체 포맷에 대한 더 많은 정보가 있고 이 파일을 사용하기 전에 알고 있어야 되는 몇 가지 보안권고도 있다.

/etc/skey.access 파일이(FreeBSD 4.x 시스템에서 기본적인) 없다면 모든 유저는 유닉스 패스워드를 사용할 수 있다. 파일이 있다면 skey.access 파일 문구에 유닉스 패스워드를 사용할 수 있도록 설정되어있는 경우를 제외하고 모든 유저는 S/Key 를 사용해야 된다.

여기 설정구문 중 아주 일반적인 3 가지를 보여주는 샘플 skey.access 설정파일이 있다.

```
permit internet 192.168.0.0 255.255.0.0
permit user fnord
permit port ttyd0
```

첫 번째 라인은(*permit internet*) 유닉스 패스워드를 사용하도록 지정된 값과 IP 소스 주소와 mask 가 일치하는 유저만 허용한다. 이것을 보안 메커니즘에 고려하지 않아도 되지만 승인된 유저들에게 그들이 안전하지 않은 네트워크를 사용하고 있으며 인증을 위해 S/key 를 사용해야 된다는 것을 알려줘야 된다.

여기서는 fnord 인 두 번째 라인(*permit user*)은 지정된 유저 이름이 언제라도 유닉스 패스워드를 사용할 수 있도록 허용한다. 일반적으로 말하면 이 옵션은 dumb 터미널처럼 key 프로그램을 사용하지 못하거나 교육할 수 없는 사람들에게만 사용해야 된다.

세 번째 라인은(*permit port*) 유닉스 패스워드를 사용하도록 지정된 터미널 라인의 모든 유저 로그인을 허용한다; 이것은 전화 접속에 사용될 것이다.

OPIE 도 S/Key 처럼 로그인 세션의 IP 주소에 따라 유닉스 패스워드 사용을 제한할 수 있다. 관련된 파일은 FreeBSD 5.0 와 새로운 시스템에서 기본적으로 가지고 있는 /etc/opieaccess 이다. 이 파일에 대한 더 많은 정보와 사용할 때 알아야 되는 보안권고에 대해 opieaccess(5)를 체크한다.

여기 샘플 opieaccess 파일이 있다:

```
permit 192.168.0.0 255.255.0.0
```

지정된 값과 IP 소스 주소와(스푸핑(spoofing)에 공격 받기 쉬운) mask 가 일치하는 유저만 언제라도 유닉스 패스워드를 사용할 수 있다.

opieaccess 의 모든 룰에 맞지 않으면 기본값은 OPIE 가 아닌 모든 로그인을 거부한다.

14.6 KerberosIV

Kerberos 는 유저들이 보안 서버의 서비스를 통해 직접 인증을 받을 수 있는 네트워크에 추가된 시스템/프로토콜이다. 원격 로그인, 원격 복사, 내부 보안 시스템파일 복사 및 위험성이 높은 서비스를 더 안전하게 정확히 제어할 수 있다.

다음 설명은 FreeBSD 에 배포된 Kerberos 를 어떻게 설정하는지 가이드로 사용할 수 있다. 그러나 완벽한 설명은 관련된 매뉴얼 페이지를 참고한다.

14.6.1 KerberosIV 설치

Kerberos 는 FreeBSD 에 추가적인 컴포넌트다. 이 소프트웨어를 설치하는 가장 쉬운 방법은 FreeBSD 를 처음 설치하는 과정 중 **sysinstall** 에서 *krb4* 나 *krb5* 배포본을 선택하면 된다. 이 배포본은 Kerberos 요소 'eBones'(KerberosIV)이나 'Heimdal'(Kerberos5)를 설치한다. 이런 요소는 미국/캐나다 밖에서 개발되어 미국의 암호화 코드 수출제한이 있던 시기에도 두 나라밖의 시스템 관리자들이 사용할 수 있었기 때문에 포함되었다.

대신 MIT Kerberos 요소는 포트 컬렉션 security/krb5에서 사용할 수 있다.

14.6.2 최초의 데이터베이스 생성

이 사항은 Kerberos 서버에만 해당한다. 첫째 어떠한 예전 Kerberos 데이터베이스도 가지고 있지 않아야 된다. /etc/kerberosIV 로 디렉터리를 변경하고 다음 파일들만 있는지 체크한다:

```
# cd /etc/kerberosIV
# ls
README        krb.conf        krb.realms
```

다른 파일이(principal.*이나 master_key 같은) 있다면 예전 Kerberos 데이터베이스를 삭제하기 위해 kdb_destroy 명령을 사용하거나 Kerberos 가 실행 중이 아니라면 간단히 이들 파일을 삭제하면 된다.

이제 Kerbers 영역을 정의하기 위해 krb.conf 와 krb.realms 파일을 수정한다. 우리의 경우 영역은 *EXAMPLE.COM* 이고 서버는 grunt.example.com 이다. 우리는 다음 krb.conf 파일을 수정하거나 생성했다:

```
# cat krb.conf
❶EXAMPLE.COM
❷EXAMPLE.COM ❸grunt.example.com admin server
CS.BERKELEY.EDU okeeffe.berkeley.edu
ATHENA.MIT.EDU kerberos.mit.edu
ATHENA.MIT.EDU kerberos-1.mit.edu
ATHENA.MIT.EDU kerberos-2.mit.edu
ATHENA.MIT.EDU kerberos-3.mit.edu
LCS.MIT.EDU kerberos.lcs.mit.edu
TELECOM.MIT.EDU bitsy.mit.edu
ARC.NASA.GOV trident.arc.nasa.gov
```

우리의 경우 이곳에 다른 영역은 필요 없다. 이 예제는 머신이 여러 개의 영역을 어떻게 인지하는지 보여준다.

❶번 라인은 이 시스템이 동작하는 영역의 이름이다. 다른 라인은 realm/host 엔트리를 포함하고 있다. 라인의 ❷번 내용은 영역이고 ❸번은 "키 분산 센터"처럼 동작하는 영역의 호

스트다. 단어 *admin server* 다음의 호스트 이름이 의미하는 것은 호스트도 관리 데이터베이스 서버를 제공한다는 것이다. 이들 용어에 대한 자세한 사항은 Kerberos 매뉴얼 페이지를 참고한다.

이제 grunt.example.com 을 *EXAMPLE.COM* 영역에 추가하고 *EXAMPLE.COM* 영역의 .example.com 도메인에 있는 모든 호스트 엔트리를 추가한다. 따라서 krb.realms 파일은 다음과 같이 업데이트될 것이다:

```
# cat krb.realms
grunt.example.com EXAMPLE.COM
.example.com EXAMPLE.COM
.berkeley.edu CS.BERKELEY.EDU
.MIT.EDU ATHENA.MIT.EDU
.mit.edu ATHENA.MIT.EDU
```

역시 다른 영역은 이곳에 필요 없다. 이 파일은 머신이 여러 영역을 어떻게 인지하는지에 대한 예제였다.

첫 번째 라인은 특정 시스템을 영역으로 바꾼다. 나머지 라인은 특정 서브 도메인의 기본 시스템을 영역으로 어떻게 바꾸는지 보여준다.

이제 데이터베이스를 생성할 준비가 되었다. 이것은 Kerberos 서버에서만(또는 키 분산 센터) 실행하면 된다. kdb_init 명령을 다음과 같이 실행한다:

```
# kdb_init
Realm name [default  ATHENA.MIT.EDU ]: EXAMPLE.COM
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.

Enter Kerberos master key:
```

이제 키를 저장해서 로컬 머신의 서버가 키를 로드하도록 해야 된다. kstash 명령을 다음과 같이 사용한다:

```
# kstash

Enter Kerberos master key:

Current Kerberos master key version is 1.

Master key entered. BEWARE!
```

위 명령은 암호화된 마스터 패스워드를 /etc/kerberosIV/master_key 에 저장한다.

14.6.3 모두 실행하기

Kerberos 로 각 시스템을 안전하게 만들기 위해 두 가지 중요한 요소를 데이터베이스에 추가해야 된다. 이들 이름은 *Kpasswd* 와 *rcmd* 이고 두 요소는 각 시스템의 이름을 참조하여 생성된다.

이들 데몬 kpasswd 와 rcmd 는 다른 시스템에서 Kerberos 패스워드를 변경하고 rcp, rlogin 과 rsh 같은 명령을 실행할 수 있도록 한다.

이제 이런 엔트리를 추가해 보자:

```
# kdb_edit
Opening database...

Enter Kerberos master key:

Current Kerberos master key version is 1.

Master key entered.  BEWARE!
Previous or default values are in [brackets] ,
enter return to leave the same, or new value.

Principal name: passwd
Instance: grunt
```

```
<Not found>, Create [y] ? y

Principal: passwd, Instance: grunt, kdc_key_ver: 1
New Password:
Verifying password

New Password:

Random password [y] ? y

Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 2000-01-01 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
Principal name: rcmd
Instance: grunt

<Not found>, Create [y] ?

Principal: rcmd, Instance: grunt, kdc_key_ver: 1
New Password:
Verifying password

New Password:

Random password [y] ?

Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 2000-01-01 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
Principal name:          <---- 아무것도 입력하지 않으면 빠져나간다.
```

14.6.4 서버 파일 생성

이제 각 머신에 정의된 모든 서비스 인스턴스를 추출해야 된다. 추출하기 위해 ext_srvtab 명령을 이용한다. 이 명령은 각 Kerberos 클라이언트의 /etc/kerberosIV 디렉터리로 *안전하게* 복사하거나 옮겨야 되는 파일을 생성한다. 이 파일은 각 서버와 클라이언트에 필요하고 Kerberos 운용에 중요하다.

```
# ext_srvtab grunt
Enter Kerberos master key:

Current Kerberos master key version is 1.

Master key entered. BEWARE!
Generating 'grunt-new-srvtab'....
```

위 명령은 srvtab 로 이름을 바꿔야 되는 임시파일을 생성하기 때문에 모든 서버가 이 파일을 로드할 수 있다. mv(1) 명령으로 이 파일을 최초 시스템으로 이동시킨다:

```
# mv grunt-new-srvtab srvtab
```

이 파일이 클라이언트 시스템에 있고 네트워크가 안전하지 않다고 생각되면 *client*-new-srvtab 를 이동 가능한 미디어에 복사해서 물리적으로 안전하게 이동시킨다. 클라이언트의 /etc/kerberosIV 디렉터리에서 srvtab 로 다시 이름을 바꾸고 퍼미션을 600 으로 변경한다:

```
# mv grumble-new-srvtab srvtab
# chmod 600 srvtab
```

14.6.5 데이터베이스 생성

이제 몇몇 유저 엔트리를 데이터베이스에 추가해야 된다. 첫째로 유저 jane 엔트리를 생성하자. Kdb_edit 명령을 다음과 같이 사용한다:

```
# kdb_edit
Opening database...

Enter Kerberos master key:

Current Kerberos master key version is 1.

Master key entered.  BEWARE!
Previous or default values are in [brackets] ,
enter return to leave the same, or new value.

Principal name: jane
Instance:

<Not found>, Create [y] ? y

Principal: jane, Instance: , kdc_key_ver: 1
New Password:                <---- 보안 패스워드를 입력한다
Verifying password

New Password:                <---- 패스워드를 다시 입력한다
Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 2000-01-01 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ?
Attributes [ 0 ] ?
Edit O.K.
Principal name:          <---- 아무것도 입력하지 않으면 빠져나간다
```

14.6.6 테스트

첫째 Kerberos 데몬을 실행해야 한다. /etc/rc.conf 를 정확히 수정했다면 재 부팅할 때 자동으로 데몬이 실행된다. 이것은 오직 Kerberos 서버에만 필요하다. Kerberos 클라이언트는 /etc/kerberosIV 디렉터리에서 필요한 것을 자동으로 수행한다.

```
# kerberos &
Kerberos server starting
Sleep forever on error
Log file is /var/log/kerberos.log
Current Kerberos master key version is 1.

Master key entered. BEWARE!

Current Kerberos master key version is 1
Local realm: EXAMPLE.COM
# kadmind -n &
KADM Server KADM0.0A initializing
Please do not use 'kill -9' to kill this job, use a
regular kill instead

Current Kerberos master key version is 1.

Master key entered.  BEWARE!
```

이제 위에서 만든 ID jane 의 티켓을 kinit 명령으로 얻을 수 있다:

```
% kinit jane
MIT Project Athena (grunt.example.com)
Kerberos Initialization for "jane"
Password:
```

실제로 인증을 받았다면 체크할 수 있도록 klist 를 사용하여 결과를 리스트 한다:

```
% klist
Ticket file:     /tmp/tkt245
Principal:       jane@EXAMPLE.COM

  Issued           Expires          Principal
Apr 30 11:23:22  Apr 30 19:23:22  krbtgt.EXAMPLE.COM@EXAMPLE.COM
```

kpasswd 데몬이 Kerberos 데이터베이스 인증을 받을 수 있다면 passwd 를 사용해서 패스워드를 바꿀 수 있는지 확인한다:

```
% passwd
realm EXAMPLE.COM
Old password for jane:
New Password for jane:
Verifying password
New Password for jane:
Password changed.
```

14.6.7 su 권한 추가

Kerberos 는 root 권한이 필요한 *각* 유저에게 *각자의* su 패스워드를 할당할 수 있다. 이제 su(1)로 root 인증이 필요한 ID 를 추가할 수 있다. 이것은 주체(principal)와 관련된 root 인스턴스를 가지고 제어된다. kdb_edit 를 사용하여 Kerberos 데이터베이스에 *jane.root* 엔트리를 생성한다:

```
# kdb_edit
Opening database...

Enter Kerberos master key:

Current Kerberos master key version is 1.

Master key entered.  BEWARE!
Previous or default values are in [brackets] ,
enter return to leave the same, or new value.

Principal name: jane
Instance: root
```

```
<Not found>, Create [y] ? y

Principal: jane, Instance: root, kdc_key_ver: 1
New Password:                    <---- 보안 패스워드를 입력한다
Verifying password

New Password:                <---- 패스워드를 다시 입력한다

Principal's new key version = 1
Expiration date (enter yyyy-mm-dd) [ 2000-01-01 ] ?
Max ticket lifetime (*5 minutes) [ 255 ] ? 12 <--- 이 값을 유지한다
Attributes [ 0 ] ?
Edit O.K.
Principal name:                  <---- 아무것도 입력하지 않으면 빠져나간다
```

이제 동작하는지 확인해 보자:

```
# kinit jane.root
MIT Project Athena (grunt.example.com)
Kerberos Initialization for "jane.root"
Password:
```

유저를 root 의 .klogin 파일에 추가해야 된다:

```
# cat /root/.klogin
jane.root@EXAMPLE.COM
```

이제 su(1)를 사용해 보자:

```
% su
Password:
```

그리고 어떤 결과가 나타나는지 확인한다:

```
# klist
Ticket file:      /tmp/tkt_root_245
Principal:        jane.root@EXAMPLE.COM

  Issued           Expires          Principal
May  2 20:43:12  May  3 04:43:12  krbtgt.EXAMPLE.COM@EXAMPLE.COM
```

14.6.8 다른 명령어 이용

이전 예제에서 *jane* 이라고 부르는 주체와 *root* 인스턴스를 만들었다. 이것은 주체와 같은 유저 이름이고 Kerberos 의 기본값이다; <*principal*>.<*instance*>의 형식 <*username*>.root 는 root 홈 디렉터리의 .klogin 파일에 필요한 엔트리가 있다면 <*username*>이 su(1)로 root 가 되는 것을 허용한다.

```
# cat /root/.klogin
jane.root@EXAMPLE.COM
```

유저 개인의 홈 디렉터리에 그 형식과 같은 라인이 있다면 마찬가지다:

```
% cat ~/.klogin
jane@EXAMPLE.COM
jack@EXAMPLE.COM
```

이 설정은 *jane* 나 *jack*(kinit 를 통해) 인증을 가지고 있는 *EXAMPLE.COM* 의 사람이 rlogin(1), rsh(1), rcp(1)를 사용하여 jane 의 계정이나 이 시스템(grunt) 파일에 접근하는 것을 허용한다.

