

PAM이란?

목차

목차.....	0
1. Solaris.....	3
1.1 PAM 개요.....	3
1.2 PAM 동작 원리.....	4
1.3 PAM 사용 이점.....	5
1.4 PAM 파일과 위치.....	5
1.5 PAM 라이브러리.....	5
1.6 PAM 구성 파일.....	6
1.7 PAM 구성 검색 순서.....	7
1.8 PAM 구성 파일 구문.....	7
1.9 PAM 스택.....	9
1.10 PAM 스택 사용예시.....	13
1.11 SSH(Secure Shell).....	13
1.12 PAM 프로그래밍.....	19
1.13 PAM 컴파일 방법.....	20
1.14 PAM 디버그 사용.....	21
1.15 PAM 테스트 방법.....	22
2. FreeBSD.....	23
2.1 PAM 개요.....	23
2.2 PAM 동작 원리.....	24
2.3 PAM 사용 이점.....	25
2.4 PAM 파일과 위치.....	25
2.5 PAM 라이브러리.....	25
2.6 PAM 설정하기.....	26
2.7 PAM 사용예시(일반).....	28
2.8 PAM 사용예시(/etc/pam.d/su).....	29
2.9 PAM 사용예시(/etc/pam.d/system-auth).....	30
2.10 PAM 사용예시(/etc/pam.d/sshd).....	32
2.11 SSH(Secure Shell).....	32
2.12 PAM 프로그래밍.....	38
2.13 PAM 컴파일 방법.....	39

2.14 PAM 환경설정.....	40
2.15 PAM 디버그 사용.....	40
2.16 PAM 테스트 방법.....	41
3. HP-UX.....	42
3.1 PAM 개요.....	42
3.2 PAM 동작 원리.....	43
3.3 PAM 사용 이점.....	44
3.4 PAM 파일과 위치.....	44
3.5 PAM 라이브러리.....	44
3.6 PAM 구성 파일.....	45
3.7 PAM 이 로그인에 대해 작동하는 방식.....	48
3.8 SSH(Secure Shell).....	49
3.9 PAM 프로그래밍.....	55
3.10 PAM 컴파일 방법.....	57
3.11 PAM 디버그 사용.....	57
3.12 PAM 테스트 방법.....	58
4. AIX.....	59
4.1 PAM 개요.....	59
4.2 PAM 동작 원리.....	60
4.3 PAM 사용 이점.....	61
4.4 PAM 파일과 위치.....	61
4.5 PAM 라이브러리.....	61
4.6 PAM 모듈.....	62
4.7 PAM 구성 파일.....	64
4.8 pam_aix 모듈.....	66
4.9 PAM 모듈 추가.....	68
4.10 /etc/pam.conf 파일 변경.....	68
4.11 PAM 디버그 사용.....	69
4.12 SSH(Secure Shell).....	70
4.13 PAM 프로그래밍.....	76
4.14 PAM 컴파일 방법.....	77
4.15 PAM 테스트 방법.....	78
5. Linux.....	79
5.1 PAM 개요.....	79
5.2 PAM 동작 원리.....	80
5.3 PAM 사용 이점.....	81
5.4 PAM 파일과 위치.....	81

5.5 PAM 라이브러리.....	81
5.6 PAM 설정하기.....	82
5.7 PAM 사용예시(일반).....	84
5.8 PAM 사용예시(/etc/pam.d/su).....	85
5.9 PAM 사용예시(/etc/pam.d/system-auth).....	86
5.10 PAM 사용예시(/etc/pam.d/sshd).....	88
5.11 SSH(Secure Shell).....	88
5.12 PAM 프로그래밍.....	94
5.13 PAM 컴파일 방법.....	96
5.14 PAM 환경설정.....	96
5.15 PAM 디버그 사용.....	97
5.16 PAM 테스트 방법.....	97
6. About BaroPAM.....	99

1. Solaris

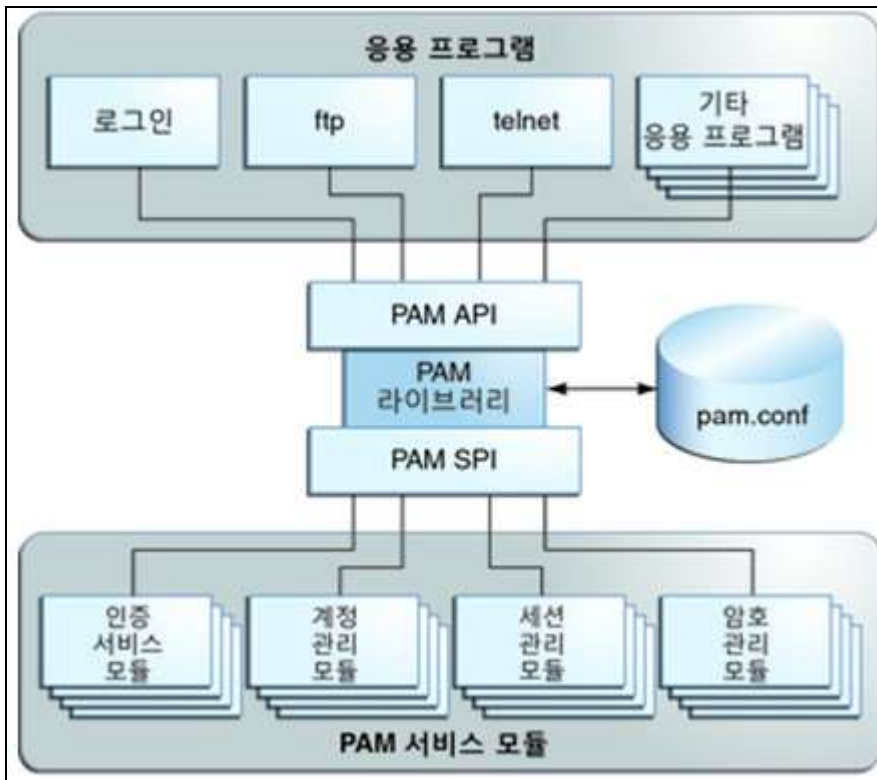
1.1 PAM 개요

PAM(Pluggable Authentication Module, 플러그 가능한 인증 모듈)은 Linux/Unix 시스템에서 서비스를 재 컴파일하지 않고, 다양한 인증 기술을 시스템 항목 서비스에 접목할 수 있도록 해주는 프레임워크로 중앙 집중적인 인증 매커니즘을 지원하는 것이다. 게다가 시스템의 기본적인 인증 기법을 제공하여 이것을 사용하면 응용 프로그램 개발자 뿐만 아니라 시스템 관리자들이 인증을 유연성 있게 관리할 수 있도록 도와 준다.

전통적으로 시스템 자원에 대한 접근을 관리하는 프로그램들은 내장된 매커니즘에 의해 사용자 인증 과정을 수행한다. 이러한 방식은 오랫동안 이루어졌지만 이러한 접근 방식은 확장성이 부족하고 매우 복잡하다. 그렇기 때문인지 이러한 인증 매커니즘을 끌어내기 위한 수많은 해킹 시도가 있었다.

Solaris의 방식을 따라서 Unix/Linux 사용자들은 그들만의 PAM을 구현하는 방식을 찾았다.

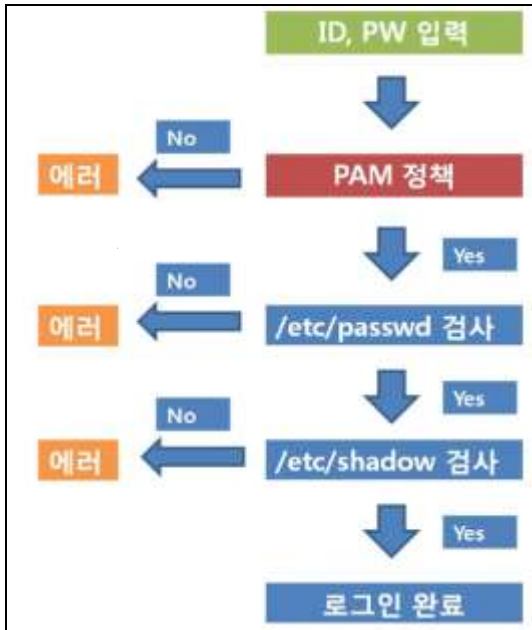
PAM의 아키텍처는 다음과 같다.