예를 들어 jane 는 이제 Kerberos 로 다른 시스템에 로그인할 수 있다:

```
% kinit
MIT Project Athena (grunt.example.com)
Password:
% rlogin grunt
Last login: Mon May  1 21:14:47 from grumble
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.

FreeBSD BUILT-19950429 (GR386) #0: Sat Apr 29 17:50:09 SAT 1995
```

그렇지 않으면 같은 머신에서 Jack 이 Jane 의 계정으로 로그인할 수 있다(jane 는 .klogin 파일을 위와 같이 설정하고 Kerberos 가 적용되는 사람은 null 인스턴스로 *jack* 주체를 설정한다)

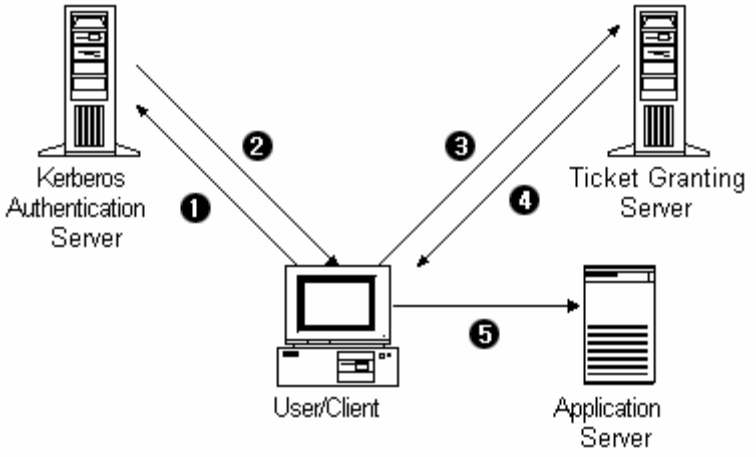
```
% kinit
% rlogin grunt -l jane
MIT Project Athena (grunt.example.com)
Password:
Last login: Mon May  1 21:16:55 from grumble
Copyright (c) 1980, 1983, 1986, 1988, 1990, 1991, 1993, 1994
        The Regents of the University of California.  All rights reserved.
FreeBSD BUILT-19950429 (GR386) #0: Sat Apr 29 17:50:09 SAT 1995
```

14.7 Kerberos5

FreeBSD-5.1 이상의 모든 FreeBSD 릴리즈는 **Kerberos5** 만 지원한다. 따라서 **Kerberos5** 가 유일하게 포함된 버전이고 설정은 **KerberosIV**와 대부분 비슷하다. 다음 정보는 FreeBSD-5.0 릴리즈 이후의 **Kerberos5** 에만 적용된다. KerberosIV를 사용하려는 유저는 security/krb4 포트를 설치한다.

Kerberos 는 유저들이 보안 서버의 서비스를 통해 직접 인증을 받을 수 있도록 네트워크에 추가된 시스템/프로토콜이다. 원격 로그인, 원격 복사, 내부 보안 시스템파일 복사와 위험성이 높은 서비스를 더 안전하고 정확히 제어할 수 있다.

Kerberos 인증 절차 요약



❶ 주체(이 예제에서는 User/Client)는 TGT(Ticket-Granting Ticket)를 요청하기 위해 KDC 에 인증 서비스를 요청한다.

❷ KDC 는 클라이언트에게 TGT 를 준다. 이 TGT 는 키(세션 키 티켓)를 가지고 있고 클라이언트(User/Client)의 패스워드로 암호화되어 있다.

❸ 클라이언트는 다른 주체(이 예제에서 Application Server)에 접근하기 위한 티켓을 Ticket Granting Server 에 요청하기 위해 KDC 에서 받은 TGT 를 사용한다.

❹ Ticket Granting Server 는 클라이언트가 서버의 특정 자원에 접근할 수 있는 서버 키를 준다. 여기서 다시 세션 키 두 개가 생성된다. 하나는 어플리케이션 서버의 키(티켓)로 암호화되어 어플리케이션 서버(Application Server)에게 다른 하나는 유저(User/Client)에게 보내진다.

❺ 클라이언트는 필요한 서비스를 요청할 수 있다.

Kerberos 는 인증을 확인하는 프록시 시스템이라고 설명할 수 있고 신뢰되는 추가적인 인증 시스템이라고도 한다. Kerberos 는 네트워크에서 안전한 유저 인증이라는 오직 한가지 기능만 제공한다. 권한 기능이나(유저가 무엇을 할 수 있는지) 감사 기능은(유저가 무엇을 했는지) 제공하지 않는다. 클라이언트와 서버가 Kerberos 인증을 사용하면 사생활과 데이터 보호를 위해 모든 통신을 암호화할 수 있다.

따라서 Kerberos 는 권한과 감사 서비스를 제공하는 다른 보안기능과 같이 사용하도록 강력히 권장한다.

다음 설명은 분산된 FreeBSD 에 Kerberos 를 어떻게 설정하는지 가이드로 사용할 수 있다. 그렇지만 완벽한 설명은 관련된 매뉴얼 페이지를 참고한다.

Kerberos 설치방법을 설명하기 위해 다양한 이름 공간(name spaces)을 다음과 같이 제어한다:

- DNS 도메인("존")은 example.org 다.
- Kerberos 영역은 EXAMPLE.ORG 다.

  Note: 내부적으로 서비스를 하더라도 Kerberos 를 설정할 때 실제 도메인 이름을 사용한다. 이 방법으로 DNS 문제를 피하고 다른 Kerberos 영역과 상호작용을 보장한다.

14.7.1 역사적 배경

Kerberos 는 네트워크 보안 문제의 해결책으로 MIT 에서 만들었다. Kerberos 프로토콜은 강력한 암호화를 사용하기 때문에 클라이언트는 안전하지 않은 네트워크를 통해서 서버에게 인증을 받을 수 있다.

Kerberos 는 네트워크 인증 프로토콜과 툴 프로그램을(예를 들어 Kerberos 텔넷) 설명하는 프로그램이다. 현재 버전의 프로토콜은 RFC 1510 에 설명되어있는 버전 5 다.

광범위한 운영체제에서 사용되는 이 프로토콜의 여러 가지 툴도 무료로 사용할 수 있다. Massachusetts Institute of Technology(MIT)가 최초로 Kerberos를 개발해서 Kerberos 패키지를 계속해서 개발하고 있다. 일반적으로 미국의 암호화 제품에 사용되어서 역사적으로 미국 수출법에 영향을 미쳤다. MIT Kerberos는 포트(security/krb5)에서 사용할 수 있다. Heimdal Kerberos는 버전 5 의 다른 도구이고 미국 수출법을 피해서 미국 밖에서 개발되었다(따라서 비영리 유닉스 변종에 많이 포함되었다). Heimdal Kerberos 배포본은 포트(security/heimdal)에서 사용할 수 있고 기본 FreeBSD 설치에 포함되어 있다.

다양한 독자를 위해 여기서는 FreeBSD 에 포함된 Heimdal 배포본 사용에 대해 다룬다.

14.7.2 Heimdal KDC 설정

키 배포 센터(KDC)는 Kerberos 가 제공하는(Kerberos 티켓) 인증 서비스를 중앙화한다. KDC 는 Kerberos 영역의 모든 컴퓨터에 의해 신뢰되기 때문에 강력한 보안이 필요하다.

Kerberos 서버가 실행되는 동안 약간의 자원이 필요하지만 KDC 로만 작동하는 머신이 보안상 권장된다.

KDC 설정을 시작하기 위해 KDC 로(여러분의 시스템을 반영하도록 경로를 적절히 수정해야 될 것이다) 동작하도록 /etc/rc.conf 파일에 적절한 설정이 필요하다.

```
kerberos5_server_enable="YES"
kadmind5_server_enable="YES"
kerberos_stash="YES"
```

Note: *Kerberos_stash* 는 FreeBSD 4.X 에서만 사용할 수 있다.

Kerberos 설정 파일 /etc/krb5.conf 를 설정한다:

```
[libdefaults]
    default_realm = EXAMPLE.ORG
[realms]
    EXAMPLE.ORG = {
        kdc = kerberos.example.org
    }
[domain_realm]
    .example.org = EXAMPLE.ORG
```

이 /etc/krb5.conf 파일은 KDC 가 완벽한 호스트 이름 kerberos.example.org 를 가지고 있음을 보여준다. KDC 가 다른 호스트 이름을 가지고 있다면 존 파일에 CNAME(alias) 엔트리를 추가해야 된다.

Note: BIND DNS 서버로 적절히 설정한 대형 네트워크에는 위의 예제를 다음과 같이 수정할 수 있다:

```
[libdefaults]
 default_realm = EXAMPLE.ORG
```

다음 라인으로 example.org 존 파일을 확장한다:

```
_kerberos._udp        IN  SRV     01 00 88 kerberos.example.org.
_kerberos._tcp        IN  SRV     01 00 88 kerberos.example.org.
_kpasswd._udp         IN  SRV     01 00 464 kerberos.example.org.
_kerberos-adm._tcp    IN  SRV     01 00 749 kerberos.example.org.
_kerberos             IN  TXT     EXAMPLE.ORG.
```

그리고 Kerberos 데이터베이스를 생성한다. 이 데이터베이스는 마스터 패스워드로 암호화된 주체 키를 모두 가지고 있다. 이 패스워드는 파일에(/var/heimdal/m-key) 저장되기 때문에 기억할 필요는 없다. 마스터 키를 생성하기 위해 kstash 를 실행하고 패스워드를 입력한다.

마스터 키가 생성되면 kadmin 프로그램에 -l 옵션을 사용하여 데이터베이스를 초기화할 수 있다. 이 옵션은 kadmind 네트워크 서비스를 통하지 않고 직접 데이터베이스 파일을 수정하도록 한다. 이 설정으로 데이터베이스를 생성하기 전에 발생하는 연결 문제를 제어할 수 있다. kadmin 프롬프트에서 최초의 영역 데이터베이스를 생성하기 위해 init 명령을 사용한다.

마지막으로 kadmin 에서 add 명령을 사용하여 첫 번째 주체 키를 생성한다. 지금은 주체에 기본 옵션을 사용하고 modify 명령으로 나중에 변경할 수 있다. 가능한 옵션을 보기 위해 언제라도 *?* 명령을 사용할 수 있다.

샘플 데이터베이스를 생성하는 세션은 다음과 같다:

```
# kstash
Master key: xxxxxxxx
Verifying password - Master key: xxxxxxxx

# kadmin -l
kadmin> init EXAMPLE.ORG
Realm max ticket life [unlimited]:
kadmin> add tillman
Max ticket life [unlimited]:
```

```
Max renewable life [unlimited]:
Attributes []:
Password: xxxxxxxx
Verifying password - Password: xxxxxxxx
```

이제 KDC 서비스를 시작한다. 서비스를 시작하기 위해 /etc/rc.d/kerberos start 와 /etc/rc.d/kadmind start 를 실행한다. 지금은 kerberized 데몬이 실행되지 않지만 KDC 명령어 라인으로 방금 생성한 주체(유저)의 티켓 리스트를 확인하여 KDC 기능을 체크할 수 있다:

```
% k5init tillman
tillman@EXAMPLE.ORG's Password:

% k5list
Credentials cache: FILE:/tmp/krb5cc_500
      Principal: tillman@EXAMPLE.ORG

  Issued           Expires          Principal
Aug 27 15:37:58  Aug 28 01:37:58  krbtgt/EXAMPLE.ORG@EXAMPLE.ORG
```

14.7.3 Heimdal 서비스로 Kerberos 서버 활성화하기

첫째 Kerberos 설정파일 /etc/krb5.conf 를 복사해야 한다. 복사하는 것은 단순히 KDC 에서 클라이언트 컴퓨터로 안전하게(scp(1)같은 네트워크 유틸리티를 사용하거나 플로피 디스크로 물리적으로 안전하게) 복사한다.

다음으로 /etc/krb5.keytab 파일이 필요하다. 이 사항이 Kerberos 를 제공하기 위해 데몬이 활성화된 서버와 워크스테이션의 가장 큰 차이점이다. 서버는 keytab 파일이 필요하다. 이 파일은 KDC 가 각자의 인증을 확인하도록 하는 서버 호스트 키를 가지고 있다. 이 키가 공개되면 서버의 보안이 깨지기 때문에 보안에 유념하여 서버에 전송해야 된다. 이 말은 FTP 처럼 평범한 텍스트 채널을 통해 전송하는 것이 굉장히 어리석다는 것을 의미한다.

보통 keytab 은 kadmin 프로그램을 사용하는 서버에 전송한다. kadmin 을 사용하여 호스트

주체도(krb5.keytab 의 끝인 KDC) 생성해야 되기 때문에 유용하다.

앞에서 받은 티켓을 가지고 있어야 되고 이 티켓으로 kadmind.acl 에서 kademin 인터페이스를 사용할 수 있어야 된다. 접근 제한 리스트 디자인에 대한 자세한 사항은 Heimdal 정보 페이지에서 "원격 관리" 섹션을 본다. 원격 kadmin 접근을 활성화하지 않는다면 단순히 안전하게 KDC 에 연결하여(로컬 콘솔, ssh(1) 또는 Kerberos telnet(1)) kadmin -l 을 사용하여 관리한다.

/etc/krb5.conf 파일을 설치한 후 Kerberos 서버에서 kadmin 을 사용할 수 있다. add --random-key 명령은 서버 호스트 주체를 추가하고 ext 명령은 서버 호스트 주체의 keytab 만 추출할 수 있다. 예를 들면 다음과 같다:

```
# kadmin
kadmin> add --random-key host/myserver.example.org
Max ticket life [unlimited]:
Max renewable life [unlimited]:
Attributes []:
kadmin> ext host/myserver.example.org
kadmin> exit
```

ext 명령은("extract"의 약자) 기본적으로 추출한 키를 /etc/krb5.keytab 에 저장한다. KDC 에서(보안상의 이유로) kadmind 를 실행하지 않아서 원격에서 kadmin 에 접속할 수 없다면, 호스트 주체를 KDC 에 직접 추가한 후 다음과 같은 명령을 사용하여 임시 파일로(KDC 의 /etc/krb5.keytab 을 덮어쓰지 않고) 추출할 수 있다:

```
# kadmin
kadmin> ext --keytab=/tmp/example.keytab host/myserver.example.org
kadmin> exit
```

이제 안전하게 keytab 을 서버 컴퓨터에 복사한다(예를 들어 scp 나 플로피를 사용하여). KDC 의 keytab 을 덮어 쓰지 않기 위해 기본 keytab 이름이 아닌 다른 이름을 지정한다.

서버가 KDC 와(krb5.conf 파일로) 통신할 수 있기 때문에 인증을 받을(krb5.keytab 파일로) 수 있다. 이제 Kerberos 서비스를 활성화할 준비가 되었다. 예를 들면 아래와 같은 라인을 /etc/inetd.conf 에 넣고 /etc/rc.d/inetd restart 로 inetd(8) 서비스를 다시 시작하여 telnet 서비스를 활성화한다:

```
telnet    stream  tcp    nowait  root    /usr/libexec/telnetd    telnetd -a user
```

중요 비트를 -a 로(인증을 위해) user 에게 설정한다. 더 자세한 사항은 telnet(8) 매뉴얼 페이지를 참고한다.

14.7.4 Heimdal 로 클라이언트 Kerberos 활성화

클라이언트 컴퓨터를 설정하는 것은 상당히 쉽다. 모든 설정은 /etc/krb5.conf 에 있는 Kerberos 설정파일만 필요하다. 단순히 KDC 에서 이 파일을 클라이언트 컴퓨터에 안전하게 복사한다.

위에서 생성한 주체 티켓을 가져와서 보여주고 삭제하기 위해 kinit, klist 와 kdestroy 를 사용해서 클라이언트 컴퓨터를 테스트한다. 연결이 되지 않고 티켓을 받아 오는 것이 클라이언트나 KDC 의 문제가 아닌 서버의 문제라도 Kerberos 가 활성화된 서버에 연결할 수 있도록 Kerberos 어플리케이션을 사용할 수 있다.

telnet 같은 어플리케이션을 테스트할 때 패스워드가 보이는지 확인하기 위해 패킷 스니퍼(tcpdump(1) 같은)를 사용한다. 전체 데이터 흐름을 암호화하는(ssh 와 비슷한) -x 옵션으로 telnet 을 사용한다.

Kerberos 클라이언트 핵심 어플리케이션은(전통적으로 kinit, klist, kdestroy 와 kpasswd 라는 이름으로) FreeBSD 기본 시스템에 설치된다. FreeBSD 5.0 이전 버전은 k5init, k5list, k5destroy, k5passwd 와 k5stash 라고 되어 있다.

코어(core)뿐만 아니라 다양한 Kerberos 클라이언트 어플리케이션들도 기본적으로 설치된다. 이것이 최소한의 Hemidal 기본 구성이며 텔넷은 유일하게 Kerberos 가 가능한 서비스다.

Heimdal 포트는 클라이언트 어플리케이션 중에서 빠진 것을 추가할 수 있다: Kerberos 가 활성화되는 버전의 ftp, rsh, rcp, rlogin 과 다른 일반적인 프로그램 몇 개. MIT 포트도 Kerberos 클라이언트 어플리케이션 전체 스위트(suite)를 포함한다.

14.7.5 유저 설정파일: .k5login 과 .k5users

영역에 있는 유저는 전형적으로 로컬 유저 계정과(tillman 이라는 로컬 계정) 매핑되는 Kerberos 주체를(tillman@EXAMPLE.ORG) 가지고 있다. telnet 같은 클라이언트 어플리케이션에는 보통 유저 이름이나 주체가 필요 없다.

그러나 가끔 Kerberos 주체와 대응하지 않는 로컬 유저 계정에 접근하기를 원한다. 예를 들면 tillman@EXAMPLE.ORG 가 로컬 유저 계정 webdevelopers 에 접근할 필요가 있다. 다른 주체도 이 로컬 계정에 접근할 필요가 있을 수 있다.

유저 홈 디렉터리에 있는 .k5login 과 .k5users 파일은 .hosts 와 .rhosts 의 강력한 결합처럼 사용할 수 있다. 예를 들면 다음과 같은 내용의 .k5login 이 유저 webdevelopers 의 홈 디렉터리에 있다면 나열되어 있는 두 개의 주체는 필요한 공유 패스워드 없이 계정에 접근할 수 있다.

```
tillman@example.org
jdoe@example.org
```

이들 명령의 매뉴얼 페이지를 참고하도록 한다. ksu 매뉴얼 페이지에서 .k5users 를 확인할 수 있다.

14.7.6 Kerberos 팁: 티켓과 문제해결

- Heimdal 이나 MIT Kerberos 포트를 사용할 때 PATH 환경변수에 Kerberos 버전의 클라이언트 어플리케이션을 시스템 버전의 클라이언트 이전에 나열해야 된다.

- 시간이 동기화 되었는가? 시간이 동기화되지(5 분 이내로) 않았다면 인증은 실패한다.

- 표준 프로토콜이 아닌 kadmin 을 제외하고 MIT 와 Heimdal 은 정확하게 상호작용한다.

- 호스트 이름을 변경한다면 host/ 주체를 변경해야 되고 keytab을 업데이트 한다. 이 사항을 Apache의 www/mod_auth_kerb에 사용하는 www/ 주체 같은 특별한 keytab 엔트리에도 적용한다.

- 영역에 있는 모든 호스트는 DNS 로(또는 최소한 /etc/hosts 로) 분해할(순방향과 역방향으로) 수 있어야 된다. CNAME 가 동작하더라도 A 와 PTR 레코드는 정확한 위치에 있어야 된다. 관련 에러 메시지가 별로 직관적이지 않기 때문에 주의한다: "refuses authentication because Read req failed: Key table entry not found"

- KDC 의 클라이언트로 동작할지도 모르는 어떤 운영체제들은 setuid 로 root 가 될 수 있도록 ksu 에 퍼미션을 설정하지 않는다. 이 의미는 ksu 가 작동하지 않아서 보안상 안전하겠지만 상당히 귀찮아 진다는 뜻이다. 이것은 KDC 의 에러가 아니다

- MIT Kerberos 에서 주체가 기본 값인 10 시간 이상의 티켓을 가지고 있기를 원한다면 두 주체의 최대 유효기간을 변경하기 위해 kadmin 에서 modify_principal 을 사용해야 된다. 그리고 주체는 좀더 오래 지속되는 티켓을 요청하기 위해 kinit 에 -l 옵션을 사용할 수 있다.

  Note: 문제 해결을 위해 KDC 에서 패킷 스니퍼를 실행하고 워크스테이션에서 kinit 를 실행했다면 이 전에 패스워드를 입력했더라도 kinit 가 실행되자마자 TGT(Ticket Granting Ticket)는 발송된다. 이 말은 Kerberos 서버는 인증을 받지 않은 요청에 TGT 를 자유롭게 전송한다; 그러나 모든 TGT 는 유저 패스워드에서 파생된 키로 암호화 된다. 따라서 유저가 패스워드를 입력하면 KDC 로 전송되지 않고 kinit 가 받은 TGT 를 암호화 하는데 사용된다. 디코딩하여 유효한 시간에 유효한 티켓을 얻었다면 유저는 유효한 Kerberos 인증을 갖게 된다. 이 인증은 나중에 Kerberos 서버와 안전하게 통신할 수 있는 세션 키와 실제 Kerberos 서버의 키로 암호화된 실제 티켓이 포함되어 있다. 이 암호화의 두 번째 레이어는 유저에게 알려져 있지 않지만 각 TGT 의 인증을 Kerberos 서버가 확인할 수 있도록 한다.

- 영역에 있는 모든 컴퓨터의 시간을 동기화해야 된다. 여기에는 NTP 가 적당할 것이다. NTP 에 대한 더 많은 정보는 23.7 장을 본다.

- 기간이 긴 티켓을(예를 들어 일주일) 사용하려면 티켓이 저장되어있는 머신에 연결하기 위해 OpenSSH 를 사용하고 ssh_config 에서 Kerberos *TicketCleanup* 을 *no* 로 설정해야 된다. 그렇지 않고 로그아웃 하면 티켓은 바로 삭제된다.

- 호스트 주체는 기간이 좀더 긴 티켓을 가질 수 있다. 유저 주체는 일주일짜리 티켓을 가지고 있지만 연결하려는 호스트가 9 시간짜리라면 호스트 주체의 케시가 만기되면 티켓 케시도 동작하지 않는다.

- 안전하지 않은 특정 패스워드 사용을 방지하기 위해 krb5.dict 파일을 설정할 때 (kadmind 매뉴얼 페이지에 간단한 설명이 있다) 패스워드 정책이 할당된 주체에만 적용됨을 기억한다. krb5.dict 파일 포맷은 간단하다: 라인당 한 문자열씩 입력한다. 그리고 /usr/share/dic/words 에 심볼릭 링크를 생성하는 것이 유용할 것이다.

14.7.7 MIT 포트와 차이점

MIT 와 Heimdal 설치의 가장 큰 차이점은 다른 명령 설정을(그러나 동일한) 가지고 다른 프로토콜을 사용하는 kadmin 프로그램과 관련된 것이다. 이 말은 많은 것을 함축하고 있다. KDC 가 MIT 라면 원격에서 KDC 를 관리하기 위해 Heimdal kadmin 프로그램을 사용할 수 없다.

클라이언트 어플리케이션도 같은 태스크를 수행하는데 약간 다른 명령라인 옵션을 가지고 있을 것이다. MIT Kerberos 웹 사이트(http://web.mit.edu/Kerberos/www/)의 지시를 따른다. 그리고 경로 문제를 주의한다: MIT 포트는 기본적으로 /usr/local에 설치되기 때문에 PATH 환경변수에 시스템 디렉터리가 먼저 나열되어 있으면 MIT 대신 “일반” 시스템 어플리케이션이 실행된다.

> **Note:** FreeBSD가 제공하는 MIT security/krb5 포트에서 왜 telnetd와 klogind로 로그인하는 것이 약간 다른지 궁금하다면 포트로 설치한 /usr/local/share/doc/krb5/README.FreeBSD 파일을 확인한다. 게시 파일에서 부정확한 퍼미션을 수정하려면 인증에 사용하는 login.krb5 바이너리가 필요하고 포워드된 인증에 맞게 소유권을 적절히 변경할 수 있다.

14.7.8 Kerberos 의 문제를 완화하기

14.7.8.1 Kerberos 에 모두 접근할 수 있거나 아무도 접근할 수 없다.

네트워크에서 활성화된 모든 서비스는 Kerberos 와(또는 네트워크 공격에 안전한) 동작하도록 수정하고 그렇지 않으면 유저 인증이 획득 당해서 다시 사용될 것이다. 예를 들면 모든

원격 쉘에(예를 들어 rsh 와 telnet) Kerberos 를 활성화할 수 있지만 패스워드를 평범한 텍스트로 보내는 POP3 메일 서버는 변경하지 못한다.

14.7.8.2 Kerberos 는 싱글 유저 워크스테이션에 계획되었다

멀티 유저 환경에서 Kerberos 는 보안이 떨어진다. 모든 유저들이 읽을 수 있는 /tmp 디렉터리에 티켓을 저장하기 때문이다. 유저가 여러 사람들과 동시에 컴퓨터를 공유한다면(예: 멀티 유저) 유저의 티켓을 다른 유저가 볼 수(복사) 있다.

이 문제는 명령라인 옵션에 *-c* 파일 이름이나 KRB5CCNAME 환경변수로 해결할 수 있지만 흔히 사용하지 않는다. 주로 유저의 홈 디렉터리에 티켓을 저장하고 간단한 파일 퍼미션으로 이 문제를 해결한다.

14.7.8.3 KDC 가 문제의 원인이다.

KDC 는 마스터 패스워드 데이터베이스를 가지고 있기 때문에 가장 안전해야 된다. KDC 에서는 어떤 서비스도 실행하지 말아야 되며 물리적으로도 안전해야 된다. Kerberos 는 같은 키로("마스터" 키) 암호화한 모든 패스워드를 KDC 에 파일로 저장하기 때문에 위험성이 높다.

노출된 마스터 키는 생각처럼 위험하지 않을 수 있다. 마스터 키는 Kerberos 데이터베이스를 암호화하는데 사용되고 난수 발생기의 소스로 사용된다. 공격자가 마스터 키에 접근할 수 없도록 KDC 에 접근하는 것은 안전해야 된다

추가적으로, KDC 를 사용할 수 없다면(서비스 거부 공격이나 네트워크 문제일 것이다) 인증을 받을 수 없기 때문에 네트워크 서비스를 사용할 수 없고 따라서 서비스 거부 공격의 목표가 된다. 이것은 여러 대의 KDC 나(하나의 마스터와 여러 슬레이브) 대체인증(PAM 이 적당하다)으로 방지할 수 있다.

14.7.8.4 Kerberos 단점

Kerberos는 유저, 호스트 그리고 서비스가 직접 인증을 받을 수 있다. 그러나 유저, 호스트와 서비스가 KDC에 인증을 받는 메커니즘은 없다. 이 의미는 트로이(Trojan) kinit가(예를

들어) 모든 유저 이름과 패스워드를 기록할 수 있다. security/tripwire 또는 다른 파일시스템 무결성 체크 툴을 여기에 사용할 수 있다.

14.7.9 참고할 만한 자료

- Kerberos FAQ(http://www.faqs.org/faqs/kerberos-faq/)
- 인증 시스템 디자인: a Dialogue in Four Scenes (http://web.mit.edu/Kerberos/www/dialogue.html)
- RFC 1510, The Kerberos 네트워크 인증 서비스(V5), RFC 1510
- MIT Kerberos 홈페이지 (http://www.ietf.org/rfc/rfc1510.txt?number=1510)
- Heimdal Kerberos 홈페이지 (http://www.pdc.kth.se/heimdal/)

14.8 방화벽

방화벽은 인터넷에 연결되어 있으며 사설 네트워크에 안전한 보안을 제공하는 어플리케이션을 찾는 사람들에게 관심이 증가되는 영역이다. 이번 장에서는 방화벽이 무엇이고 어떻게 사용하며 FreeBSD 커널에서 제공하는 방화벽 기능을 어떻게 사용하는지 설명한다.

> **Note:** 사람들은 보통 내부 네트워크와 "크고 위험한 인터넷"의 모든 보안 문제를 방화벽이 해결할 것이라고 생각한다. 보안 문제를 도울 수 있지만 엉성하게 설정된 방화벽은 없는 것보다 더 큰 문제를 유발할 수 있다. 방화벽은 시스템에 다른 보안 계층을 추가할 수 있지만 굳은 결심으로 내부 네트워크를 뚫으려는 크래커는 막을 수 없다. 방화벽을 뚫을 수 없다고 너무 믿으면 크래커가 쉽게 내부 네트워크를 뚫게 만들어서 내부 보안에 실패할 것이다.

14.8.1 방화벽은 무엇인가?

오늘날 인터넷에서 일반적으로 사용하는 두 종류의 방화벽이 있다. 첫 번째는 *패킷 필터링*

*라우터*로 더 알려져 있다. 이런 종류의 방화벽은 *multi-homed* 머신(네트워크에 접속하기 위해 여러 IP 주소와 네트워크 인터페이스를 가지고 있는 컴퓨터 호스트를 의미한다. Multi-homed 호스트는 동일한 네트워크나 다른 네트워크에 물리적으로 연결되어 있다)을 사용하고 각각의 패킷을 포워드 하거나 막기 위해 룰을 설정한다. 프록시 서버로 알려져 있는 두 번째 종류는 인증을 제공하고 패킷을 포워드하는 데몬이 패킷 포워딩이 비활성 되어있는 커널을 가진 multi-homed 머신에 설치되어 있을 것이다.

가끔 사이트에서 두 종류의 방화벽을 결합하기 때문에 허가된 머신만(*bastion host*로 알려진) 패킷 필터링 라우터를 통해서 내부 네트워크로 패킷을 보낼 수 있다. 프록시 서비스는 일반 인증 메커니즘 보다 일반적으로 더 보안에 강한 베스천(bastion) 호스트에서 실행된다.

이번 장의 남은 부분에서 다루게 될 커널 패킷 필터를(IPFW 로 알려진) FreeBSD 는 가지고 있다. 프록시 서버는 추가적인 소프트웨어로 FreeBSD 에 설치할 수 있지만 다양한 프록시 서버가 있기 때문에 이번 섹션에서 모든 것을 설명하기는 불가능하다.

14.8.1.1 패킷 필터링 라우터

라우터는 두 개 이상의 네트워크 사이에 패킷을 전달하는 머신이다. 패킷 필터링 라우터는 패킷 전달 여부를 결정하기 전에 각 패킷을 룰 리스트와 비교하는 프로그램이다. 대부분의 최근 IP 라우팅 소프트웨어는 기본적으로 모든 패킷을 전달하는 패킷 필터 기능을 가지고 있다. 필터를 활성화하려면 룰을 설정해야 된다.

방화벽은 패킷을 통과시킬지 결정하기 위해 설정된 룰과 패킷의 헤더 내용이 같은지 관찰한다. 룰에 대응하는 패킷을 발견하면 룰에 설정된 동작을 따른다. 룰의 동작은 패킷 삭제, 패킷 포워드 또는 패킷의 원래 주인에게 ICMP 메시지를 보낼 수도 있다. 룰은 대응된 순서에 따라 처음으로 대응된 룰이 적용된다. 따라서 룰 리스트는 "룰 체인"이라고 할 수 있다.

패킷 대응 규칙은 사용되는 소프트웨어에 따라 다르지만 일반으로 패킷의 소스 IP 주소, 목적지 IP 주소, 소스 포트번호, 목적지 포트 번호(포트를 지원하는 프로토콜에) 또는 패킷 타입에(UDP, TCP, ICMP 등등) 따라 룰을 지정할 수 있다.

14.8.1.2 프록시 서버

프록시 서버는 일반적인 시스템 데몬(telnet, ftpd 등) 대신 특별한 서버를 가진 머신이다. 이

런 서버는 보통 전 방향(한쪽으로만) 연결만 허용하기 때문에 프록시 서버라고 부른다. 방화벽 호스트에 프록시 텔넷 서버를 운용(예를 들어)하면 사람들은 외부에서 방화벽으로 접속하기 위해 텔넷을 사용할 수 있으며 어떤 인증 메커니즘을 통해 내부 네트워크로 접근할 수 있다(다른 방식으로 프록시 서버는 내부 네트워크에서 오는 신호를 외부로 보내는 역할을 할 수 있다).

프록시 서버는 보통 일반 서버보다 보안에 강하고 때때로 원-타임 패스워드 시스템을 포함하는 광범위하고 다양한 인증 메커니즘을 이용할 수 있다. 따라서 누군가 여러분이 사용하는 패스워드를 발견했더라도 한번 사용한 후 패스워드가 즉시 소멸되어 그들이 이 패스워드로 시스템에 접근할 수 없다. 프록시 서버는 유저들이 실제로 호스트 머신에 접근하지 못하도록 하여 보안 시스템에 백도어를 설치하는 것은 아주 어렵게 된다.

프록시 서버는 종종 접근제한 이상의 방법을 가지고 있어서 지정한 호스트만 서버에 접근할 수 있다. 관리자는 보통 어떤 유저가 어떤 머신과 통신할 수 있는지 지정할 수 있다. 역시 어떤 프록시 소프트웨어를 선택하느냐에 따라서 특별한 기능을 사용할 수도 있다.

14.8.2 IPFW 기능은 무엇인가?

FreeBSD 에서 제공하는 IPFW 는 커널 안에 있는 패킷 필터링과 계정시스템으로 유저 기반의 제어 유틸리티 ipfw(8)를 가지고 있다. 이 두 가지로 커널이 라우팅을 결정할 때 사용하도록 룰을 정의하고 질의할 수 있다.

IPFW 에 관련된 두 개의 분야가 있다. 방화벽 섹션은 패킷 필터링을 수행한다. 그리고 방화벽 섹션에서 사용하는 것과 비슷한 룰 기반에서, 라우터 사용량을 추적하는 IP 계정 섹션이 있다. 예를 들면 얼마나 많은 라우터 트래픽이 특정 머신에서 오는지 또는 얼마나 많은 WWW 트래픽을 포워딩하는지 관리자가 모니터 할 수 있다.