PAM의 기본 원리는 응용 프로그램이 password 파일을 읽어 오는 대신 PAM이 직접 인증을 수행 하도록 하는 것이다. PAM은 시스템 관리자가 원하는 인증 매커니즘이 무엇이든 상관하지 않는다.

여러 사이트에서 선택 받은 인증 매커니즘은 아직도 password 파일이다. 왜 그럴까? 우리가 원하는 것을 해주기 때문이다. 대부분의 사용자는 password 파일이 필요한 것이 무엇인지 이미 알고 있다. 그리고 원하는 작업을 수행하는데 있어 이미 그 기능이 검증되었기 때문일 것이다.

PAM의 인증 절차는 다음과 같다.



1.2 PAM 동작 원리

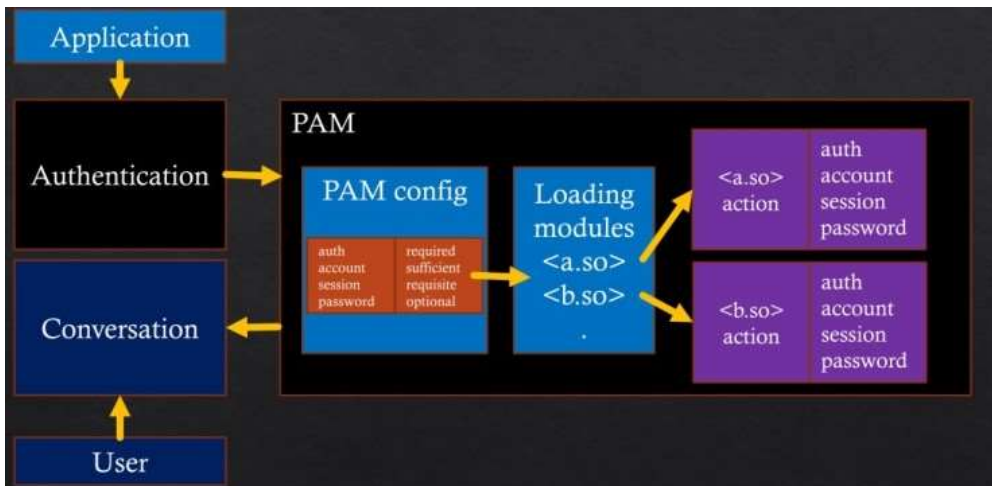
PAM은 Windows 환경의 DDL(Dynamic Link Library)과 같은 것이다. Library로 프로그램이 어떤 사용자에게 대한 인증을 수행하려면 PAM Library가 있는 함수를 호출한다. PAM은 해당 함수의 Library를 제공하여 응용 프로그램이 특정 사용자를 인증하도록 요청할 수 있다.

PAM을 통해 /etc/passwd 파일이나 /etc/shadow 파일을 확인하거나 또는 보다 복잡한 확인 작업을 수행하기도 하는데, 예로는 LDAP 서버에 접속하는 것이다.

확인 작업을 마치고, 인증 여부를 결정하게 되면 "인증됨/인증되지 않음"의 메시지를 자신을 호출한 응용 프로그램에 전송한다.

간단한 확인 작업이라고 했는데, 그 과정이 많아 보일 수가 있다. 여기에 나오는 각 모듈은 크기가 작고 작업 수행 시간이 매우 빠르다. 이것은 매우 놀랍기도 하지만 PAM을 사용하게 된다.

PAM의 동작은 프로그램의 실행 하자마자 실행되는 보안 설정 파일들의 프로그램들이다.



PAM의 동작 원리는 다음과 같다.

- 인증이 필요한 프로그램 동작 시 PAM 라이브러리 호출
- PAM설정 파일을 참조
- 참조한 파일의 내용을 바탕으로 모듈 동작
- 프로그램에 동작 결과를 반환하여 인증여부 결정.

1.3 PAM 사용 이점

PAM을 사용하면 사용자 인증을 위한 시스템 항목 서비스(예: ftp, login, telnet, rsh) 사용을 구성할 수 있다. PAM이 제공하는 몇 가지 이점은 다음과 같다.

- 유연한 구성 정책
- 응용 프로그램별 인증 정책
- 기본 인증방식을 선택할 수 있는 기능
- 높은 보안 시스템에서 여러 권한 부여를 요구할 수 있는 기능
- 최종 사용자의 사용 편의성
- 암호가 여러 인증 서비스에 대해 동일한 경우 암호 재입력 없음
- 사용자가 여러 명령을 입력할 필요 없이 여러 인증 서비스에 대해 사용자에게 암호를 요구할 수 있는 기능
- 사용자 인증 서비스에 선택적 옵션을 전달할 수 있는 기능
- 시스템 항목 서비스를 변경할 필요 없이 사이트별 보안 정책을 구현할 수 있는 기능

1.4 PAM 파일과 위치

파일 위치	설명	비고
/usr/lib/security	제공 가능한 PAM 모듈 디렉토리로 실제 PAM Library를 동적으로 인증 모듈을 호출하여 실행한다. (*so)	
/etc/security/unix	PAM 실행에 필요한 설정파일로 /usr/lib/security에 있는 모듈에 대한 설정 파일. (서비스명.conf)	
/etc/pam.d	PAM 데몬 파일(애플리케이션별 PAM의 설정 파일 위치)로 PAM을 사용하는 응용 프로그램이 설정 파일이 없다면 기본값으로 설정된 설정 파일을 자동으로 사용한다.	

1.5 PAM 라이브러리

라이브러리	설명	비고
pam_allow	모든 호출에 대해 PAM_SUCCESS를 반환한다.	
pam_authtok_check	암호 변경에 대해 암호 토큰의 유효성을 검사한다.	
pam_authtok_get	PAM 스택에 암호 프롬프트 기능을 제공한다.	
pam_authtok_store	PAM_USER에 대한 암호 토큰을 업데이트한다.	
pam_deny	모든 호출에 대해 모듈 유형 기본 실패 반환 코드를 반환한다.	
pam_dhkeys	두 PAM 서비스에 보안 RPC 인증 및 보안 RPC 인증 토큰 관리 기능을 제공한	

	다.	
pam_krb5	Kerberos 사용자의 ID를 확인하고 Kerberos 자격 증명 캐시를 관리하는 기능을 제공한다.	
pam_krb5_migrate	PAM_USER를 클라이언트의 로컬 Kerberos 영역으로 마이그레이션 하는데 도움을 준다.	
pam_ldap	구성된 LDAP 디렉토리 서버를 통한 PAM 인증 및 계정 관리 스택에 필요한 기능을 제공한다.	
pam_list	이 호스트에서 사용자 계정의 유효성을 검사하는 기능을 제공한다. 이 유효성 검사는 호스트에 있는 사용자 및 넷 그룹 목록을 기반으로 한다.	
pam_passwd_auth	암호 스택에 인증 기능을 제공한다.	
pam_pkcs11	사용자가 X.509 인증서 및 PKCS#11 토큰에 저장된 해당 전용 개인 키를 사용하여 시스템에 로그인할 수 있도록 해준다.	
pam_roles	사용자가 역할을 맡을 권한이 부여되었는지 확인하고 역할에 의한 직접 로그인을 금지한다.	
pam_smb_passwd	로컬 Oracle Solaris 사용자에게 대한 SMB 암호 변경 또는 추가를 지원한다.	
pam_smbfs_login	Oracle Solaris 클라이언트와 해당 CIFS/SMB 서버 간에 암호를 동기화한다.	
pam_tsol_account	레이블과 관련된 Trusted Extensions 계정 제한을 확인한다.	
pam_tty_tickets	이전 인증 성공으로 생성된 티켓을 검사하기 위한 방식을 제공한다.	
pam_unix_account	사용자 계정이 잠기거나 만료되지 않았는지 확인하고 사용자 암호를 변경할 필요가 없음을 확인하는 기능을 제공한다. access_times 및 access_tz에 대한 검사를 포함한다.	
pam_unix_auth	암호가 PAM_USER에 대해 올바른 암호인지 확인하는 기능을 제공한다.	
pam_unix_cred	사용자 자격 증명 정보를 설정하는 기능을 제공한다. 자격 증명 기능과는 독립적으로 인증 기능을 대체할 수 있도록 해준다.	
pam_unix_session	세션을 열거나 닫고 /var/adm/lastlog 파일을 업데이트한다.	
pam_user_policy	사용자별 PAM 구성을 호출한다.	
pam_zfs_key	사용자의 암호화된 홈 디렉토리에 대한 ZFS 암호화 문장암호를 로드 및 변경하는 기능을 제공한다.	