IPFW 의 디자인으로 라우터가 없는 머신의 입력과 출력 연결에 IPFW 을 패킷 필터링처럼 사용할 수 있다. 이것은 IPFW 를 특이하게 사용하는 경우고 상황에 맞는 명령과 기술을 사용해야 된다.

14.8.3 FreeBSD 에서 IPFW 사용

커널 IPFW 시스템의 원하는 기능에 따라 커널 설정파일에 하나 이상의 옵션을 추가하고 커

널을 다시 컴파일한다. 커널을 다시 컴파일하는 방법에 대한 자세한 사항은 "커널 다시 설정하기(8 장)"를 본다.

> **주의:** *모든 IP 를 거부*하는 것이 IPFW 의 기본 정책이다. 시작할 때 접근을 허용하는 룰을 추가하지 않았다면 재 부팅 후 방화벽이 활성화된 커널로 바뀌어서 서버에 접근할 수 없다. 방화벽 기능을 처음으로 사용할 때 /etc/rc.conf 파일에 *firewall_type=open* 으로 설정하고 새로운 커널 기능이 제대로 작동하는 것을 테스트한 후 /etc/rc.firewall 의 방화벽 룰을 깨끗이 정리할 것을 권장한다. ssh 을 사용하는 것보다 안전하게 로컬 콘솔에서 방화벽 설정을 초기화하는 것이 좋을 것이다. 커널을 빌드하는 다른 옵션은 *IPFIREWALL* 과 *IPFIREWALL_DEFAULT_TO_ACCEPT* 옵션을 사용한다. 이 방법은 기본 IPFW 의 룰을 변경해서 *모든 IP 를 허용*하여 서버에 접근하지 못하는 것을 피할 수 있다.

IPFW 와 관련된 4 가지 커널 설정 옵션이 있다:

options IPFIREWALL

패킷 필터링 코드를 커널에 컴파일한다.

options IPFIREWALL_VERBOSE

패킷 로그는 syslogd(8)을 통해 남긴다. 필터 룰에 패킷 로그를 지정하였더라도 이 옵션이 없으면 로그를 남기지 않는다.

options IPFIREWALL_VERBOSE_LIMIT=10

syslogd(8)를 통해 로그로 남기는 페킷의 수를 엔트리 별로 제한한다. 적대적인 환경에서 방화벽 활동을 로그로 남기길 원하겠지만 syslog 가 가득 차는 서비스 거부 공격을 당하도록 서비스를 열어두고 싶지 않을 것이다.

체인 엔트리가 지정된 패킷 제한에 도달하면 로그는 특정 엔트리가 될 때까지 꺼진다. 다시 로그를 기록하려면 ipfw(8) 유틸리티를 사용하여 관련된 카운터를 리셋해

야 된다:

```
# ipfw zero 4500
```

4500 은 로그를 계속 기록하려는 체인 엔트리다.

options IPFIREWALL_DEFAULT_TO_ACCEPT

"거부"에서 "허가"로 기본 룰을 변경한다. 이 옵션은 *IPFIREWAL* 을 지원하는 커널로 부팅했지만 방화벽을 아직 설정하지 않았다면 방화벽에 접근이 거부되는 것을 방지한다. 증가하는 특정 문제의 필터로 ipfw(8)을 사용하는 것도 매우 유용하다. 이 옵션은 방화벽을 열고 동작을 변경시키기 때문에 사용에 유의한다.

Note: FreeBSD 의 이전 버전은 *IPFIREWALL_ACCT* 옵션을 가지고 있다. 방화벽 코드가 자동으로 계정 기능을 포함하기 때문에 이제 사용하지 않는다.

14.8.4 IPFW 설정

IPFW 소프트웨어 설정은 ipfw(8) 유틸리티를 사용할 수 있다. 이 명령의 구문은 아주 복잡해 보이지만 구조를 알면 비교적 단순하다.

이 유틸리티에 사용되는 4 개의 명령어 카테고리가 있다: 추가/삭제, 리스트, 플러싱 (flushing) 그리고 클리어(clear). 추가/삭제는 패킷을 허용, 거부, 로그로 남기는지 제어하는 룰을 생성하는데 사용된다. 리스트는 룰 설정의(다른 말로 체인) 내용과 패킷 카운터를(계정) 검사하는데 사용된다. 플러싱은 체인에서 모든 엔트리를 삭제하는데 사용한다. 클리어는 하나 이상의 계정 엔트리를 0 으로 만드는데 사용한다.

14.8.4.1 IPFW 룰 변경

명령어 구문 형식은 다음과 같다:

ipfw[-N]*명령* [index]*동작* [log] *프로토콜 주소* [options]

이 형식의 명령을 사용할 때 유효한 플래그가 하나 있다:

-N

출력에서 주소와 서비스 이름을 분석한다.

주어진 *명령*은 아주 짧은 형식으로 줄일 수 있다. 유효한 *명령*은 다음과 같다:

add

방화벽/계정 룰 리스트에 엔트리 추가

delete

방화벽/계정 룰 리스트에서 엔트리 삭제

이전 버전의 IPFW 는 방화벽과 계정 엔트리로 나누어져 있었다. 현재 버전은 각 방화벽 엔트리별로 패킷 계정을 제공한다.

index 값이 제공되었다면 체인에서 엔트리를 특정 위치로 옮기는데(앞에서 패킷은 룰 리스트 중 처음으로 대응하는 룰의 동작을 따른다고 했다. 따라서 룰 순서를 지정하는데 사용된다) 사용된다. 그렇지 않으면 엔트리는 마지막 체인 엔트리 보다 100 이 큰 인덱스의 마지막 체인에 지정된다.

*IPFIREWALL_VERBOSE*를 커널에 컴파일 하였다면 *log* 옵션은 룰에 대응하는 패킷을 시스템 콘솔에 출력한다.

유효한 *동작*은 다음과 같다:

reject

패킷을 버리고 패킷을 보낸 호스트에게 ICMP 호스트나 포트에 도달할 수 없다는 패킷을 보낸다.

allow

패킷을 그냥 통과 시킨다(엘리어스: *pass*, *permit* 와 *accept*).

deny

패킷을 버린다. 소스에게 ICMP 메시지로 통보하지 않는다(따라서 패킷은 절대 목적지에 도달할 수 없다).

count

패킷 카운터를 업데이트 하지만 이 룰 기반에서 패킷의 허가/거부는 하지 않는다. 다음 체인 엔트리를 계속 검색한다.

각 *동작*은 가장 짧게 알아볼 수 있도록 지정한다.

지정할 수 있는 ***프로토콜***은 다음과 같다:

all

모든 IP 패킷 대응

icmp

ICMP 패킷 대응

tcp

TCP 패킷 대응

udp

UDP 패킷 대응

주소는 다음과 같이 범위를 지정할 수 있다:

*address/mask [port]*에서 *address/mask* [*port*]까지 [*interface*를 통해]

지원하는 포트와 *protocols*를 결합할 때만 *port*를 지정할 수 있다(UDP 와 TCP).

*interface*는 옵션으로 해당 인터페이스로 유입되는 패킷만 처리하도록 IP 주소나 로컬 IP 인터페이스의 도메인 이름 또는 인터페이스 이름을 지정한다. 인터페이스 유닛 번호는 와일드카드로(*) 지정할 수 있다. 예를 들어 *ppp**는 모든 커널 ppp 인터페이스와 대응된다.

address/mask 지정에 사용되는 구문은 3 가지 중 하나다:

address
address/mask-bits
address:make-pattern

유효한 호스트 이름을 IP 주소 대신 지정할 수 있다. *maks-bits*는 어떤 비트를 주소 매스크에 설정해야 되는지 숫자로 표현한다. 예를 들어 192.216.222.1/24 로 지정하는 것은 C 클레스 서브넷에(이 경우 192.216.222) 메칭되는 모든 주소를 허가하는 매스크를 생성한다. *mask-pattern*은 주어진 주소와 논리적으로 AND 연산된 IP 주소다. 키워드 any 는 "모든 IP 주소"를 지정할 때 사용한다.

차단하려는 **포트 번호**는 다음과 같이 지정한다:

port [,*port* [,*port* [...]]]

위와 같이 하나의 포트나 여러 개의 포트 지정할 수 있고 포트 범위를 지정하려면 다음과 같이 시작과 끝나는 포트를 입력한다.

port-port

사용 가능한 *options* 은 다음과 같다:

frag

패킷이 데이터그램(데이터 또는 패킷을 의미한다)의 첫 번째 조각이 아니라면 대응한다.

in

입력 패킷이면 대응한다.

out

출력 패킷이면 대응한다.

ipoptions *spec*

IP 헤더가 *spec* 에 나열한 옵션을 가지고 있다면 대응한다. 지원되는 IP 옵션: *ssrr* (strict source route), *lsrr* (loose source route), *rr* (record packet route)와 *ts* (time stamp). 특별한 옵션이 없으면 읽기로 지정될 것이다.

established

패킷이 이미 연결된 TCP 연결의 일부분이면(다시 말해 RST 나 ACK 비트 설정을 가지고 있다면) 대응한다. *established* 룰을 체인의 앞부분에 지정하여 방화벽의 성능을 최적화할 수 있다.

setup

패킷이 TCP 연결을(SYN 비트는 설정 되지만 ACK 비트는 아니다) 만들려고 하면 대응한다.

tcpflags *flags*

TCP 헤더가 콤마로 나누어진 *flags* 리스트를 포함하면 대응한다. 지원되는 플래그는 *fin*, *syn*, *rst*, *psh*, *ack* 와 *urg* 다. 특별한 플래그가 없으면 읽기로 표시될 것이다.

icmptypes *types*

ICMP 종류가 *types* 리스트에 있다면 대응한다. 이 리스트는 콤마로(,) 나누어진 각 범위를 and/or 로 결합한 것처럼 나누어져 있을 것이다. 일반적으로 사용되는 ICMP 종류: *0* 에코 응답(핑 응답), *3* 목적지 도달불가, *5* 리다이랙트, *8* 에코 요청(핑 요청)과 *11* 시간 초과 (traceroute(8)에서 TTL 을 표현하는데 사용된다).

14.8.4.2 IPFW 룰 리스트

명령어 구문은 다음과 같다:

ipfw [-a][-c][-d][-e][-t][-N][-S] list

이 형식의 명령을 사용할 때 7 개의 유효한 플래그가 있다:

-a

나열하는 동안 카운터 값을 보여준다. 이 옵션이 계정 카운터를 볼 수 있는 유일한 방법이다.

-c

조밀한 형식으로 룰을 보여준다.

-d

동적인 룰과 추가로 정적인 룰도 보여준다.

-e

*-d*가 지정되어 있다면 소멸된 동적 룰도 보여준다.

-t

각 체인 엔트리에 마지막으로 대응된 시간을 보여준다. 이 시간 리스트는 ipfw(8) 유틸리티로 사용되는 입력구문과 호환되지 않는다.

-N

주어진 주소와 서비스 이름을 분석한다.

-S

각 룰에 속한 설정을 보여준다. 이 플래그가 지정되지 않았다면 비활성된 룰은 나열되지 않는다.

14.8.4.3 IPFW 룰들 삭제

체인을 삭제하기 위한 구문은 다음과 같다:

ipfw flush

이 명령은 커널에 의해(인덱스 65535) 강제로 고정된 기본 정책을 제외한 방화벽 체인의 모든 엔트리를 삭제한다. 룰을 정리할 때 경고 사용; 기본 거부 정책은 허용 엔트리가 체인에 추가될 때까지 네트워크에서 시스템을 차단한다.

14.8.4.4 IPFW 패킷 카운터 정리

하나 또는 더 많은 패킷 카운트를 정리하기 위한 구문은 다음과 같다:

ipfw zero [*index*]

index 인수 없이 사용하면 모든 패킷 카운터를 정리한다. *index* 를 제공하였다면 정리 동작은 지정한 체인 엔트리에만 영향을 미친다.

14.8.5 ipfw 예제 명령

이 명령은 호스트 evil.crackers.org 로부터 nice.people.org 호스트의 텔넷 포트로 모든 패킷을 거부한다:

ipfw add deny tcp from evil.crackers.org to nice.people.org 23

다음 예제는 crackers.org 전체 네트워크로부터(C 클래스) nice.people.org 머신으로(모든 포트)의 모든 TCP 트래픽을 거부하고 로그로 남긴다.

ipfw add deny log tcp from evil.crackers.org/24 to nice.people.org

사람들이 내부 네트워크로(C 클래스의 서브넷) X 세션을 보내지 못하도록 하려면 다음 명령

이 필요한 필터링을 할 것이다:

```
# ipfw add deny tcp from any to my.org/28 6000 setup
```

계정 레코드를 보려면 다음 명령을 사용한다:

```
# ipfw -a list
```

또는 짧은 형식의 명령은 다음과 같다:

```
# ipfw -a l
```

체인 엔트리가 마지막으로 대응된 시간을 볼 수 있다:

```
# ipfw -at l
```

14.8.6 패킷 필터링 방화벽 만들기

> Note: 다음은 단지 가정이다. 각 방화벽의 필요 조건이 다르기 때문에 여러분의 특정 필요조건에 맞는 방화벽을 어떻게 만드는지 설명할 수 없다.

방화벽을 처음 설정할 때 안전한 환경에서 방화벽을 설정할 수 없고 테스트할 공간이 없다면 로그가 남는 명령어 버전을 사용해서 커널이 로그를 남기도록 강력히 권장한다. 이 방법으로 문제 영역을 빨리 확인할 수 있고 큰 혼란 없이 처리할 수 있다. 초기설정이 끝난 후라도 가능한 공격에 대한 흔적인 "거부"를 로그로 남기고 필요한 경우 방화벽의 룰을 수정하는 것도 로그로 남기도록 한다.

> Note: accept 된 것을 로그로 남긴다면 많은 로그 데이터를 만든다는 것을 알고 있어야 한다. 로그 엔트리는 방화벽을 지나는 모든 패킷별로 로그 엔트리를 하나씩 생성하기 때문에 거대한 FTP/http 전송 등은 시스템을 아주 느리게 한다. 또한 패킷이 지나가기 전에 커널에 의해 더 많은 작업이 필요하기 때문에 이러한 패킷에 대한 로그가 증가하게 되어있다. 또한 syslogd 는 추가적인 모든 데이터를 디스크에 로그로 남길 때 프로세서를 오랫동안 사용하고 /var/log 파티션을 쉽게 채울 수 있다.

/etc/rc.conf.local 이나 /etc/rc.conf 에서 방화벽을 활성화한다. 관련 매뉴얼 페이지에서 방화벽을 정지하고 미리 설정된 설정을 어떻게 보는지 설명한다. 미리 설정되어 있는 것을 사용하지 않겠다면 rc.conf 에 지정할 수 있는 파일에 ipfw list 명령으로 현재 룰 세트를 출력한다. 방화벽을 활성화하기 위해 /etc/rc.conf.local 이나 /etc/rc.conf 를 사용하지 않는다면 IP 인터페이스를 설정하기 전에 방화벽이 활성화되어 있어야 된다.

다음 문제는 방화벽이 실제로 무엇을 해야 되는지 설정하는 것이다. 이것은 외부에서 네트워크 접근을 원하는지 그리고 내부에서 외부로 얼마나 많이 접근할 것 인가에 따라 다르다. 일반적인 몇 가지 룰은 다음과 같다:

- TCP 1024 포트 아래로 들어오는 패킷은 모두 막는다. 대부분 보안과 민감한 서비스가 finger, SMTP(메일)와 telnet 같은 것이기 때문이다.

- 들어오는 모든 UDP 트래픽을 막는다. UDP 와 관련된 유용한 서비스와 트래픽은 매우 적고 보안을 위협하기 때문이다(예: Suns RPC 와 NFS 프로토콜). 또한 UDP 는 비 연결 지향 프로토콜이기 때문에 단점도 될 수 있다; 입력되는 UDP 트래픽을 막는 것은 UDP 응답으로 나가는 트래픽도 막게 된다. 이 방법은 외부에 있는 서버를 사용하는 사람들에게(내부에 있는) 문제가 될 수 있다. 외부 서버에 접근을 허용하려면 방화벽을 통해 내부 UDP 포트 191 에서 1525 번으로 들어오는 패킷을 허가해야 된다. ntp 는 포트 123 으로 입력되는 다른 서비스이기 때문에 이 포트도 열어 줘야 될 것이다.

- 외부로부터 포트 6000 으로 들어오는 트래픽을 막는다. 포트 6000 은 X11 서버에 접근하는데 사용되기 때문에 보안을 위협할 수 있다(특히 워크스테이션에서 xhost +를 습관적으로 사용하는 사람). X 11 은 실제로 6000 부터 시작하는 포트 범위를 사용할 수 있기 때문에 6000 이상을 제한하여 머신에서 사용할 수 있는 X 디스플레이 개수를 제한할 수 있다. RFC 1700 에(할당된 번호) 정의하는 상한선은 6063 이다.