1.6 PAM 구성 파일

login 및 ssh와 같이 PAM 프레임워크를 사용하는 시스템 응용 프로그램은 /etc/pam.d 디렉토리에 있는 PAM 구성 파일에 구성된다. /etc/pam.conf 파일도 사용될 수 있다. 이러한 파일을 변경하면 시스템의 모든 사용자에게 영향을 준다.

/etc/security/pam_policy 디렉토리에도 PAM 구성 파일이 보관된다. 이러한 파일은 여러 서비스를 취급하며 사용자별로 지정할 수 있도록 설계되었다. 이 디렉토리의 파일은 수정하지 않아야 한다.

1) /etc/pam.d 디렉토리 - 와일드카드 파일 other를 비롯한 서비스별 PAM 구성 파일이 있다. 응용 프로그램에 대한 서비스를 추가하려면 service-name 파일을 추가한다. 이 파일은 응용 프로그램에 사용되는 서비스 이름다. 적합한 경우 응용 프로그램에서 other 파일의 PAM 스택을 사용할 수 있다.

/etc/pam.d 디렉토리의 서비스 파일은 대부분의 PAM 구현에 있는 기본 구성을 제공한다. 이러한 파일은 pkg(5) 매뉴얼 페이지에 설명된 IPS 방식을 사용하여 자체 어셈블된다. 이 기본 동작은 다른 플랫폼간 PAM 응용 프로그램과의 상호 운용성을 간소화한다. 자세한 내용은 pam.conf(4) 매뉴얼 페이지를 참조하라.

2) /etc/pam.conf 파일 - 레거시 PAM 구성 및 정책 파일다. 이 파일은 비어 있는 상태로 제공된다. 선호되는 PAM 구성 방식에서는 /etc/pam.d 디렉토리의 파일을 사용한다. 자세한 내용은 pam.conf(4) 매뉴얼 페이지를 참조하라.

3) /etc/security/pam_policy 디렉토리 - 여러 서비스에 대한 정책이 포함된 PAM 정책 파일이 있다. 필요에 따라 이러한 파일을 개별 사용자, 개별 사용자 그룹 또는 모든 사용자에게 지정할 수 있다. 이렇게 지정할 경우 pam.conf 또는 /etc/pam.d 디렉토리에 있는 PAM 구성 파일을 겹쳐쓰게 된다. 이러한 파일은 수정하지 마라. 사용자별 파일을 추가하려면 사이트별 PAM 구성 파일을 만드는 방법을 참조하라. 사용자별 파일에 대한 자세한 내용은 pam_user_policy(5) 매뉴얼 페이지를 참조하라.

보안 관리자는 모든 PAM 구성 파일을 관리한다. 항목 순서가 잘못되면, 즉 PAM 스택이 올바르게 없으면 예측하지 못한 역효과가 발생할 수 있다. 예를 들어, 잘못 구성된 파일은 사용자를 잠그게 되므로 복구하려면 단일 사용자 모드가 필요할 수 있다. 자세한 내용은 PAM 스택 및 PAM 구성 오류 문제를 해결하는 방법을 참조하라.

1.7 PAM 구성 검색 순서

PAM 프레임워크에 대한 응용 프로그램 호출은 다음과 같은 순서로 구성된 PAM 서비스를 검색한다.

- 1) /etc/pam.conf에서 서비스 이름을 조회한다.
- 2) /etc/pam.d/service-name 파일에 있는 특정 서비스가 사용된다.
- 3) /etc/pam.conf 파일에서 서비스 이름 other를 확인한다.
- 4) /etc/pam.d/other 파일이 사용된다.

이 순서에 따라 기존 /etc/pam.conf 파일이 /etc/pam.d에 있는 서비스별 PAM 구성 파일과 함께 작동할 수 있다.

1.8 PAM 구성 파일 구문

pam.conf 파일 및 PAM 사용자별 파일은 pam.d 디렉토리에 있는 서비스별 파일과는 다른 구문을 사용한다.

- 1) /etc/pam.conf 파일 및 /etc/security/pam_policy 파일의 항목은 다음 두 형식 중 하나다.

```
[service-name] [module-type] [control-flag] [module-path] [module-options]
[service-name] [module-type] [include] [path-to-included-PAM-configuration]
```

- 2) /etc/pam.d 디렉토리의 service-name 파일에 있는 항목에는 서비스 이름이 생략된다. 파일 이름이 서비스 이름을 제공한다.

```
[module-type] [control-flag] [module-path] [module-options]
[module-type] [include] [path-to-included-PAM-configuration]
```

PAM 구성 파일 구문의 항목은 다음과 같다.

1) service_name

서비스의 이름(대소문자 구분 안함)다(예: login 또는 ssh). 응용 프로그램에서는 응용 프로그램이 제공하

는 서비스에 대해 서로 다른 서비스 이름을 사용할 수 있다. 예를 들어 sshd 데몬이 제공하는 다양한 서비스의 서비스 이름을 확인하려면 sshd(1M) 매뉴얼 페이지에서 PAM을 검색하라.

미리 정의된 서비스 이름 "other"는 특정 서비스 구성이 제공되지 않은 경우 사용되는 기본 서비스 이름이다.

2) module-type

PAM이 어떤 타입의 인증이 사용될 것인지 서비스의 유형(즉, auth, account, session 또는 password)을 나타낸다.

모듈 타입	설명	비고
auth	사용자 인증에 사용하며, 올바른 비밀번호인지 확인하는 절차를 가진다. 응용 프로그램이 사용자에게 비밀번호를 입력하도록 안내하고, 사용자와 해당 사용자의 그룹에 대한 권한을 인증한다.	
account	사용자의 접근 허가 여부를 확인하며 계정 만료, 특정 시간대에 접근이 허용되었는지 여부를 확인한다. 인증하는 기능이 아니며 현재 시간, 사용자의 위치와 같은 다른 요소들의 접근 권한을 결정하는데, 예로 root 사용자의 로그인은 콘솔로만 한다. 이런 식으로 결정하는 것이다.	
password	비밀번호를 설정하고 확인한다. 비밀번호를 기준에 맞게 변경하도록 하는 모듈을 지정한다.	
session	사용자가 인증 받기 전후에 필요한 홈 디렉토리 마운트, 메일박스 생성 등의 유저 섹션을 구분하기 위해 부가적인 작업을 수행한다. 혹시라도 사용자의 로그인 전후에 수행해야 할 작업이 있다는 내용을 지정한다.	

3) control-flag

PAM에서 사용되는 모듈들이 결과에 따라 어떠한 동작을 취해야 하는지를 지시하는 서비스에 대한 성공 또는 실패 값을 결정하는 모듈의 역할을 나타낸다.

flag	설명	비고
required	인증이 거부되기 전에 PAM이 이 서비스에 등록된 모든 모듈들을 요구함에도 불구하고 실패할 경우 인증을 거부하도록 한다. 해당 모듈은 개별 사용자에게 인증을 반드시 진행해야 한다. 인증이 안될 경우 실패가 반드시 반환되어야 한다.	
requisite	이 모듈을 이용하는 인증이 실패할 경우, 즉시 인증을 거부하도록 한다. required와 비슷하지만 이 플래그가 인증을 실패할 경우 설정 파일에서 이 값 다음으로 나오는 모듈들을 호출되지 않는다. 실패한 결과는 즉시 응용 프로그램으로 전달된다.	
sufficient	이전에 요청되었던 모듈이 실패하더라도 이 모듈에 의해서 인증이 성공할 경우 PAM은 인증을 승인한다. 모듈이 성공 값을 반환하고 설정 파일에서 required와 sufficient 제어 플래그가 필요하지 않다면 PAM은 해당 응용 프로그램에서 성공을 반환한다.	
optional	이 모듈이 성공 또는 실패하는지는 그 모듈이 서비스에 대한 형식에 유일한 모듈일 경우에 해당한다. (성공 여부 상관 X) 이 제어 플래그는 모듈의 결과를 무시한다. PAM으로 다른 모듈을 지속적으로 확인하도록 한다. 확인 작업이 실패하게 되더라도 계속 진행하게 된다. 특정 모듈이 실패하더라도 사용자가 로그인하는 것을 허용하고자 할 때 이 플래그를 사용하면 된다.	
include	이 플래그는 인자(argument)에 지정된 또 다른 설정 파일의 내용이나 지침을 포함시키기 위해 사용된다. 즉, 서로 다른 PAM 설정 파일의 내용을 연결하고 구성하는 방식으로 사용된다.	

substack	이 플래그는 다른 PAM 관련 모듈을 불러오는 것은 include와 동일 하지만, substack 은 substack의 동작 결과에 따라 나머지 모듈을 처리하지 않는다
----------	---

required/requisite 차이는 required와 requisite 인터페이스 모두 반드시 "성공" 되어야만 PAM을 이용한 인증이 완료되나 보안 및 가용성 측면에서는 분명한 차이가 존재한다. required의 경우 실패를 해도 실패한 지점이나 결과값에 대한 리턴을 하지 않는데, requisite의 경우 실패한 지점과 원인을 리턴하기에 보안 관점에서는 공격자에게 인증이 거부된 원인을 제공하게 되며, 가용성 측면에서는 실패한 원인을 리턴하여 원인 분석을 용이하게 해준다.