- 어떤 포트를 내부 서버가 사용하는지(예: SQL 서버 등) 체크한다. 보통 위에서 지정한 1~1024 범위를 벗어나기 때문에 이들 포트도 막는 것이 좋을 것이다.

방화벽 설정을 위한 다른 체크 리스트는 CERT (http:www.cert.org/tech_tips/packet_filtering.html)에서 확인할 수 있다.

위의 내용은 오직 가이드라인만 제시한다. 어떤 필터 룰을 방화벽에 사용할지 직접 결정해야 된다. 위에서 제공한 충고를 따르고 누군가 네트워크에 들어왔더라도 어떤 책임도 갖지 않는다.

14.8.7 IPFW 오버헤드와 최적화

많은 사람들은 얼마나 많은 IPFW 오버헤드가 시스템에 추가되는지 알고 싶어한다. 이에 대한 대답은 대부분 룰 설정과 프로세서 속도에 따라 다르다. 그리고 이더넷과 관련된 대부분의 어플리케이션과 작은 룰 셋에 대해서는 무시해도 된다. 여러분의 호기심을 만족시키고 실재 필요한 평가를 위해 다음 내용을 읽도록 한다.

다음 평가는 486-66 에 2.2.5-STABLE 를 사용하였다(IPFW 가 FreeBSD 의 마지막 릴리즈에서 약간 변경되었지만 비슷한 속도로 수행되고 있다). IPFW 는 *ip_fw_chk* 루틴에 사용되는 시간을 측정해서 1000 패킷마다 결과를 콘솔에 출력하도록 수정하였다.

룰 셋 두 가지를 룰별로 1000 번씩 테스트하였다. 첫 번째는 룰이 반복되는 최악의 상황을 보여주도록 디자인 하였다:

```
# ipfw add deny tcp from any to any 55555
```

위 명령은 마지막으로 패킷이 룰에 대응하지(포트 번호의 장점) 않는다고 결정하기 전에 대부분의 IPFW 의 패킷 체크 루틴이 수행되는 최악의 시나리오를 보여준다. 이 룰이 999 번 반복되고 모든 IP 접근(any 에서 any 까지)을 허용한다.

두 번째 룰 설정은 룰 체크를 빨리 중단하도록 디자인하였다:

```
# ipfw add deny ip from 1.2.3.4 to 1.2.3.4
```

위의 룰에 대응하지 않는 소스 IP 주소는 이런 룰을 매우 빨리 지나가도록 한다. 전과 마찬가지로 1000 번째 룰은 모든 IP 를 허용한다.

앞의 경우 패킷별로 프로세스 오버헤드는 대략 2.703 ms/packet 또는 대략 룰 별로 2.7 밀리 초다. 그래서 이론상 이러한 룰에 의한 패킷 프로세싱 제한은 초당 대략 370 패킷이 된다. 10Mbps 이더넷에 ~1500 바이트 패킷 크기는 오직 55.5%의 대역폭만 이용하게 된다.

나중의 경우 각 패킷은 대략 1.172ms 로 진행하거나 대충 룰별로 1.2 밀리 초가 소요된다. 이론적인 패킷 프로세스 제한은 10Mbps 이더넷 대역을 소비할 수 있는 초당 대략 853 패킷이다.

과도한 수의 룰 테스트와 이러한 룰은 실제 세계에서는 통하지 않는다. 이러한 결과는 시간 정보만 보여주는데 사용된다. 여기 효율적인 룰을 설정할 때 고려할 몇 가지가 있다:

- TCP 트래픽을 주로 제어하기 위해 *established* 룰을 먼저 적용한다. 이 룰 전에 어떤 *allow tcp* 문도 두지 않는다.

- 룰 설정에서 거의 사용하지 않는 룰 이전에 자주 사용되는 룰을 둔다(물론 방화벽 변경 없이). 어떤 룰이 보통 사용되는지 ipfw -a 1 로 패킷 카운팅 통계를 체크해 볼 수 있다.

14.9 OpenSSL

많은 유저가 간과하는 기능 중 한가지는 FreeBSD 에 포함된 OpenSSL 툴 킷이다. OpenSSL 은 일반 통신 레이어에 암호화 전송 레이어를 제공한다; 따라서 많은 네트워크 어플리케이션 및 서비스와 OpenSSL 이 상호 작동하게 한다.

OpenSSL을 사용하는 방법에는 메일 클라이언트의 암호화된 인증, 신용카드 결제와 같은 웹 기반 트랜젝션 등이 포함될 것이다. www/apache13-ssl과 mail/sylpheed-claws와 같은 포트들은 OpenSSL을 지원하도록 컴파일 한다.

> Note: 대부분의 경우 포트 컬렉션은 WITH_OPENSSL_BASE make 변수를 “yes”로 설정하지 않았다면 security/openssl을 빌드한다.

FreeBSD 에 포함되어 있는 OpenSSL 버전은 보안 소켓 레이어 v2/v3 (SSLv2/SSlv3), 전송 레이어 보안 v1(TLSv1) 네트워크 보안 프로토콜 그리고 어플리케이션에 사용할 수 있도록 일반적인 암호화 라이브러리를 제공한다.

> Note: OpenSSL 이 IDEA 알고리즘을 지원하지만 미국의 특허권 때문에 기본적으로 비활성되어 있다. 사용하기를 원하면 라이선스를 다시 확인하고 제한을 수락할 수 있다면 make.conf 에 MAKE_IDEA 변수를 설정한다.

아마도 OpenSSL 을 가장 일반적으로 사용하는 방법은 어플리케이션을 사용할 수 있도록 증명서를 제공하는 것이다 이들 증명서는 기업체나 개인의 자격을 증명한다. 증명서에 여러 증명 기관이나 CA(Certificate Authority)의 검증이 없다면 보통 경고 메시지가 뜬다. 증명 기관은 기업체나 개인의 유효한 자격을 검증하기 위해 증명서에 서명을 해주는 VeriSign 과 같은 회사다. 이러한 절차는 비용이 소요되므로 증명서를 사용하기 위해 꼭 필요한 것은 아니다; 그러나 의심이 많은 사람들에게 두려움을 덜어 줄 수 있다.

14.10.1 증명서 만들기

증명서를 만들기 위해 다음 명령을 사용할 수 있다:

```
# openssl req -new -nodes -out req.pem -keyout cert.pem
Generating a 1024 bit RSA private key
..............++++++
.....................................++++++
writing new private key to 'cert.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:PA
Locality Name (eg, city) []:Pittsburgh
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Company
Organizational Unit Name (eg, section) []:Systems Administrator
Common Name (eg, YOUR name) []:localhost.example.org
Email Address []:trhodes@FreeBSD.org

Please enter the following 'extra' attributes
to be sent with your certificate request
```

```
A challenge password []:SOME PASSWORD
An optional company name []:Another Name
```

"Common name" 프롬프트가 나타나면 도메인 이름을 입력한다. 이 프롬프트에는 증명서를 검증하는 서버 이름이 필요하다; 원하는 문자를 입력하면 되지만 도메인 이름을 입력하면 쓸데없는 증명서를 만들게 된다. 예를 들어 만기일이나 다른 암호화 알고리즘 같은 다른 옵션도 사용할 수 있다. 완전한 옵션 리스트는 openssl(1) 매뉴얼 페이지에서 찾을 수 있다.

cert.pem 파일은 앞서 명령을 입력한 디렉터리에 없다. 이 증명서는 수많은 CA(인증 기관) 중 한곳의 서명을 받도록 보내졌을 것이다.

CA 의 서명이 필요 없다면 자체적으로 서명이 된 증명서를 만들 수 있다. 첫째로 CA 키를 생성한다:

```
# openssl gendsa -des3 -out ₩
myca.key 1024
```

증명서를 만들기 위해 이 키를 사용한다:

```
# openssl req -new -x509 -days 365 -key ₩
myca.key -out new.crt
```

두 개의 새로운 파일이 디렉터리에 생성되었다: 인증 기관 서명 파일 myca.key 와 증명서 new.crt 가 생성되었다. 이들 파일은 root 만 읽을 수 있도록 하여 /etc 와 같은 적당한 디렉터리로 이동시킨다. chmod 유틸리티로 0600 퍼미션이 적당할 것이다.

14.10.2 증명서를 사용하는 예제

이들 파일로 무엇을 할 수 있는가? 적절한 사용법은 Sendmail MTA 의 연결을 암호화하는데 사용한다. 그래서 로컬 MTA 를 통해 메일을 보내는 유저가 깨끗한 텍스트 인증을 사용하지 않게 한다.

> Note: 어떤 MUA 는 로컬에 증명서를 설치하지 않으면 에러가 발생하기 때문에 이것은 이상적인 사용법이 아니다. 증명서 설치에 대한 더 자세한 정보는 소프트웨어에 포함되어 있는 문서를 참고한다.

다음 라인을 로컬 .mc 파일에 입력한다:

```
dnl SSL Options
define(`confCACERT_PATH',`/etc/certs')dnl
define(`confCACERT',`/etc/certs/new.crt')dnl
define(`confSERVER_CERT',`/etc/certs/new.crt')dnl
define(`confSERVER_KEY',`/etc/certs/myca.key')dnl
define(`confTLS_SRV_OPTIONS', `V')dnl
```

/etc/certs/는 증명서와 키 파일을 저장할 디렉터리다. 이제 남은 단계는 로컬 .cf 파일을 다시 빌드한다. /etc/mail 디렉터리에서 make install 을 입력하여 쉽게 빌드할 수 있다. 그리고 Sendmail 데몬을 다시 시작하는 make restart 명령을 사용한다.

/var/log/maillog 파일에 에러가 없고 모든 것이 정상이라면 Sendmail 이 프로세스 리스트에 나타난다.

간단한 테스트를 위해 telnet(1) 유틸리티를 사용하여 메일 서버에 연결한다:

```
# telnet example.com 25
Trying 192.0.34.166...
Connected to example.com.
Escape character is '^]'.
220 example.com ESMTP Sendmail 8.12.10/8.12.10; Tue, 31 Aug 2004 03:41:22 -0400
(EDT)
ehlo example.com
250-example.com Hello example.com [192.0.34.166], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-AUTH LOGIN PLAIN
250-STARTTLS
250-DELIVERBY
```

```
250 HELP
quit
221 2.0.0 example.com closing connection
Connection closed by foreign host.
```

"STARTTLS" 라인이 출력에 나타나면 모든 것이 정상적으로 동작하는 것이다.

14.10 IPsec 로 VPN 만들기

FreeBSD 게이트웨이를 사용하여 인터넷으로 나눠진 두 개의 네트워크에 VPN 을 생성한다.

14.10.1 IPsec 이해

이 섹션은 FreeBSD 와 Microsoft Windows 2000/XP 머신 환경에서 안전하게 통신할 수 있도록 IPsec 를 설정하여 사용하는 과정을 설명한다. IPsec 설정은 사용자 커널 설정에(8 장) 대해 이해하고 있어야 된다.

IPsec 는 인터넷 프로토콜 레이어의 최 상단에 있는 프로토콜로서 두 대 이상의 머신이 안전하게 통신할 수 있게 한다. FreeBSD IPsec "network stack"은 IPv4 와 IPv6 를 지원하는 KAME(http://www.kame.net/) 프로젝트 기반이다.

> **Note:** FreeBSD 5.X 는 OpenBSD 에서 가져온 "Fast IPsec"라는 "하드웨어 가속" IPsec 스택을 가지고 있다. IPsec 의 성능을 최적화하기 위해 crypto(4) 서브시스템으로 하드웨어 암호화를 적용한다. 이 서브시스템은 새로운 것이기 때문에 KAME 버전의 IPSec 에서 사용할 수 있는 모든 기능을 지원하지 않는다. 그러나 하드웨어 가속 IPsec 를 활성화하려면 커널 설정파일에 다음 옵션을 추가해야 된다:
>
> ```
> options         FAST_IPSEC   # new IPsec (cannot define w/ IPSEC)
> ```
>
> 현재 KAME 버전의 IPsec 와 Iue 에서 "Fast IPsec" 서브시스템을 사용하는 것은 불가능하다. 더 많은 정보는 fast_ipsec(4) 매뉴얼 페이지를 참고한다.

IPsec 는 두 개의 서브 프로토콜로 이루어진다:

- *Encapsulated Security Payload(ESP)*는 대칭 암호화 알고리즘으로(Blowfish 와 3DES 같은) 내용을 암호화하여 불법적인 패킷 캡쳐 시도로부터 IP 패킷 데이터를 보호한다.

- *Authentication Header(AH)*는 암호화 체크섬 계산과 보안 해싱(hashing) 기능으로 IP 패킷 헤더 필드를 해싱하여 불법적인 패킷 캡쳐 시도와 스푸핑(변조)으로부터 IP 패킷 헤더를 보호한다.

ESP 와 AH 는 환경에 따라 같이 사용하기도하고 따로 사용할 수도 있다.

IPsec 는 두 호스트 사이의 트래픽을 직접 암호화할 수 있고 안전한 통신을 위해 두 개의 서브넷에 터널모드(*Tunnel Mode*) 라는 "가상 터널"을 생성할 수 있다. 후자는 일반적으로 가상 사설 망(*Virtual Private Network* (*VPN*))으로 더 유명하다. FreeBSD 의 IPsec 서브시스템에 대한 더 자세한 정보는 ipsec(4) 매뉴얼 페이지를 참고한다.

커널에 IPsec 지원을 추가하려면 커널 설정파일에 다음 옵션을 추가한다:

```
options    IPSEC          #IP security
options    IPSEC_ESP      #IP security (crypto; define w/ IPSEC)
```

IPsec 디버깅을 지원하려면 다음 커널 옵션도 추가해야 된다:

```
options    IPSEC_DEBUG   #debug for IP security
```

14.10.2 문제

VPN 을 구성하는 표준은 없고 장점과 단점을 가진 여러 가지 기술을 사용하여 구축할 수 있다. 이 문서에서는 여러 가지 시나리오와 각 시나리오에 따른 VPN 구축 전략을 보여준다.

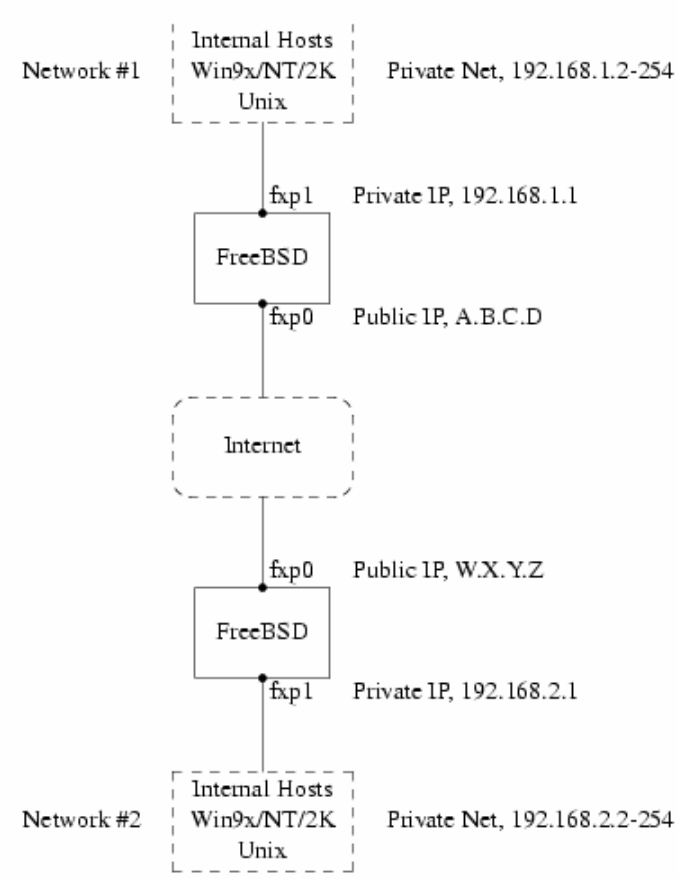
14.10.3 시나리오 #1: 인터넷에 연결되어 하나처럼 동작하는 두 개의 네트워크

이것이 내가 처음으로 VPN 에 관심을 가지게 된 시나리오다. 다음과 같은 전제를 기반으로 한다:

- 최소한 두 개의 사이트가 있다.
- 두 사이트는 사설 IP 를 사용한다
- 두 사이트는 FreeBSD 가 동작하는 게이트웨이를 통해 인터넷에 연결되어있다.
- 각 네트워크의 게이트웨이는 최소한 한 개의 공인 IP 주소를 가지고 있다.
- 두 네트워크의 내부 주소는 공인이든 사설 IP 주소던지 상관없다. 필요하다면 게이트웨이 머신에서 NAT 를 실행할 수 있다.
- 두 네트워크의 내부 IP 주소는 충돌하지 않는다. 이 작업에 VPN 기술과 NAT 를 조합하여 사용하는 것도 이론상 가능하지만 엄청난 설정 작업이 수반될 것 이라고 생각한다.

양쪽 내부 네트워크에 같은 범위의 사설 IP 주소를 사용하고 있었다면 한쪽은 번호를 바꿔야 된다.

네트워크 토플로지는 다음과 비슷할 것이다:



그림에서 두 개의 공인 IP 주소(Public IP A.B.C.D 와 Public IP W.X.Y.X)를 주의한다. 이 문서에서는 두 개의 공인 IP 를 의미하는 문자로 바꾸어 사용했다. 이 문서에서 이들 문자를 보게 되면 여러분의 공인 IP 주소로 바꾼다. 그리고 두 개의 게이트웨이 머신이 가지고 있는 .1 내부 IP(192.168.1.1 과 192.168.2.1 을 의미한다)도 마찬가지로 변경한다. 두 개의 네트워크는 서로 다른 사설 IP 주소를(각각 192.168.1.x 와 192.168.2.x) 가지고 있다. 사설 네트워크의 모든 머신은 .1 머신을 기본 게이트웨이로 사용하였다.

네트워크 관점에서, 가끔 패킷을 잃어버리는 경향이 있는 약간 느린 라우터에 머신들이 직접 연결되어있더라도 각 네트워크는 다른 네트워크에 머신이 있는 것처럼 보여야 된다.

이 의미는 머신 192.168.1.20 에서 다음과 같이 ping 을 실행했을 때 동작해야 된다.

ping 192.168.2.34

Windows 머신은 로컬 네트워크 머신을 탐색할 수 있는 것처럼 다른 네트워크에 있는 머신을 볼 수 있고 공유 파일을 탐색할 수 있어야 한다.

그리고 전체가 보안상 안전해야 된다. 이 의미는 두 개의 네트워크 트래픽을 암호화해야 된다는 것이다.

두 네트워크 사이에 VPN 을 생성하려면 다음과 같이 여러 단계가 필요하다:

① 인터넷으로 연결되어 있는 두 개의 네트워크 사이에 “가상” 네트워크를 생성한다. 제대로 동작하는지 ping(8) 같은 툴을 사용하여 테스트한다.

② 필요하다면 두 네트워크 사이의 트레픽을 암호화하고 복호화하는 보안 정책을 적용한다. 이 트래픽이 암호화되었는지 tcpdump(1) 같은 툴을 사용하여 테스트한다.

③ Windows 머신이 VPN 을 통해 다른 네트워크를 볼 수 있도록 FreeBSD 게이트웨이에 부가적인 소프트웨어를 설정한다.

14.10.3.1 단계 1: “가상” 네트워크 생성 및 테스트

네트워크 #1 의 게이트웨이 머신에 로그인해서 IP 주소 W.X.Y.Z 머신의 사설 주소로 ping 192.168.2.1 을 실행한다. 이 명령이 동작하려면 무엇이 필요한가?

① 게이트웨이 머신은 192.168.2.1 에 어떻게 도달하는지 알아야 된다. 다시 말해 192.168.2.1 의 라우트를 알아야 된다.

② 192.168.x 같은 사설 IP 주소 범위는 인터넷에 나타나지 않는다. 대신 192.168.2.1 로 보내는 각 패킷은 다른 패킷에 감싸진다. 이 패킷은 A.B.C.D 에서 나타나서 W.X.Y.Z 로 보내져야 한다. 이 절차를 *캡슐화*(encapsulation)라고 한다.

③ 이 패킷이 W.X.Y.Z 에 도달하면 *캡슐해제*(unencapsulated)하여 192.169.2.1 에 전송해야 된다.

두 개의 네트워크 사이에 "터널"이 필요하다고 생각할 수 있다. 두 "터널 입구"는 IP 주소 A.B.C.D 와 W.X.Y.Z 이고 사설 IP 주소가 거쳐갈 수 있도록 알려줘야 한다. 터널은 공인 인터넷을 거쳐 사설 IP 주소로 트래픽을 전송한다.

이 터널은 일반적인 인터페이스나 FreeBSD 의 **gif** 장치를 사용하여 생성된다. 각 게이트웨이 호스트의 gif 인터페이스는 4 개의 IP 주소로 설정된다고 생각할 수 있다; 공인 IP 주소 두 개와 사설 IP 주소 두 개.

gif 장치를 지원하도록 양쪽 머신의 FreeBSD 커널에 컴파일해야 된다. 양쪽 머신의 커널 설정파일에 다음 라인을 추가하고 컴파일하여 설치한 후 재 부팅한다.

```
pseudo-device gif
```

커널 설정은 두 단계로 이루어진다. 첫째, 터널은 gifconfig(8)을 사용하여 외부 IP 주소(공인)가 무엇인지 알려줘야 한다. 그리고 사설 IP 주소는 ifconfig(8)을 사용하여 설정해야 된다.

네트워크 #1 의 게이트웨이 머신에서 터널을 설정하기 위해 다음 두 명령을 실행한다.

```
# gifconfig gif0 A.B.C.D W.X.Y.Z
# ifconfig gif0 inet 192.168.1.1 192.168.2.1 netmask 0xffffffff
```

다른 머신에서도 같은 명령을 실행하지만 IP 주소의 순서를 반대로 한다.

```
# gifconfig gif0 W.X.Y.Z A.B.C.D
```

```
# ifconfig gif0 inet 192.168.2.1 192.168.1.1 netmask 0xffffffff
```

그리고 다음 명령을 실행하여 설정을 볼 수 있다. 예를 들어 네트워크 #1 게이트웨이에서 다음과 같은 내용을 보게 된다:

```
# gifconfig gif0
gif0: flags=8011<UP,POINTTOPOINT,MULTICAST> mtu 1280
inet 192.168.1.1 --> 192.168.2.1 netmask 0xffffffff
physical address inet A.B.C.D --> W.X.Y.Z
```

위에서 보았듯이 터널은 물리적인 주소 A.B.C.D 와 W.X.Y.Z 사이에 생성되고 192.168.1.1 과 192.168.2.1 사이의 트래픽은 터널을 통과하게 된다.

여기서도 **netstat -rn** 으로 확인할 수 있는 양쪽 머신의 라우팅 테이블에 엔트리를 추가해야 된다. 다음은 네트워크 #1 의 게이트웨이 호스트를 출력한 것이다.

```
# netstat -rn
Routing tables

Internet:
Destination        Gateway        Flags      Refs      Use      Netif   Expire
...
192.168.2.1        192.168.1.1    UH           0          0      gif0
...
```

"Flags" 값이 보여주듯이 이것은 호스트 라우트다. 이 의미는 각 게이트웨이가 다른 게이트웨이에 어떻게 접근하는지 알고 있지만 각 네트워크의 내부에 어떻게 접근하는지 알지 못한다는 뜻이다. 이 문제는 금방 해결된다.

양쪽 머신에 방화벽을 운용하는 것과 비슷하게 VPN 트래픽을 우회시켜야 된다. 양쪽 네트워크 사이의 모든 트래픽을 허용하거나 VPN 한쪽 끝에서 다른 쪽 끝까지 방화벽 룰에 포함시킨다.

VPN 을 통하는 모든 트래픽을 허용하도록 방화벽을 설정했다면 테스트는 아주 쉽다. 게이트웨이 머신에 ipfw(8)을 사용한다면 다음 명령이 양쪽 VPN 사이의 모든 트래픽을 허용한다.

```
# ipfw add 1 allow ip from any to any via gif0
```

이 명령을 양쪽 게이트웨이 호스트에서 실행한다.

이것은 각 게이트웨이 머신이 서로 ping 을 허용하도록 한다. 192.168.1.1 에서 다음 명령을 실행해서 응답을 받을 수 있어야 되고 반대편 게이트웨이에서도 똑 같은 결과를 받아야 한다.

```
# ping 192.168.2.1
```

그러나 양쪽 네트워크의 내부 머신에는 아직 연결하지 못한다. 이유는 라우팅 때문이다. 게이트웨이 머신이 반대편에 어떻게 연결하는지 알고 있더라도 양쪽 게이트웨이 내부에 연결하는 방법은 알지 못한다.

이 문제를 해결하기 위해 각 게이트웨이 머신에 고정(static) 라우트를 추가해야 된다. 첫 번째 게이트웨이에서 고정 라우트를 추가하는 명령은 다음과 같다:

```
# route add 192.168.2.0 192.168.2.1 netmask 0xffffff00
```

이 명령은 “네트워크 192.168.2.0 에 있는 호스트에 연결하기 위해 호스트 192.168.2.1 에 패킷을 보낸다”. 다른 쪽 게이트웨이에서도 비슷한 명령을 실행해야 되지만 192.168.1.x 주소를 대신 사용한다.

IP 트래픽은 한쪽 네트워크의 호스트에서 다른 네트워크에 도달할 수 있다.

이제 두 네트워크 사이에 2/3 정도의 VPN 이 생성되었다. 아직 사설 쪽은 설정하지 않았다. ping(8)와 tcpdump(1)을 사용하여 이 설정을 테스트 한다. 게이트웨이 호스트에 로그인하여 다음 명령을 실행한다:

```
# tcpdump dst host 192.168.2.1
```

같은 호스트의 다른 로그인 세션에서는 다음 명령을 실행한다:

```
# ping 192.168.2.1
```

다음과 비슷한 결과를 보게 된다:

```
16:10:24.018080 192.168.1.1 > 192.168.2.1: icmp: echo request
16:10:24.018109 192.168.1.1 > 192.168.2.1: icmp: echo reply
16:10:25.018814 192.168.1.1 > 192.168.2.1: icmp: echo request
16:10:25.018847 192.168.1.1 > 192.168.2.1: icmp: echo reply
16:10:26.028896 192.168.1.1 > 192.168.2.1: icmp: echo request
16:10:26.029112 192.168.1.1 > 192.168.2.1: icmp: echo reply
```

결과에서 보았듯이 ICMP 메시지는 암호화되지 않은 체 되돌아온다. 패킷에서 더 많은 데이터 바이트를 캡쳐하기 위해 tcpdump(1)에 *-s* 매개변수를 사용하였다면 더 많은 정보를 볼 수 있다.

다음 섹션에서 두 네트워크 사이의 보안 링크를 설명하고 나면 모든 트래픽이 자동으로 암호화된다.

요약:

- "pseudo-device gif"로 양쪽 커널을 설정한다
- 게이트웨이 호스트 # 1 의 /etc/rc.conf 에 다음 라인을(필요하다면 IP 주소를 변경한다) 추가한다.

```
gifconfig_gif0="A.B.C.D W.X.Y.Z"
ifconfig_gif0="inet 192.168.1.1 192.168.2.1 netmask 0xffffffff"
static_routes="vpn"
route_vpn="192.168.2.0 192.168.2.1 netmask 0xffffff00"
```

- 양쪽 호스트의 방화벽 스크립트에서(/etc/rc.firewall 또는 비슷한) 다음 명령을 실행한다.

```
# ipfw add 1 allow ip from any to any via gif0
```

- IP 순서를 반대로해서 게이트웨이 호스트 #2 의 /etc/rc.conf 도 변경한다

14.10.3.2 단계 2: 보안 링크

안전한 링크를 위해 IPsec 를 사용한다. IPsec 는 두 개의 호스트에 동일한 암호화 키를 제공하고 두 호스트 사이의 데이터 암호화에 이 키를 사용하는 메커니즘을 제공한다.

여기서 고려해야 될 두 가지 설정 영역이 있다.

① 사용할 암호화 메커니즘에 동의하는 두 호스트를 위한 메커니즘이 있어야 한다. 이 메커니즘에 두 호스트가 동의한다면 이 둘 사이에는 “보안 관계”가 성립한다고 말한다.
② 어떤 트래픽을 암호화해야 되는지 지정하는 메커니즘이 있어야 한다. 외부로 나가는 모든 트래픽을 암호화하기를 원치 않을 것이다. VPN 의 일부 트래픽만 암호화하기를 원할 것이다. 어떤 트래픽을 암호화할지 결정하는 룰을 “보안 정책”이라고 한다.

보안 관계와 보안 정책은 커널이 관리하고 유저 기반 프로그램으로 수정할 수 있다. 그러나 사용하기 전에 IPsec 와 Encapsulated Security Paylod(ESP) 프로토콜을 지원하도록 커널을 설정해야 된다. 커널 설정파일에 다음 내용을 추가해서 컴파일하고 설치한 후 재 부팅하면 된다:

```
options IPSEC
options IPSEC_ESP
```

양쪽 게이트웨이 호스트의 커널을 위와 같이 설정해야 된다.

보안 관계를 설정할 때 두 가지를 선택할 수 있다. 두 호스트 사이의 암호화 알고리즘과 암호화 키 등을 선택하는 설정이나 Internet Key Exchange 프로토콜(IKE)을 실행하는 데몬을 사용할 수 있다.

개인적으로 후자를 권장하지만 개별적으로 설정하는 것도 어렵지 않다.

보안 정책을 편집하고 표시하려면 setkey(8)을 실행한다. route(8)이 커널의 라우팅 테이블이기 때문에 setkey 는 커널의 보안 정책 테이블이라고 유추할 수 있다. 또한 setkey 는 현재 보안 관계를 보여줄 수 있고 이러한 관점에서 netstat –r 과 유사하다.

FreeBSD 에서 보안 관계를 관리하는 여러 가지 데몬이 있다. 이 문서에서는 이들 데몬 중

하나인 racoon 을 어떻게 사용하는지 설명한다. racoon 은 FreeBSD 포트 컬렉션 security/ 카테고리에 있으며 일반적인 방법으로 설치한다.

racoon 을 양쪽 게이트웨이 호스트에서 실행해야 된다. 양쪽 호스트에서 VPN 의 반대편 IP 와 보안 키로(양쪽 게이트웨이에 동일한 키) 설정한다.

두 데몬은 각자 연결하여 서로 누구인지 확인한다(설정한 보안 키를 사용하여). 그리고 데몬은 새로운 보안 키를 생성해서 VPN 을 통한 트래픽 암호화에 이 키를 사용한다. 키를 정기적으로 변경하기 때문에 두 데몬이 다른 키를 선택했을 때 공격자가 이전 키를 크랙했더라도 큰 영향을 받지 않는다.

racoon 설정은 ${PREFIX}/etc/racoon 에 저장되므로 약간만 수정하면 되는 설정파일을 이곳에서 찾을 수 있다. 변경해야 되는 racoon 의 다른 컴포넌트는 “pre-shared key”다.

기본 racoon 설정은 이 키를 ${PREFIX}/etc/racoon/psk.txt 에서 찾는다. pre-shared key 는 VPN 링크를 통과하는 트래픽을 암호화할 때 사용하는 키가 아니고 키 관리 데몬이 다른 데몬과 신뢰하는데 사용하는 단순한 토큰이다.

psk.txt 에는 여러분이 관심을 가져야 되는 각 원격 사이트를 위한 라인을 가지고 있다. 이 예제에서는 상대 사이트를 위한 한 라인을(VPN 의 양쪽 단은 다른 쪽과 통신하기 때문이다) 각 psk.txt 파일에 가지고 있는 두 사이트가 있다.

게이트웨이 호스트 #1 의 라인은 다음과 비슷할 것이다:

```
W.X.Y.Z          secret
```

이것은 반대편의 *공인* IP 주소고 보안을 제공하는 텍스트 문자열이다. 여러분은 “secret”를 키로 사용하지 않는다.

```
A.B.C.D          secret
```

역시 같은 보안 키를 가진 반대편의 공인 IP 다. racoon 을 실행하기 전에 psk.txt 는 *0600* 모드여야(root 만 읽고 쓸 수 있도록) 된다.

양쪽 게이트웨이 머신에서 racoon 을 실행한다. ISAKMP(Internet Security Association Key

Management Protocol) 포트로 UDP 를 전송하는 IKE 트래픽을 허용하도록 방화벽 룰을 추가해야 된다.

```
# ipfw add 1 allow udp from A.B.C.D to W.X.Y.Z isakmp
# ipfw add 1 allow udp from W.X.Y.Z to A.B.C.D isakmp
```

racoon 이 실행되면 ping 을 한쪽 게이트웨이에서 다른 쪽으로 테스트할 수 있다. 연결은 아직 암호화되지 않지만 racoon 은 양쪽 호스트 사이에 보안 관계를 설정한다. 따라서 시간이 좀 소요될 것이고 ping 명령에 대한 응답을 받기 전에 약간 지체될 수 있다.