4) module-path

PAM에게 어떤 모듈을 사용할 것인지 그리고 그것을 어디서 찾을지의 모듈 유형을 구현하는 모듈의 경로다. 경로 이름이 절대 경로가 아닌 경우 경로 이름은 /usr/lib/security/\$ISA/ 경로에 대한 상대 경로로 간주된다. \$ISA 매크로 또는 토큰은 PAM 프레임워크가 모듈 경로의 아키텍처 특정 디렉토리를 검색하도록 지시한다.

5) module-option

서비스 모듈로 전달될 수 있는 nowarn 및 debug와 같은 옵션이다. 모듈의 매뉴얼 페이지에서는 해당 모듈에 대한 옵션을 설명한다. 각각의 모듈들은 각각의 인수를 가지고 있다.

arguments	설명	비고
debug	시스템 로그에 디버깅 정보를 남기다.	
no_warn	응용 프로그램에 경고 메시지를 제공하지 않는다.	
use_first_pass	비밀번호를 두 번 확인하지 않는다. 대신 auth 모듈에서 입력한 비밀번호를 사용자 인증 과정 시에도 재사용 해야 한다. (이 옵션은 auth 및 password 모듈에 해당하는 옵션임)	
try_first_pass	이 옵션은 use_first_pass 옵션과 비슷한데, 사용자는 두 번 비밀번호를 입력할 필요가 없기 때문이다. 하지만 기존의 비밀번호를 다시 입력하도록 되어 있다.	
use_mapped_pass	이 인자는 이전 모듈에서 입력된 텍스트 인증 토큰을 입력 받도록 하는데, 이 값으로 암호화 또는 암호화가 해제된 키 값을 생성한다. 그 이유는 모듈에 대한 인증 토큰 값을 안전하게 저장하거나 불러오기 위함이다.	
expose_account	이 값은 모듈로 하여금 계정 정보를 중요하다고 판단하지 않게 한다. 시스템 관리자에 의해 임의로 설정한 것이라 여겨진다.	
nullok	이 인자는 호출된 PAM 모듈이 null 값의 비밀번호를 입력하는 것을 허용한다.	

6) path-to-included-PAM-configuration

PAM 구성 파일 또는 /usr/lib/security 디렉토리를 기준으로 한 파일 이름을 지정한다.

1.9 PAM 스택

응용 프로그램이 다음 함수 중 하나를 호출하면 PAM 프레임워크는 PAM 구성 파일을 읽고 해당 응용 프로그램의 PAM 서비스 이름을 구현하는 모듈을 확인한다.

```
pam_authenticate(3PAM)
pam_acct_mgmt(3PAM)
pam_setcred(3PAM)
```

pam_open_session(3PAM)
 pam_close_session(3PAM)
 pam_chauthtok(3PAM)

구성 파일에 모듈이 하나만 포함된 경우 해당 모듈의 결과가 작업의 결과를 결정한다. 예를 들어, passwd 응용 프로그램에 대한 기본 인증 작업에는 /etc/pam.d/passwd 파일에 하나의 모듈(pam_passwd_auth.so.1)이 포함된다.

auth required	pam_passwd_auth.so.1
---------------	----------------------

이러한 항목은 login 서비스 이름에 대한 auth 스택을 만든다. 이 스택의 결과를 결정하려면 개별 모듈의 결과 코드에 통합 프로세스가 필요하다.

통합 프로세스에서 모듈은 파일에 나오는 순서대로 실행된다. 각 성공 또는 실패 코드는 모듈의 제어 플래그에 따라 전체 결과에 통합된다. 제어 플래그는 스택의 조기 종료를 유발할 수 있다. 예를 들어, requisite 또는 definitive 모듈이 실패하면 스택이 종료된다. 이전 실패한 없는 경우 sufficient, definitive 또는 binding 모듈이 성공하면 스택이 종료된다. 스택이 처리된 후 개별 결과는 하나의 전체 결과로 합쳐서 응용 프로그램에 전달된다.

제어 플래그는 PAM 모듈이 성공 또는 실패를 결정하는 데 수행하는 역할을 나타낸다. 제어 플래그 및 효과는 다음과 같다.

1) Binding - 기록된 이전 실패가 없는 경우 binding 모듈의 요구 사항 충족에 성공하면 응용 프로그램에 즉시 성공을 반환한다. 이러한 조건이 충족되지 않는다면 모듈은 더 이상 실행되지 않는다.

실패의 경우 required 실패가 기록되고 모듈 처리가 계속된다.

2) Definitive - 기록된 이전 실패가 없는 경우 definitive 모듈의 요구 사항 충족에 성공하면 응용 프로그램에 즉시 성공을 반환한다.

이전 실패가 기록된 경우 모듈이 추가로 실행되지 않고 해당 실패가 응용 프로그램에 즉시 반환된다. 실패의 경우 모듈의 추가 실행 없이 즉시 오류가 반환된다.

3) Include - PAM 스택의 이 시점에서 사용될 별도의 PAM 구성 파일에서 행을 추가한다. 이 플래그는 성공이나 실패 동작을 제어하지 않는다. 새 파일이 읽히면 PAM include 스택이 증가된다. 새 파일에서 스택 확인이 완료되면 include 스택 값이 감소한다. 파일의 끝에 도달하고 PAM include 스택이 0이면 스택 처리가 종료된다. PAM include 스택에 대한 최대 수는 32이다.

4) Optional - optional 모듈의 요구 사항을 충족하는 성공은 서비스를 사용하는 데 필요하지 않는다.

실패의 경우 optional 실패가 기록된다.

5) Required - required 모듈의 요구 사항을 충족하는 성공은 스택이 성공하는 데 필요하다. 스택에 대한 최종 성공은 실패를 보고한 binding 또는 required 모듈이 없을 경우에만 반환된다.

실패의 경우 이 서비스에 대한 나머지 모듈이 실행된 후 오류가 반환된다.

6) Requisite - requisite 모듈의 요구 사항을 충족하는 성공은 스택이 성공하는 데 필요하다. 스택이 응용 프로그램에 성공을 반환할 수 있으려면 스택의 모든 requisite 모듈이 성공을 반환해야 한다.

실패의 경우 모듈의 추가 실행 없이 즉시 오류가 반환된다.