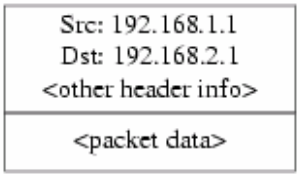
보안 관계가 설정되면 setkey(8)을 사용하여 설정된 것을 볼 수 있다. 보안 관계 정보를 보려는 호스트에서 setkey -D 를 실행한다.

이제 반은 해결하였다. 남은 문제는 보안 정책설정이다.

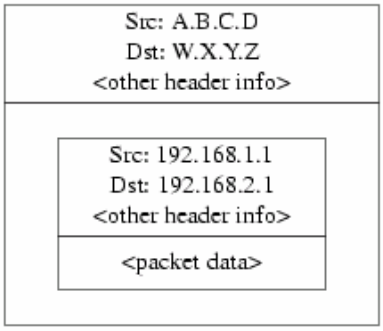
현명한 보안 정책을 수립하기 위해 설정한 것을 다시 확인한다

여러분이 보낸 각 IP 패킷에는 패킷에 대한 정보를 가지고 있는 헤더를 가지고 있다. 헤더는 소스와 목적지 IP 주소를 가지고 있다. 이미 알고 있듯이 192.168.x.y 와 같은 사설 IP 주소 범위는 공인 인터넷에 나타나지 않는다. 대신 이들 IP 는 다른 패킷에 캡슐화 (encapsulate) 된다. 이 패킷은 공인 소스와 목적지 IP 주소로 사설 주소를 대체한다.

외부로 나가는 패킷이 다음과 같이 생겼다면:



다음과 같이 다른 패킷으로 캡슐화된다:



이 캡슐화는 gif 장치로 수행한다. 그림에서처럼 패킷은 이제 실제 IP 를 가지고 외부로 나가고 최초의 패킷은 인터넷으로 나가는 패킷 속에 쌓여있다.

그리고 우리는 VPN 사이의 모든 트래픽이 암호화되기를 원한다. 이 말을 다시 설명하면:

"A.B.C.D 에서 출발하고 목적지가 W.X.Y.Z 라면 필요한 보안 관계를 사용하여 암호화시킨다"

"패킷이 W.X.Y.Z 에서 도착하고 목적지가 A.B.C.D 라면 필요한 보안 관계를 사용하여 복호화시킨다"

이 설정은 거의 근접했지만 정확하지는 않다. 이렇게 했다면 VPN 의 일부가 아닌 W.X.Y.Z 에서 출발하거나 W.X.Y.Z 에 도착하는 모든 트래픽이 암호화된다. 이 설정은 여러분이 원하는 것이 아니다. 정확한 정책은 다음과 같다:

"다른 패킷으로 캡슐화된 패킷이 A.B.C.D 에서 출발하고 목적지가 W.X.Y.Z 라면 필요한 보안 관계로 암호화한다"

"다른 패킷으로 캡슐화된 패킷이 W.X.Y.Z 에서 도착하고 목적지가 A.B.C.D 라면 필요한 보안 관계로 복호화한다"

약간의 변경이지만 필요한 요소다.

보안 정책도 setkey(8)을 사용하여 설정할 수 있다. setkey(8) 기능은 정책을 정의하기 위한 언어다. 표준입력으로 설정을 입력하거나 설정 정보를 가진 파일 이름을 지정하기 위해 *-f* 옵션을 사용할 수 있다.

W.X.Y.Z 로 나가는 모든 트래픽을 암호화시키는 게이트웨이 호스트 #1 의 설정은 다음과 같다:

```
spdadd A.B.C.D/32 W.X.Y.Z/32 ipencap -P out ipsec esp/tunnel/A.B.C.D-W.X.Y.Z/require;
```

위의 내용을 파일(예를 들어 /etc/ipsec.conf)에 넣고 다음 명령을 실행한다:

```
# setkey -f /etc/ipsec.conf
```

spdadd 는 보안 정책 데이터베이스에 룰을 추가하도록 setkey(8)에게 지시한다. 이 라인의 나머지는 이 정책에 어떤 패킷이 대응되는지 지정한다. A.B.C.D/32 와 W.X.Y.Z/32 는 IP 주소와 이 정책을 적용할 네트워크나 호스트를 확인하는 넷 마스크다. 우리는 이들 두 호스트 사이의 트래픽에 적용하기를 원한다. *ipencap* 는 다른 패킷을 캡슐화하는 패킷에만 이 정책을 적용하라고 커널에게 지시한다. *-P out* 은 외부로 나가는 패킷에만 이 정책을 적용하라는 것이고 *ipsec* 는 패킷이 안전하다는 것을 보여준다.

두 번째 라인은 이 패킷을 어떻게 암호화하는지 지정한다. *esp* 는 사용할 프로토콜이지만 *tunnel* 은 패킷이 IPsec 패킷으로 캡슐화된다는 것을 보여준다. 계속 사용되는 A.B.C.D 와 W.X.Y.Z 는 사용할 보안 관계를 선택하는데 사용되고 마지막 *require* 는 이 룰에 대응된다면 이 패킷을 암호화하라는 명령이다.

이 룰은 외부로 나가는 패킷에만 적용된다. 내부로 들어오는 페킷에도 비슷한 룰이 필요하다.

```
spdadd W.X.Y.Z/32 A.B.C.D/32 ipencap -P in ipsec esp/tunnel/W.X.Y.Z-A.B.C.D/require;
```

이 경우 *out* 대신 *in* 을 사용하고 IP 주소를 반대로 입력한다.

다른 게이트웨이 호스트에도(공인 IP 주소 W.X.Y.Z 를 가지고 있는) 비슷한 룰이 필요하다.

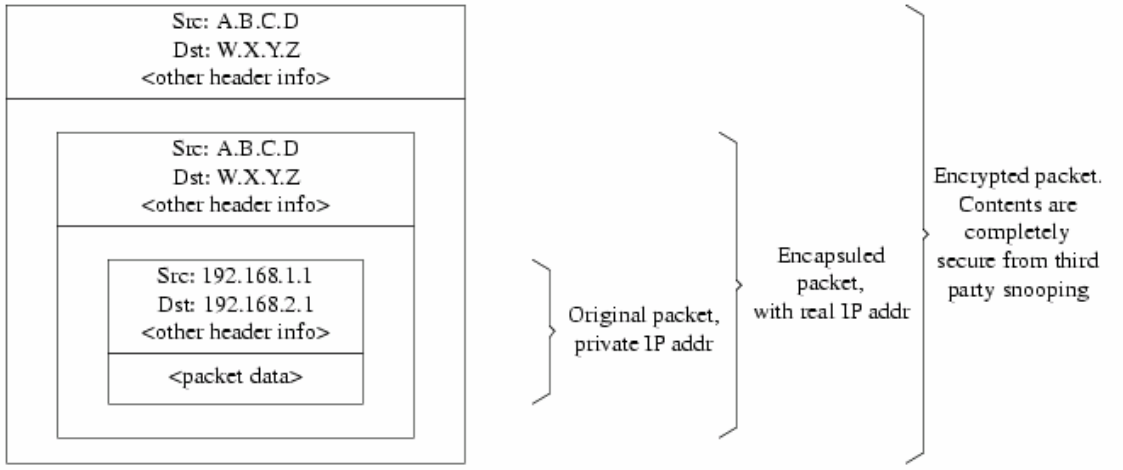
```
spdadd W.X.Y.Z/32 A.B.C.D/32 ipencap -P out ipsec esp/tunnel/W.X.Y.Z-A.B.C.D/require;

spdadd A.B.C.D/32 W.X.Y.Z/32 ipencap -P in ipsec esp/tunnel/A.B.C.D-W.X.Y.Z/require;
```

마지막으로 ESP 와 IPENCAP 패킷을 정방향과 역방향으로 허용하도록 방화벽에 룰을 추가한다. 이들 룰을 양쪽 호스트에 추가해야 된다.

```
# ipfw add 1 allow esp from A.B.C.D to W.X.Y.Z
# ipfw add 1 allow esp from W.X.Y.Z to A.B.C.D
# ipfw add 1 allow ipencap from A.B.C.D to W.X.Y.Z
# ipfw add 1 allow ipencap from W.X.Y.Z to A.B.C.D
```

룰이 대칭되기 때문에 각 게이트웨이 호스트에 같은 룰을 사용할 수 있다.

외부로 나가는 패킷은 이제 다음과 비슷할 것이다:



이들 패킷이 VPN 의 끝에 도착하면 복호화(racoon 으로 설정한 보안 관계를 사용하여)된다. 그리고 이들 패킷은 내부 네트워크에서 이동할 수 있는 패킷만 남을 때까지 두 번째 레이어를 벗겨내는 gif 인터페이스에 들어간다.

이전처럼 ping(8) 테스트를 사용하여 상태를 체크할 수 있다. A.B.C.D 게이트웨이 머신에 로그인하여 다음 명령을 실행한다:

```
# tcpdump dst host 192.168.2.1
```

같은 호스트의 다른 로그인 세션에서는 다음 명령을 실행한다:

```
# ping 192.168.2.1
```

이때 다음과 같은 결과를 볼 수 있다:

```
XXX tcpdump output
```

앞에서 보았듯이 tcpdump(1)가 ESP 패킷을 보여준다. *-s* 옵션을 사용했다면 암호화되었기 때문에 알 수 없는 문자를(외관상) 보게 된다.

축하한다! 두 개의 원격 사이트 사이에 VPN 을 설정했다.

요약

- 양쪽 커널을 다음 내용으로 설정한다:

```
options IPSEC
options IPSEC_ESP
```

- security/racoon을 설치한다. 양쪽 게이트웨이 호스트의 ${PREFIX}/etc/racoon/psk.txt 파일에 원격 호스트의 IP 주소 엔트리와 양쪽 호스트가 알고 있는 보안 키를 추가한다. 이 파일을 0600 모드로 변경한다.

- 각 호스트의 /etc/rc.conf 에 다음 라인을 추가한다:

```
ipsec_enable="YES"
ipsec_file="/etc/ipsec.conf"
```

- 필요한 spdadd 라인을 가지고 있는 /etc/ipsec.conf 파일을 각 호스트에 생성한다.

  게이트웨이 호스트 #1 에서는 다음과 같다:

```
spdadd A.B.C.D/32 W.X.Y.Z/32 ipencap -P out ipsec
   esp/tunnel/A.B.C.D-W.X.Y.Z/require;
spdadd W.X.Y.Z/32 A.B.C.D/32 ipencap -P in ipsec
   esp/tunnel/W.X.Y.Z-A.B.C.D/require;
```

게이트웨이 호스트 #2 는 다음과 같다:

```
spdadd W.X.Y.Z/32 A.B.C.D/32 ipencap -P out ipsec
   esp/tunnel/W.X.Y.Z-A.B.C.D/require;
spdadd A.B.C.D/32 W.X.Y.Z/32 ipencap -P in ipsec
   esp/tunnel/A.B.C.D-W.X.Y.Z/require;
```

- 양쪽 호스트 사이의 IKE, ESP 와 IPENCAP 트래픽을 허용하는 룰을 방화벽에 추가한다:

```
# ipfw add 1 allow udp from A.B.C.D to W.X.Y.Z isakmp
# ipfw add 1 allow udp from W.X.Y.Z to A.B.C.D isakmp
# ipfw add 1 allow esp from A.B.C.D to W.X.Y.Z
# ipfw add 1 allow esp from W.X.Y.Z to A.B.C.D
# ipfw add 1 allow ipencap from A.B.C.D to W.X.Y.Z
# ipfw add 1 allow ipencap from W.X.Y.Z to A.B.C.D
```

이전에 선행해야 되는 두 단계는 VPN 을 up 해서 실행할 수 있어야 된다. 각 네트워크에 있는 머신은 IP 주소를 사용하여 다른 쪽의 머신을 참조할 수 있어야 되고 링크를 통과하는 모든 트래픽은 자동으로 암호화된다.

14.11 OpenSSH

OpenSSH 는 원격 머신에 안전하게 접근하는데 사용하는 네트워크 연결 툴이다. 이 툴은 rlogin, rcp 와 telnet 을 대체할 수 있다. 게다가 다른 TCP/IP 연결도 SSH 를 통하여 안전하게 터널/포워드 할 수 있다. OpenSSH 로 암호화된 모든 트래픽은 감청되거나 연결 탈취와 다른 네트워크 레벨의 공격을 효과적으로 막을 수 있다.

OpenSSH 는 OpenBSD 프로젝트에 의해 관리되고 최근에 모든 버그가 수정된 SSH v1.2.12 기반으로 업데이트되었다. 이것은 SSH 프로토콜 1 과 2 에 호환된다. OpenSSH 는 FreeBSD 4.0 부터 기본 시스템에 포함되었다.

14.11.1 OpenSSH 사용의 이점

보통 telnet(1)이나 rlogin(1)을 사용할 때 데이터는 암호화되지 않은 평범한 텍스트 형태로 네트워크에 전송된다. 네트워크 스니퍼는 클라이언트 서버간 어느 곳에서든 유저/패스워드 정보나 데이터 전송을 캡쳐할 수 있다. OpenSSH 는 이런 일이 발생하지 않도록 다양한 인증과 암호화 방법을 제공한다.

14.11.2 sshd 활성화하기

rc.conf 파일에 다음 라인을 추가한다:

```
sshd_enable="YES"
```

시스템이 다음에 초기화될 때 OpenSSH 데몬 프로그램 sshd(8)을 로드한다. 아니면 명령어 라인에서 sshd 를 입력하여 직접 sshd 데몬을 실행할 수 있다.

14.11.3 SSH 클라이언트

ssh(1) 유틸리티는 간단히 rlogin(1)과 동작한다.

```
# ssh user@example.com
Host key not found from the list of known hosts.
Are you sure you want to continue connecting (yes/no)? yes
Host 'example.com' added to the list of known hosts.
user@example.com's password: *******
```

세션이 rlogin 이나 telnet 을 사용하여 만들어졌다면 로그인 세션은 계속 진행된다. SSH 는 클라이언트가 연결할 때 서버의 인증을 확인하기 위해 키 인증 시스템을 사용한다. 유저는 처음 연결할 때 프롬프트에 *yes* 만 입력하면 된다. 이 후의 모든 로그인은 저장된 인증 키로 확인된다. SSH 클라이언트는 나중에 로그인할 때 받은 인증키가 저장된 인증키와 다를 경우 경고를 준다. 인증키는 ~/.ssh/known_hosts 저장되거나 SSH v2 인증키는 ~/.ssh/known_hosts2 에 저장된다.

기본적으로 OpenSSH 서버는 SSH v1 와 SSH v2 연결을 허용하도록 설정되어있다. 그러나 클라이언트는 두 가지 중 하나를 선택할 수 있다. 버전 2 는 이전 버전보다 더 강력하고 안전하다.

ssh 는 -1 또는 -2 인수를 지정하여 v1 과 v2 프로토콜을 강제로 선택할 수 있다.

14.11.4 안전한 복사

scp(1) 명령은 rcp(1)와 비슷하게 동작한다; 보안을 제외하고 원격 머신에서 파일을 복사할 수 있다.

```
# scp user@example.com:/COPYRIGHT COPYRIGHT
user@example.com's password: *******
COPYRIGHT                     100% |*****************************|  4735
00:00
#
```

왜냐하면 인증키는 이전 예제에서 이미 호스트에 저장되어있기 때문에 여기서 scp(1)를 사용하면 바로 확인된다.

첫 번째 인자에 소스 파일을 지정하고 두 번째에 목적지 파일을 지정하는 scp(1) 인자는 cp(1)와 비슷하다. 왜냐하면 SSH 를 사용하는 네트워크를 거쳐 파일이 복사되므로 한 개 또는 더 많은 파일인자를 이 형식에 추가할 수 있다.
기본 형식은 다음과 같다.

```
user@host:<path_to_remote_file>.
```

14.11.5 설정

OpenSSH 데몬과 클라이언트 시스템의 다양한 설정파일은 /etc/ssh 디렉터리에 있다. ssh_config 는 클라이언트 설정이지만 sshd_configures 는 데몬 설정이다.

추가적으로 *sshd_program*(기본적으로 /usr/sbin/sshd)과 *sshd_flags* rc.conf 옵션은 더 많은 설정레벨을 제공한다.

14.11.6 ssh-keygen

패스워드를 사용하는 대신 ssh-keygen(1)으로 유저 인증에 사용하려는 RSA 키를 생성할 수 있다.

```
% ssh-keygen -t rsa1
Initializing random number generator...
Generating p:  .++ (distance 66)
Generating q:  ..............................++ (distance 498)
Computing the keys...
Key generation complete.
Enter file in which to save the key (/home/user/.ssh/identity):
Enter passphrase:
Enter the same passphrase again:
Your identification has been saved in /home/user/.ssh/identity.
...
```

ssh-keygen(1)은 인증에 사용되는 공개키와 개인키 쌍을 생성한다. 개인키는 ~/.ssh/identity 에 공개키는 ~/.ssh/identity.pub 에 저장된다. 공개키를 연결하려는 원격 머신의 ~/.ssh/authorized_keys 에 저장한다.

따라서 패스워드 대신 RSA 인증으로 원격 머신에 연결할 수 있다.

> Note: *-t rsa1* 옵션은 SSH 프로토콜 1 버전을 사용하도록 RSA 키를 생성한다. SSH 프로토콜 2 버전의 RSA 키를 사용하려면 **ssh-keyge -t rsa** 명령을 사용한다.

ssh-keygen(1)에 패스워드를 사용했다면 유저는 개인키를 사용하기 위해 매번 패스워드를 입력하는 프롬프트를 보게 된다.

SSH 프로토콜 버전 2 DSA 키는 같은 목적을 위해 **ssh-keygen -d** 명령으로 생성할 수 있다. 이 명령은 SSH 프로토콜 버전 2 세션만 사용하도록 공개/개인 DSA 키를 생성한다. 공개 키는 ~/.ssh/id_dsa.pub 에 개인키는 ~/.ssh/id_dsa 에 저장된다.

DSA 공개키는 원격 머신의 ~/.ssh/authorized_keys 에 있어야 된다.

ssh-agent(1)과 ssh-add(1)은 여러 개의 패스워드로 된 개인키 관리에 사용된다.

> **주의:** 시스템의 OpenSSH 버전에 따라 다양한 옵션과 파일을 지정할 수 있다. 문제를 예방하기 위해 ssh-keygen(1) 매뉴얼 페이지를 참고한다.

14.11.7 SSH 터널링

OpenSSH 는 암호화 세션에서 다른 프로토콜을 캡슐화하는 터널을 생성할 수 있다.
다음 명령이 ssh(1)가 telnet 터널을 생성하는 것을 보여준다.

```
% ssh -2 -N -f -L 5023:localhost:23 user@foo.example.com
%
```

ssh 명령은 다음 옵션들과 사용된다.

-2

강제로 ssh 프로토콜 버전 2 를 사용한다(예전의 SSH 서버에는 사용하지 않는다).

-N

명령이 없고 터널만 있음을 명시한다. 생략했다면 ssh 는 일반 세션을 시작한다.

-f

강제로 ssh 를 백그라운드에서 실행한다.

-L

로컬 터널을 *localport:remotehost:remoteport* 형식으로 보여준다.

user@foo.example.com

원격 SSH 서버를 의미한다.

SSH 터널은 loadlhost 의 지정된 포트에 listen 소켓을 생성한다. 그리고 로컬 호스트/포트에 SSH 를 통해 요청되는 모든 연결을 지정된 원격 호스트와 포트로 포워드한다.

예제에서 loacalhost 의 포트 *5023* 은 원격 머신의 포트 *23* 번으로 포워드된다.
23 번은 **telnet** 이기 때문에 SSH 터널을 통해 보안 **telnet** 세션을 생성한다.

이 방법으로 SMTP, POP3, FTP 등과 같은 비 보안 TCP 프로토콜을 감쌀 수 있다.

예제 14-1. SMTP 보안 터널 생성에 SSH 사용하기

```
% ssh -2 -N -f -L 5025:localhost:25 user@mailserver.example.com
user@mailserver.example.com's password: *****
% telnet localhost 5025
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 mailserver.example.com ESMTP
```

이 명령을 ssh-keygen(1)와 사용하여 더욱 깨끗하고/안전한 SSH 터널링 환경에서 유저 계정을 추가할 수 있다. 패스워드를 입력하는 대신 키를 사용할 수 있고, 터널은 각 유저 단위로 실행할 수 있다.

[예제: 실질적인 SSH 터널링 예제]

1. POP3 서버에 안전하게 접근하기

외부의 접근을 허용하는 SSH 서버가 사무실에 있다. 같은 사무실 내부 네트워크에 POP3

서버가 운용되는 메일 서버가 있다. 로컬 네트워크나 여러분의 집과 사무실 사이의 네트워크 경로는 완벽하게 신뢰할 수 있거나 신뢰할 수 없을것이다. 안전하게 메일을 체크하려면 사무실의 SSH 서버와 SSH 로 연결해서 터널을 거쳐 메일 서버에 도달할 수 있다.

```
% ssh -2 -N -f -L 2110:mail.example.com:110 user@ssh-server.example.com
user@ssh-server.example.com's password: ******
```

터널이 실행되고있을 때 메일 클라이언트의 POP3 요청을 localhost 포트 2110 으로 지정하여 보낼 수 있다. 이곳의 연결은 안전하게 터널을 거쳐 mail.example.com 으로 포워드된다.

2. 엄격한 방화벽 우회하기

어떤 네트워크 관리자는 들어오는 연결과 나가는 연결을 필터링하는 아주 강력한 방화벽 룰을 정하여 적용한다. 외부로 연결되는 것은 오직 포트 22 번과 80 번으로 SSH 와 웹 서핑만할 수 있다.

음악을 듣기 위해 Ogg Vorbis 서버 같은 서비스(업무와 관련 없는)를 사용하고 싶을 것이다. 이 Ogg Vorbis 서버가 22 이나 80 이 아닌 다른 포트로 음악을 서비스한다면 사용할 수 없을 것이다.

해결책은 방화벽 외부 머신에 SSH 터널을 생성해서 이 터널을 이용하여 Ogg Vorbis 서버를 사용한다.

```
% ssh -2 -N -f -L 8888:music.example.com:8000 user@unfirewalled-system.example.org
user@unfirewalled-system.example.org's password: *******
```

이제 music.example.com 의 8000 포트로 포워드하는 localhost 8888 포트로 클라이언트 프로그램을 지정하여 방화벽을 우회할 수 있다.

14.11.8 더 읽을만한 내용

OpenSSH(http://www.openssh.com/)

ssh(1) scp(1) ssh-keygen(1) ssh-agent(1) ssh-add(1)

sshd(8) sftp-server(8)

14.12 파일시스템 접근 제어 리스트

스냅샷과 같은 파일시스템의 증가로 FreeBSD 5.0 과 이후 버전은 파일시스템 접근제어 리스트(ACL) 보안을 제공한다.

접근제어 리스트는 높은 호환성으로(POSIX®.1e) 표준 유닉스 퍼미션 모델을 확장한다. 이 기능으로 관리자는 더욱 강력한 보안 모델의 이점을 얻을 수 있다.

UFS 파일시스템에 ACL 을 지원하려면 다음 라인을 커널에 컴파일해야 된다:

```
options UFS_ACL
```

이 옵션을 컴파일하지 않고 ACL 을 지원하는 파일시스템을 마운트하려고 하면 경고 메시지가 나타난다. 이 옵션은 GENERIC 커널에 포함되어있다. ACL 은 파일시스템에서 확장된 기능이다. 확장된 기능은 차세대 유닉스 파일시스템 UFS2 에서 자연스럽게 지원된다.

ACL 은 /etc/fstab 에 추가해야 되는 mount-time 관리 플래그로(*acls*) 활성화된다. mount-time 플래그는 파일시스템 헤더의 슈퍼블록 ACL 플래그를 수정하는 tunefs(8)을 사용하여 자동으로 영구히 설정할 수도 있다. 보통 여러 가지 이유에서 슈퍼블록 플래그를 사용하는 것이 더 선호된다:

- mount-time ACL 플래그는 remount 로(mount(8) *-u*) 변경할 수 없고 완벽하게 umount(8)한 후 새로 mount(8)해야 된다. 이 의미는 부팅 후 ACL 을 root 파일시스템에서 활성화할 수 없다는 것이다. 그리고 한번 사용되면 파일시스템의 배열을 변경할 수도 없다.

- 슈퍼블록 플래그를 설정하면 fstab 엔트리가 없거나 장치가 재 배열되더라도 항상 ACL 이 활성화된 체 파일시스템이 마운트된다. 이 방법은 ACL 이 활성화되지 않은 파일시스템이 갑자기 마운트되어 ACL 상태가 적절하지 않아서 유발할 수

있는 보안 문제를 방지한다.

Note: 완벽하게 새로 mount(8)하지 않아도 플래그를 허용하도록 ACL 동작을 변경할 수 있지만 ACL 이 정확히 활성화되지 않고 갑자기 마운트되면 도끼로 발등 찍히는 일이 될 수 있다. 따라서 ACL 이 활성화됐다면 비활성하고 확장된 속성을 그대로 두고 다시 활성화한다. 일반적으로 파일시스템에서 ACL 을 활성화했다면 다시 비활성하지 않아야 된다. 왜냐하면 파일 보호가 시스템의 유저와 호환되지 않게 되고 ACL 을 다시 활성화하면 퍼미션이 변경된 후의 이전 ACL 이 파일에 추가되어 결과를 예측할 수 없게 된다.

ACL 이 활성화된 파일시스템은 다음과 같이 파일 퍼미션 설정을 볼 때 +(플러스) 표시로 나타난다.

```
drwx------  2 robert  robert  512 Dec 27 11:54 private
drwxrwx---+ 2 robert  robert  512 Dec 23 10:57 directory1
drwxrwx---+ 2 robert  robert  512 Dec 22 10:20 directory2
drwxrwx---+ 2 robert  robert  512 Dec 27 11:57 directory3
drwxr-xr-x  2 robert  robert  512 Nov 10 11:54 public_html
```

여기 보고 있는 directory1, directory2 와 directory3 디렉터리는 모두 ACL 의 이점을 가지고 있다. public_html 디렉터리는 아니다.

14.12.1 ACL 사용

파일시스템 ACL 은 getfacl(1) 유틸리티로 볼 수 있다. 예를 들면 test 파일의 ACL 설정을 보려면 다음 명령을 사용한다:

```
% getfacl test
    #file:test
    #owner:1001
    #group:1001
    user::rw-
    group::r--
    other::r--
```

이 파일의 ACL 설정을 변경하려면 setfacl(1) 유틸리티를 실행한다.

```
% setfacl -k test
```

-k 옵션은 파일이나 파일시스템에 현재 설정되어 있는 모든 ACL 을 삭제한다. 좀더 바람직한 방법은 ACL 의 기본 필드는 남겨두는 *-b* 를 사용한다.

```
% setfacl -m u:trhodes:rwx,group:web:r--,o::--- test
```

앞서 설명한 명령의 *-m* 옵션은 기본 ACL 엔트리를 수정할 때 사용된다. 이전 명령으로 모든 정의가 삭제되어 이전에 정의된 엔트리가 없기 때문에 이 명령은 기본 옵션을 복구하고 나열된 옵션을 할당한다. 시스템에 없는 유저나 그룹을 추가한다면 “Oinvalid argument” 에러가 표준 출력으로 표시되므로 주의한다.

10.13 FreeBSD 보안 권고문

다른 운영체제처럼 FreeBSD 도 “보안 권고문”을 발행한다. 보통 이들 권고문은 메일을 사용하여 통보되고 적절한 릴리즈가 패치 된 후 Errata 에 보관된다. 이번 섹션에서는 권고문이 무엇이고 어떻게 이해해야 되며 시스템을 패치하기 위해 무엇이 필요한지 설명한다.

10.13.1 권고문은 어떻게 생겼는가?

FreeBSD 보안 권고문은 아래와 비슷하게 생겼고 FreeBSD 보안 공고 메일링 리스트에서 (http://lists.freebsd.org/mailman/listinfo/freebsd-security-notifications) 볼 수 있다.

FreeBSD-SA-XX:XX.UTIL Security Advisory

The FreeBSD Project

Topic: denial of service due to some problem ❶

Category: core ❷

Module: sys ❸

Announced: 2003-09-23 ❹

Credits: Person@EMAIL-ADDRESS ❺

Affects: All releases of FreeBSD ❻

FreeBSD 4-STABLE prior to the correction date

Corrected: 2003-09-23 16:42:59 UTC (RELENG_4, 4.9-PRERELEASE)

2003-09-23 20:08:42 UTC (RELENG_5_1, 5.1-RELEASE-p6)

2003-09-23 20:07:06 UTC (RELENG_5_0, 5.0-RELEASE-p15)

2003-09-23 16:44:58 UTC (RELENG_4_8, 4.8-RELEASE-p8)

2003-09-23 16:47:34 UTC (RELENG_4_7, 4.7-RELEASE-p18)

2003-09-23 16:49:46 UTC (RELENG_4_6, 4.6-RELEASE-p21)

2003-09-23 16:51:24 UTC (RELENG_4_5, 4.5-RELEASE-p33)

2003-09-23 16:52:45 UTC (RELENG_4_4, 4.4-RELEASE-p43)

2003-09-23 16:54:39 UTC (RELENG_4_3, 4.3-RELEASE-p39) ❼

FreeBSD only: NO ❽

For general information regarding FreeBSD Security Advisories,
including descriptions of the fields above, security branches, and the
following sections, please visit
http://www.FreeBSD.org/security/.

I. Background ❾

II. Problem Description ❿

❶ *Topic* 필드는 문제가 정확히 무엇인지 보여준다. 기본적으로 현재 보안 권고안을 설명하고 취약점과 유틸리티를 표시한다.

❷ *Category* 는 문제가 되는 시스템 부분(*core*, *contrib* 또는 *ports* 가 될 수 있다)을 알려준다. *core* 카테고리의 의미는 취약점이 FreeBSD 운영체제의 핵심 컴포넌트에 영향을 미친다는 것이다. *contrib* 카테고리의 의미는 취약점이 sendmail 처럼 FreeBSD 프로젝트에 의해 작성된 소프트웨어에 영향을 미친다는 것이다. 마지막으로 *ports* 카테고리는 포트 컬렉션으로 사용할 수 있는 소프트웨어에 영향을 미친다는 것이다.

❸ *Module* 필드는 예를 들어 *sys* 처럼 컴포넌트 위치를 설명한다. 이 예제에서는 우리가 보고 있는 *sys* 에 영향을 받기 때문에 이 취약점은 커널이 사용하는 컴포넌트에 영향을 끼친다.

❹ *Announced* 필드는 보안 권고가 발행되거나 세계로 발표된 날짜를 반영한다. 이 의미는 보안팀이 이 문제를 확인했고 이 패치가 FreeBSD 소스 저장소에 반영됐다는 것이다.

❺ *Credits* 필드는 취약점을 발견해서 보고한 개인이나 단체를 명시해준다.

❻ *Affects* 필드는 이 취약점이 어떤 FreeBSD 릴리즈에 영향을 주는지 설명한다. 커널의 경우 영향을 받는 파일의 ident 결과를 빨리 조회하면 버전을 확인하기 쉽다. 포트는 /var/db/pkg 의 포트이름 뒤에 나열된 버전번호를 확인한다. 시스템을 FreeBSD CVS 저장소와 동기화되지 않고 한번씩 다시 빌드했다면 확인할 수 없다.

❼ *Corrected* 필드는 날짜와 시간 그리고 시간 옵셋과 일치하는 릴리즈를 표시한다.

❽ *FreeBSD only* 필드는 이 취약점이 FreeBSD 에만 영향을 주는지 또는 다른 운영체제에도 영향을 주는지 표시한다.

❾ *Background* 필드는 정확히 영향을 받는 유틸리티가 무엇인지 알려준다. 이 유틸리티가 왜 FreeBSD 에 있으며 무엇에 쓰는지 그리고 개발된 배경에 대한 간략한 정보를 준다.

❿ *Problem Description* 필드는 보안 문제에 대한 더 자세한 정보를 제공한다. 여기서는 결함이 있는 코드나 어떻게 이 유틸리티가 보안 문제를 유발하는 악의적인 목적에 사용되었는지에 대한 정보를 포함한다.

(11) *Impact* 필드는 어떤 종류의 영향을 시스템에 끼칠 수 있는지 설명한다. 예를 들어 서비스 거부 공격, 유저가 더 많은 권한 이용가능, 공격자의 슈퍼유저 권한획득 등이 포함된다.

(12) *Workaround* 필드는 시스템을 업그레이드할 수 없는 시스템 관리자가 문제를 해결할 수 있는 방법을 제공한다. 이것은 시간적인 제약이나 네트워크 유용성 또는 다른 많은 이유로 인한 것이다. 어떤 경우든 보안을 가볍게 치부하면 안되고 영향을 받는 시스템은 패치하거나 보안 문제를 해결해야 된다.

(13) *Solution* 필드에서는 시스템 패치를 설명한다. 이것은 단계적으로 테스트되었고 시스템을 패치하여 보안 문제를 검증한다.

(14) *Correction Details* 필드는 문자에 밑줄을 쳐서 변경된 기간별로 CVS 분기나 릴리즈 이름을 표시한다. 각 분기별로 영향을 받는 파일의 수정된 횟수도 보여준다.

(15) *References* 필드는 소스의 다른 정보를 제공한다. 이곳에는 웹 URL, 책, 메일링 리스트와 뉴스그룹을 포함할 수 있다.