7) Sufficient - 기록된 이전 required 실패가 없는 경우 sufficient 모듈의 성공은 모듈의 추가 실행 없

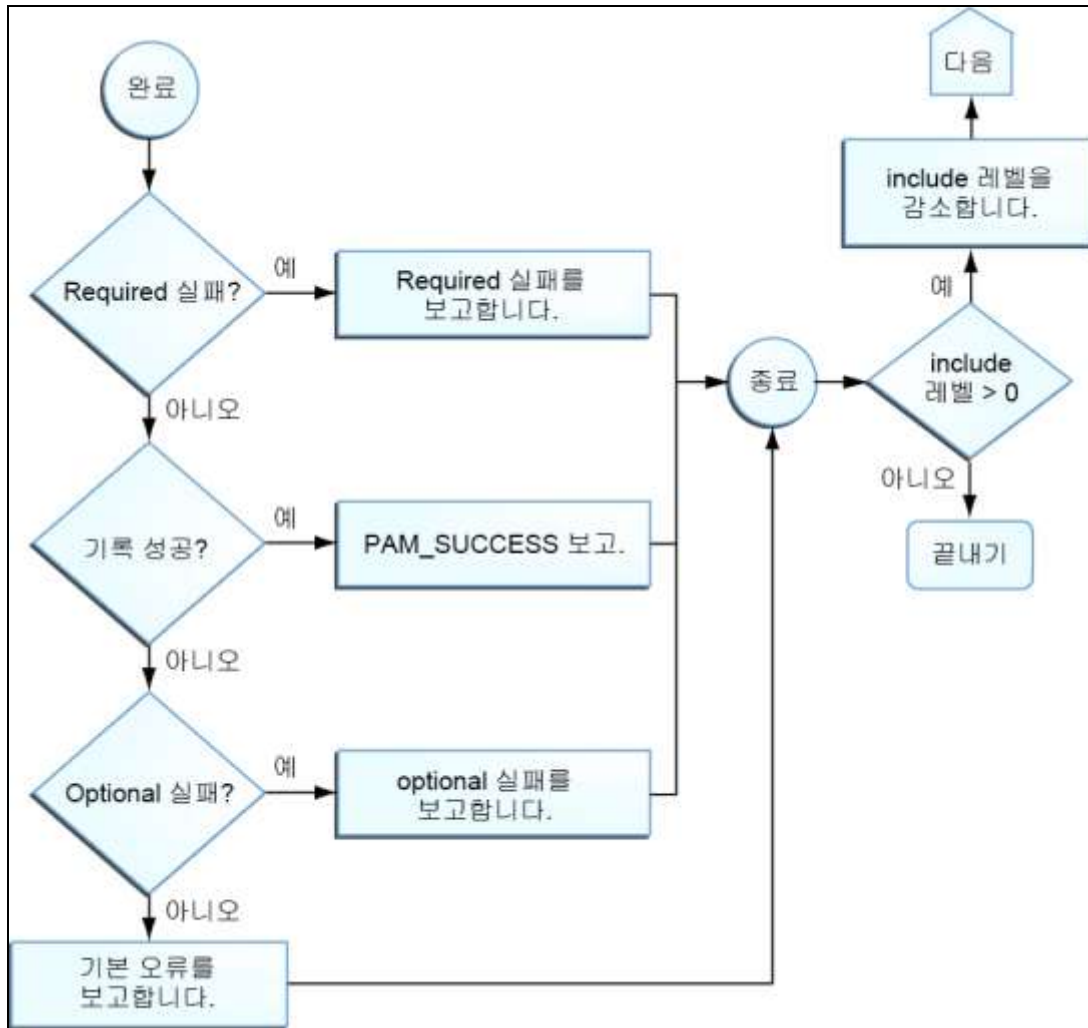
이 응용 프로그램에 즉시 성공을 반환한다.

실패의 경우 optional 실패가 기록된다.

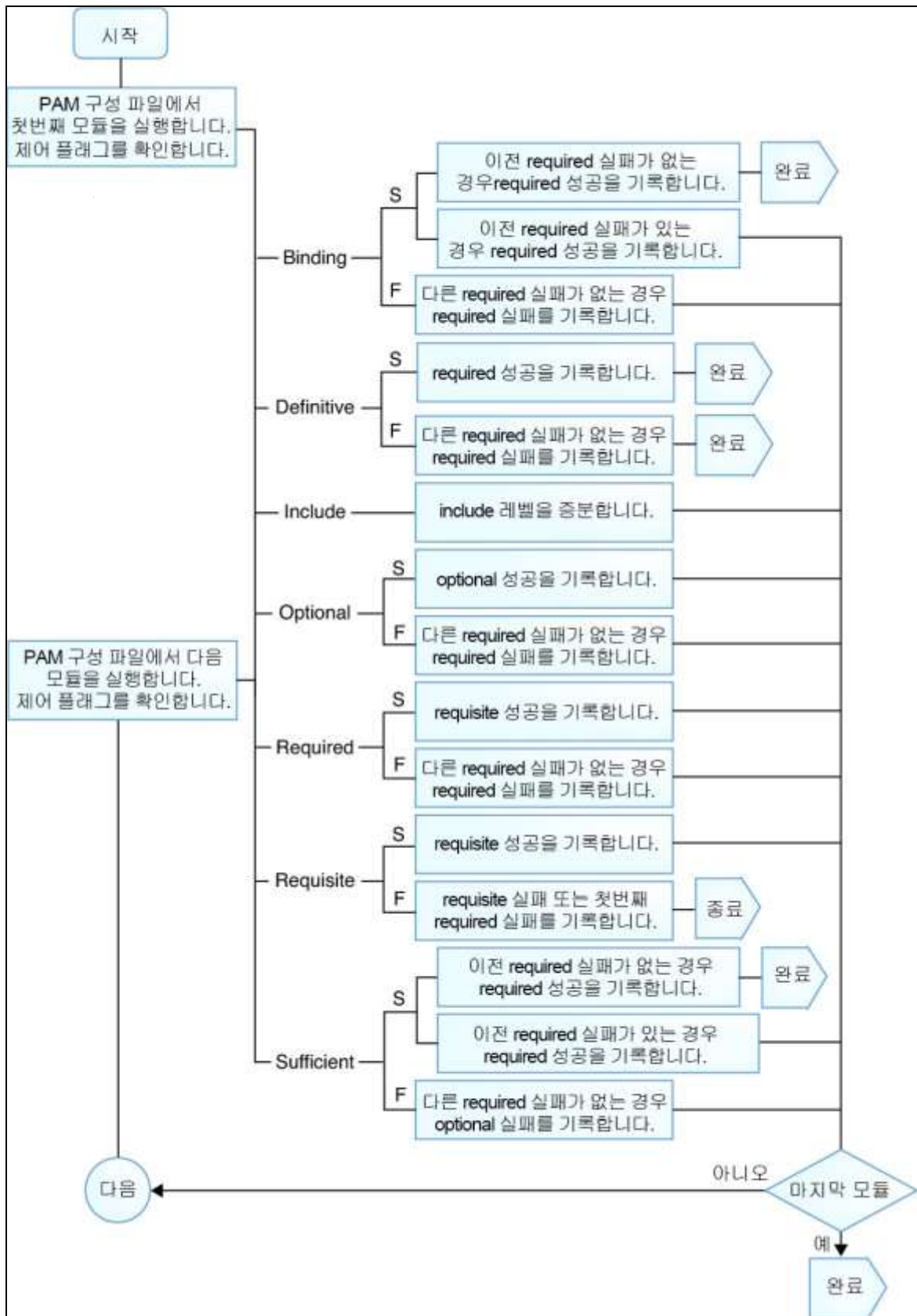
다음 두 연결된 다이어그램은 통합 프로세스에서 결과가 어떻게 결정되는지 보여 준다.

첫번째 다이어그램은 제어 플래그의 각 유형에 대해 성공 또는 실패가 어떻게 기록되는지 보여 준다. 두번째 다이어그램은 결과를 보여 준다.

두번째 다이어그램은 통합된 값이 어떻게 결정되는지 보여준다. Optional 실패 및 required 실패는 실패를 반환하고 성공은 성공을 반환한다. 이러한 반환 코드를 처리하는 방법은 응용 프로그램이 결정한다.



플로우 다이어그램은 제어 플래그가 PAM 스택에 어떤 영향을 미치는지 보여준다.



플로우 다이어그램은 통합된 값이 PAM 스택에서 어떻게 결정되는지 보여준다.

1.10 PAM 스택 사용예시

다음 예에서는 샘플 /etc/pam.d/other 파일의 인증 관리를 위한 기본 정의를 보여 준다. 이러한 정의는 서비스별 인증 정의가 구성되지 않은 경우에 인증에 사용된다.

```
##
# Default definitions for Authentication management
# Used when service name is not explicitly mentioned for authentication
#
auth definitive      pam_user_policy.so.1
auth requisite       pam_authtok_get.so.1
auth required        pam_dhkeys.so.1
auth required        pam_unix_auth.so.1
auth required        pam_unix_cred.so.1
```

먼저 pam_user_policy.so 모듈을 사용하여 사용자의 PAM 정책을 확인한다. definitive 제어 플래그는 구성된 PAM 스택에 대한 평가가 성공할 경우 이 시점에서 검사된 다른 모듈이 없기 때문에 응용 프로그램에 성공을 반환하도록 결정한다. 구성된 PAM 스택 평가가 실패할 경우에는 응용 프로그램에 실패 코드가 반환되고 추가 검사가 수행되지 않는다. 사용자별 PAM 정책이 이 사용자에게 지정되지 않았으면 다음 모듈이 실행된다.

사용자별 PAM 정책이 이 사용자에게 지정되지 않았으면 pam_authtok_get 모듈이 실행된다. 이 모듈에 대한 제어 플래그는 requisite로 설정된다. pam_authtok_get을 실패할 경우 인증 프로세스가 종료되고 응용 프로그램에 실패가 반환된다.

pam_authtok_get이 실패하지 않으면 다음 3개의 모듈이 실행된다. 이러한 모듈은 required 제어 플래그로 구성되므로 개별 실패가 반환되는지 여부에 관계없이 통합 프로세스가 계속된다. pam_unix_cred가 실행된 후에는 남은 모듈이 없다. 이때 모든 모듈이 성공하면 응용 프로그램에 성공이 반환된다. pam_dhkeys, pam_unix_auth 또는 pam_unix_cred가 실패를 반환한 경우 응용 프로그램에 실패가 반환된다.

1.11 SSH(Secure Shell)

Secure Shell은 개방형 소스 SSH 제품인 OpenSSH 제품을 기반으로 한다. (<http://www.openssh.org>)

Secure Shell은 비보안 네트워크에서 클라이언트와 원격 호스트 간의 보안 연결을 가능하게 한다.

이 보안 연결의 주요 속성은 다음과 같다.

- 클라이언트와 원격 호스트 둘 다에 대한 강력한 인증
- 클라이언트와 원격 호스트 간의 통신에 대한 강력한 암호화 및 공개 키 암호화
- 클라이언트와 원격 호스트에서 명령을 실행하는데 사용할 보안 연결

Secure Shell은 telnet, remsh, rlogin, ftp, 및 rcp와 같은 자주 사용하는 함수와 명령의 보안 대체 항목을 제공한다.

1) Secure Shell의 주요 보안 기능

Secure Shell의 주요 보안 기능은 다음과 같다.

- ① 강력한 암호화

클라이언트와 원격 호스트 간의 모든 통신은 Blowfish, 3DES, AES 및 arcfour와 같은 특허 없는 암호화 알고리즘을 사용하여 암호화된다. 암호 등의 인증 정보는 네트워크에서 일반 텍스트로 전송되지 않는다. 또한 암호화는 강력한 공개 키 기반 암호화와 더불어 잠재적 보안 공격을 차단한다.

② 강력한 인증

Secure shell은 클라이언트와 서버 간의 강력한 인증 방법 집합을 지원한다. 인증은 양방향일 수 있다. 서버는 클라이언트를 인증하고 클라이언트도 서버를 인증한다. 이렇게 하면 다양한 보안 문제로부터 세션을 보호할 수 있다.

③ 포트 전달

클라이언트와 원격 호스트 간의 TCP/IP 연결 리디렉션(및 그 반대)을 포트 전달 또는 SSH 터널링이라고 한다. Secure Shell은 포트 전달을 지원한다. 예를 들어, 포트 전달을 사용하여 클라이언트와 서버 간의 ftp 트래픽(또는 전자 메일 클라이언트와 POP/IMAP 서버 간의 전자 메일 트래픽)을 리디렉션할 수 있다. 클라이언트가 직접 서버와 통신하는 대신 보안 채널을 통해 트래픽을 sshd 서버로 리디렉션할 수 있으며, 그런 다음 sshd 서버에서 트래픽을 실제 서버 시스템의 지정된 포트로 전달할 수 있다.

2) Secure Shell의 소프트웨어 구성 요소

Secure Shell 소프트웨어는 클라이언트 및 서버 구성 요소로 이루어져 있다.

구성요소	설명	위치	대응하는 비보안 구성요소
ssh	Secure Shell을 클라이언트는 telnet 및 remsh의 보안 대체 항목이며 보안 기능이 있는 remsh와 가장 유사함.	클라이언트	remsh, telnet, rlogin
slogin	ssh에 대한 심볼릭 링크임.	클라이언트	remsh, telnet, rlogin
scp	클라이언트 보안 복사 및 서버 보안 복사를 수행함,	클라이언트 및 서버	rcp
sftp	보안 ftp 클라이언트임	클라이언트	ftp
sshd	보안 Shell 데몬임.	서버	remshd, telnetd, rlogind
sftp-server	보안 ftp 데몬임.	서버	ftpd
ssh-rand-helper	sshd가 서버에서 /dev/random 또는 /dev/urandom을 찾을 수 없을 때는 사용되는 Random Number Generator. OS 커널에 상주하는 Random Number Generator인 rng가 포함되어 있다. rng가 구성 해제되어 있으면 sshd는 prngd를 사용함.	서버	해당 사항 없음
ssh-agent	클라이언트에서 서버로의 "자동" 키 기반 로그인 도구임.	클라이언트 및 서버	Rhosts 파일 메커니즘
ssh-add	클라이언트의 키 쌍을 ssh-agent에 알리는 도구임.	클라이언트	해당 사항 없음
ssh-keygen	공개 키 인증을 위해 키 쌍을 생성하는 도구임.	클라이언트	해당 사항 없음
ssh-keyscan	Secure Shell 데몬(sshd)을 실행하는 호스트 집합에 대한 공개 키를 수집하는 클라이언트 도구임.	클라이언트	해당 사항 없음
ssh-keysign	호스트 기반 인증 중에 필요한 디지털 서명을 생성하는 도구임. ssh()에서 로컬 호스트 키 호스트 기반 인증에 액세스하는데 사용됨.	클라이언트	해당 사항 없음

3) Secure Shell 실행

위 표에서 나열된 Secure Shell 클라이언트를 실행하기 전에 먼저 Secure Shell 서버 데몬 sshd를 시작한다.

sshd 데몬은 서버 시스템의 /etc/ssh 디렉토리에 있는 sshd_config 파일에서 초기 값을 가져온다.

sshd_config에 있는 가장 중요한 구성 지시어 중 하나는 sshd 데몬에서 지원하는 인증 방법 집합이다.

- ssh 클라이언트 실행

ssh 클라이언트 응용 프로그램은 sshd 서버와의 소켓 연결을 설정한다.

sshd 서버는 자식 sshd 프로세스를 시작한다.

이 자식은 연결 소켓을 상속 받고 선택된 인증 방법을 기반으로 클라이언트를 인증한다.

인증에 성공한 경우에만 성공적으로 보안 클라이언트 세션이 설정된다.

세션이 만들어진 후 모든 후속 통신은 클라이언트와 이 자식 sshd 프로세스 간에 직접 이루어진다.

이제 클라이언트는 서버에서 원격 명령을 실행할 수 있다.

ssh 클라이언트의 명령 요청이 있을 때마다 자식 sshd 프로세스에서 해당 명령을 실행할 Shell 프로세스를 시작한다.

간단히 말해서 실행 중인 ssh 클라이언트-서버 세션은 다음 프로세스로 구성된다.

① sshd 서버에 연결된 모든 클라이언트 시스템에는 현재 이 클라이언트 시스템에서 설정된 각 ssh 연결에 대한 하나의 ssh 클라이언트 프로세스가 있다.

② 서버 시스템에는 하나의 부모 sshd 프로세스와 서버에 연결된 동시 ssh 클라이언트 개수 만큼의 자식 sshd 프로세스가 있다. 서버에 관한 분리가 활성화되어 있으면 서버에서 실행되는 자식 sshd 프로세스의 수가 두 배로 증가한다.

③ ssh 클라이언트에서 명령 실행 요청이 있을 때마다 서버 시스템의 해당 자식 sshd 프로세스는 Shell 프로세스를 시작하고 Unix 파이프를 사용하여 명령 요청을 이 Shell 프로세스로 전달한다. 이 Shell 프로세스는 Unix 파이프를 사용하여 명령 실행 결과를 자식 sshd 프로세스로 반환하고 명령 실행이 완료되면 종료된다.

- sftp 클라이언트 실행

sftp 클라이언트 응용 프로그램은 sftp 클라이언트 응용 프로그램이 ssh 클라이언트를 시작 하도록 하고 Unix 파이프를 사용하여 이 클라이언트와 통신한다. 그런 다음 ssh 클라이언트는 sshd 서버와의 소켓 연결을 설정한다.

서버 상호 작용의 나머지 부분은 ssh 클라이언트의 경우와 유사하다. 차이점은 원격 명령을 실행한 Shell 을 시작하는 대신 자식 sshd 프로세스가 sftp-server 프로세스를 시작한다는 것이다. 이 sftp 세션 중의 모든 후속 통신은 다음 프로세스 간에 발생한다.

① Unix 파이프를 사용하여 클라이언트 시스템에서 sftp 클라이언트와 ssh 클라이언트 간에

② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에

③ Unix 파이프를 사용하여 자식 sshd 프로세스와 sftp 서버 프로세스 간에

- scp 클라이언트 실행

scp 클라이언트의 경우는 sftp 클라이언트 실행과 거의 동일하다. 차이점은 sftp-server 프로세스를 시작하는 대신 자식 sshd 프로세스가 scp 프로세스를 시작한다는 것이다. scp 세션 중의 모든 후속 통신은 다음 프로세스간에 발생한다.

- ① 클라이언트 시스템에서 scp 클라이언트와 ssh 클라이언트 간에, Unix 파이프 사용
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Unix 파이프를 사용하여 자식 sshd 프로세스와 scp 서버 프로세스 간에

4) Secure Shell 권한 분리

Secure Shell은 권한 분리 기능을 통해 보다 향상된 수준의 보안을 제공한다. 부모 sshd 및 자식 sshd 프로세스는 권한이 부여된 사용자로 실행된다. 권한 분리를 활성화하면 사용자 연결당 하나의 추가 프로세스가 시작된다.

ssh 클라이언트가 권한 분리에 대해 구성된 sshd 서버에 연결하면 부모 sshd 프로세스가 권한이 부여된 자식 sshd 프로세스를 시작한다. 권한 분리를 활성화하면 자식 sshd 프로세스는 권한이 없는 자식 sshd 프로세스를 추가로 시작한다. 권한이 없는 자식 sshd 프로세스는 연결 소켓을 상속 받는다. 클라이언트와 서버 간의 모든 후속 통신은 권한이 없는 이 자식 sshd 프로세스를 사용하여 이루어진다.

클라이언트의 원격 명령 실행 요청은 대부분 비권한이며 권한이 없는 이 자식 sshd 프로세스에서 시작된 Shell에 의해 처리된다. 권한이 없는 자식 sshd 프로세스에서 권한이 부여된 기능을 실행해야 하는 경우 Unix 파이프를 사용하여 권한이 부여된 부모 sshd 프로세스와 통신한다.

권한 분리는 침입자에 의한 잠재적 손상을 제한하는데 도움이 된다. 예를 들어, Shell 명령을 실행하는 동안 버퍼 오버플로 공격이 발생할 경우 권한이 없는 프로세스 내에서 제어되므로 잠재적 보안 위험이 제한된다.

권한 분리는 Secure Shell의 기본 구성이다. sshd_config 파일에서 "UsePrivilegeSeparation NO"를 설정하여 권한 분리를 해제할 수 있다. 잠재적 권한 위험이 있으므로 권한 분리를 해제할 경우 신중하게 고려해야 한다.

5) Secure shell 인증

Secure Shell은 다음 인증 방법을 지원한다.

- GSS-API (Kerberos 기반 클라이언트 인증)
- 공개 키 인증
- 호스트 기반 인증
- 암호 인증

클라이언트는 원격 sshd 데몬과 연결될 때 원하는 인증 방법(앞에 나열된 방법 중 하나)을 선택하고 연결 요청의 일부로 적절한 자격 증명을 제공하거나 서버에서 보낸 프롬프트에 응답한다. 모든 인증 방법이 이런 방식으로 작동한다.

서버가 성공적으로 연결을 설정하려면 클라이언트로부터 적절한 키, 암호구, 암호 또는 자격 증명을 받아야 한다.

sshd 인스턴스에서 보안 요구 사항을 기반으로 지원되는 인증 방법 중 일부만 지원하도록 선택할 수 있다.

Secure Shell은 앞에 나열된 인증 방법을 지원하지만 시스템 관리자가 환경의 특정 보안 요구 사항을 기반으로 sshd 인스턴스에서 제공되는 인증 방법을 제한할 수 있다. 예를 들어, Secure Shell 환경에서 모든 클라이언트가 공개 키 또는 Kerberos 방법을 사용하여 인증해야 한다고 지정할 수 있으며, 이로 인해 나머지 방법은 비활성화될 수 있다. 지원되는 인증 방법 활성화 및 비활성화는 sshd_config 파일에 지정된 구성 지시어를 통해 이루어진다.

ssh 클라이언트 연결 요청이 있으면 서버는 먼저 지원되는 인증 방법 목록으로 응답한다. 이 목록은 sshd 서버에서 지원하는 인증 방법과 이러한 방법이 시도되는 시퀀스를 나타낸다. 클라이언트는 이러한 인증 방법을 하나 이상 생각할 수 있다. 클라이언트에서 방법이 시도되는 시퀀스를 변경할 수도 있다. 이렇게 하려면 클라이언트 구성 파일 /etc/ssh/ssh_config의 구성 지시어를 사용한다.

Secure Shell에서 지원하는 인증 방법은 다음과 같이 요약되어 있다.

① GSS-API (Kerberos 기반 클라이언트 인증)

Kerberos 기반 클라이언트 인증인 GSS-API (Generic Security Service application Programming Interface)를 사용하는 경우 클라이언트가 미리 Kerberos 자격 증명을 받아야 하며, 해당 클라이언트 디렉토리에 Kerberos 구성 파일이 있어야 한다.

클라이언트는 sshd 데몬과 연결될 때 연결 시 자격 증명을 제공한다.

서버는 이러한 자격 증명을 이 특정 사용자의 자격 증명 복사본과 일치 시킨다.

또한 선택적으로 클라이언트 호스트 환경의 타당성을 설정할 수 있다.

② 공개 키 인증

공개 키 인증을 사용하려면 Secure Shell 환경에 다음 설정이 있어야 한다..

클라이언트와 서버 둘 다에 키 쌍이 있어야 한다. 모든 ssh 클라이언트와 모든 sshd 서버가 ssh-keygen 유틸리티를 사용하여 자체적으로 키 쌍을 생성해야 한다.

클라이언트는 통신해야 하는 모든 sshd 서버에 해당 공개 키를 알려야 한다. 이렇게 하려면 각 클라이언트의 공개 키를 모든 관련 서버의 미리 지정된 디렉토리에 복사한다.

클라이언트는 통신해야 하는 모든 서버의 공개 키를 구해야 한다. 클라이언트는 ssh-keyscan 유틸리티를 사용하여 공개 키를 받는다.

이 설정이 완료 되면 sshd 서버에 연결하는 ssh 클라이언트가 공개 키와 개인 키를 사용하여 인증된다.

Secure Shell은 공개 키 인증을 단순화하는 추가 기능을 제공한다. 일부 환경에서는 항상 암호 프롬프트에 응답할 필요가 없도록 설정할 수 있다. 둘 다 클라이언트 시스템에서 실행되는 ssh-agent 및 ssh-add 프로세스를 조합해서 사용하면 암호에 응답하지 않아도 된다. 클라이언트는 ssh-add 유틸리티를 통해 ssh-agent 프로세스에 모든 키 정보를 등록한다. 그런 다음 클라이언트와 서버 간의 공개 키 인증은 sshd 데몬이 클라이언트와 상호 작용할 필요 없이 ssh-agent에 의해 수행된다.

③ 호스트 기반 및 공개 키 인증

호스트 기반 및 공개 키 인증은 보다 안전한 공개 키 인증 방법의 확장이다.

클라이언트와 서버 둘 다의 키 쌍이 있는 것 외에도 이 방법을 사용하면 클라이언트 환경에서 통신할 서버

를 제한할 수 있다.

클라이언트의 홈 디렉토리에 .rhosts 파일을 만들어 이 제한을 구현한다.

④ 암호 인증

암호 인증 방법은 단일 사용자-ID 및 암호 기반 로그인을 사용한다. 이 로그인은 /etc/passwd에 지정된 사용자 로그인을 기반으로 하거나 PAM 기반일 수 있다.

Secure Shell은 서버 시스템에서 사용할 수 있는 PAM 모듈과 완전히 통합된다. 이 목적을 위해 /etc/ssh/sshd_config 파일에 UsePAM 구성 지시어가 있다. YES로 설정하면 클라이언트에서 암호 인증 요청이 있을 때마다 sshd는 PAM 구성 파일(/etc/pam.conf)을 확인한다. 그런 다음 구성된 PAM 모듈을 통해 암호 인증이 성공할 때까지 순서대로 수행한다.

PAM 인증을 무시하려면 UsePAM 지시어를 NO로 설정한다. 그러면 클라이언트에서 암호 인증 요청이 있을 때 sshd가 서버의 PAM 구성 설정을 무시한다. 대신 sshd는 gwtppnam() 라이브러리 호출을 직접 호출하여 사용자 암호 정보를 가져온다.

Secure Shell은 PAM_UNIX, PAM_LDAP 및 PAM_KERBEROS를 사용하여 테스트 되었다. 또한 PAM_DCE 및 PAM_NTLM과 같은 다른 PAM 모듈에서 작동한다.

6) 통신 프로토콜

Secure Shell 사용자는 SSH-1 또는 SSH-2 프로토콜을 사용하여 원격 sshd 데몬과 연결할 수 있다. SSH-2가 더 안전하므로 SSH-1 대신 권장한다.

7) Secure Shell에서 사용하는 기능의 일부 목록

Secure Shell은 실제로 Shell이 아니라 호스트에서 안전하게 원격 shell 세션을 실행하기 위해 클라이언트와 원격 호스트 간에 보안 연결을 만드는 매커니즘이다. 보안 연결을 설정하기 위해 Secure Shell에서 인증과 세션 생성을 대부분 수행한다. Secure Shell에서 사용하는 기능의 일부 목록은 다음과 같다.

① login 시도 기록

telnet 또는 remsh와 마찬가지로 Secure Shell은 성공한 세션과 실패한 세션을 각각 /var/adm/wtmp 및 /var/adm/btmp 파일에 기록한다.

② PAM 모듈

Secure Shell은 클라이언트 세션에 대해 PAM 인증을 사용할 수 있다. PAM 인증을 선택하면 Secure Shell은 /etc/pam.conf 파일을 사용하고 인증을 위해 해당 PAM 모듈을 호출한다.

③ /etc/default/security 파일 사용

로그인 동작 암호 및 기타 보안 구성을 정의하는 속성이 들어 있는 시스템 범위 구성 파일이다. 몇 가지 제한은 있지만 Secure Shell에서 이러한 속성을 사용할 수 있다.

④ Shadow passwd

Secure Shell은 Shadow passwd 기능과 통합된다.

⑤ 컨트롤 시스템 로그(syslog)

Secure Shell은 syslog를 사용하여 중요한 메시지를 남긴다.

⑥ 감사 로깅

Secure shell은 해당 코드로 감사 로깅(트러스트된 모드)을 구현했다.

8) 비밀번호 없이 SSH에 접속하는 방법

SSH 접속 시 비밀번호 없이 접속하는 방법은 다음과 같다.

① ssh key 생성 (없는 경우만)

```
$ ssh-keygen -t rsa
```

② 서버로 로컬에서 만든 공개키 복사

```
$ scp ~/.ssh/id_rsa.pub [id]@[address]:id_rsa.pub
```

③ 서버 접속 (일반접속)

```
$ ssh [id]@[address]
```

④ 로그인 후 개인폴더에 .ssh 폴더 생성 (있으면 pass)

```
$ mkdir .ssh
```

⑤ .ssh폴더 권한 변경

```
$ chmod 700 .ssh
```

⑥ 공개키 인증키 목록에 추가

```
$ cat id_rsa.pub >> .ssh/authorized_keys
```

⑦ 필요없는 공개키 제거

```
$ rm -f id_rsa.pub
```

⑧ 인증키 목록 권한 수정

```
$ chmod 644 .ssh/authorized_keys
```

⑨ 서버 접속

```
$ ssh -i ~/.ssh/id_rsa [id]@[address]
```

1.12 PAM 프로그래밍

아래 소스 코드는 PAM 모듈의 기본 형태로 "#if 0"으로 막혀 있는 부분에 통제를 할 수 있는 코드를 넣어 두면 원하는 동작(차단 또는 허용, 로그 기록)을 쉽게 처리할 수 있다.

```
/* pam_test.c */

#include <stdio.h>
#include <stdlib.h>
#include <security/pam_appl.h>
```

```

#include <security/pam_modules.h>

#ifdef PAM_MODULE_ENTRY
PAM_MODULE_ENTRY("pam_test");
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN extern
#endif

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

/* expected hook, this is where custom stuff happens */
PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    #if 0
        // 아래 조건을 만족하지 못할 경우 로그인 실패로 처리함
        if (check_auth_pam(pamh) < 0)
            return PAM_PERM_DENIED;
    #endif
    return PAM_SUCCESS;
}

```

위에 있는 pam_sm...으로 된 함수들이 사용자가 임의로 추가한 함수가 아니라 PAM에서 기본 함수들로서 해당 함수 부분에 원하는 동작 코드를 넣도록 되어 있다.

PAM 모듈 자체가 기존에 동작하는 telnet이나, ssh, login 등의 바이너리에 동적 라이브러리 형태로 붙기 때문에 위 코드에 삽입한 코드들은 로그인과 동시에 해당 서버로의 접속자의 모든 상황들을 감시하고 제어할 수 있다.

1.13 PAM 컴파일 방법

PAM 인증 모듈은 Solaris의 특성별 OS에서 제공하는 최적화 옵션에 따라 컴파일 하는 방법은 다음과 같다.

```

> sudo gcc -std=gnu99 -Wall -O2 -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT -g -c baro-auth.c
> sudo gcc -std=gnu99 -Wall -O2 -D_POSIX_PTHREAD_SEMANTICS -D_REENTRANT -g -o baro-auth baro-auth.o
> sudo gcc -std=gnu99 -D_POSIX_PTHREAD_SEMANTICS -fPIC -Wall -O2 -g -I/usr/local/ssl/include -c base64.c
> sudo gcc -std=gnu99 -D_POSIX_PTHREAD_SEMANTICS -fPIC -Wall -O2 -g -I/usr/local/ssl/include -c xxtea.c
> sudo gcc -std=gnu99 -D_POSIX_PTHREAD_SEMANTICS -fPIC -Wall -O2 -g -I/usr/local/ssl/include -c pam_baro_auth.c xxtea.h base64.h
> sudo gcc -G -rdynamic -shared -fPIC -Wall -O2 -o pam_baro_auth.so pam_baro_auth.o base64.o xxtea.o -lssl -lcrypto -lpam -ldl -lz -lc

```

PAM 프로그램 컴파일할 때 반드시 sudo 명령어를 사용하여 컴파일 해야 한다. 그렇지 않으면 다음과 같은 오류가 발생한다.

```

Oct 26 05:16:23 baropam sshd[1452]: [ID 437441 auth.alert] open_module[0:/etc/pam.d/other]: module /usr/baropam/pam_baro_auth.so writable by group
Oct 26 05:16:23 baropam sshd[1452]: [ID 625676 auth.error] load_modules[0:/etc/pam.d/other]: can not open module /usr/baropam/pam_baro_auth.so

```

sudo 명령어는 유닉스 및 유닉스 계열 운영 체제에서 다른 사용자의 보안권한과 관련된 프로그램을 구동할 수 있게 해주는 프로그램이다.

이것은 substitute user do (다른 사용자의 권한으로 명령을 이행하라, 는 뜻이다.) 의 줄임말이다.

기본적으로 Sudo는 사용자 비밀번호를 요구하지만 루트 비밀번호(root password)가 필요할 수 도 있고, 한 터미널에 한번만 입력하고 그 다음부터는 비밀번호가 필요 없다.

Sudo는 각 명령줄에 사용할 수 있으며 일부 상황에서는 관리자 권한을 위한 슈퍼유저 로그인(superuser login)을 완벽히 대신하며, 주로 우분투, 리눅스와 애플의 OS X 에서 볼 수 있다.

참고로 PAM 프로그램 컴파일할 때 필요한 기본적인 환경변수인 PATH(명령어 입력시 경로를 입력하지 않고 실행가능), LD_LIBRARY_PATH(외부 라이브러리를 링크할 때 참조할 경로)를 다음과 같이 설정한다.

```

export
PATH=$HOME/bin:/usr/bin:/usr/ccs/bin:/usr/local/bin:/usr/ucb:/usr/sfw/bin:$HOME/shell:/usr/sbin:.$PATH
export LD_LIBRARY_PATH=/usr/lib:/usr/ccs/lib:/lib:/usr/ucb:/usr/local/ssl/lib:/usr/sfw/lib:/usr/dt/lib:/usr/openwin/lib:.$LD_LIBRARY_PATH

```

PAM을 컴파일하기 위해서는 암호화 모듈에 "openssl" 라이브러리를 사용하므로 반드시 "openssl"을 다음과 같은 명령어로 설치해야 한다.

```
> pkg install openssl
```

만약, 인증 관련해서 시스템에 로그 메시지를 남기고 싶은 경우 syslog.conf에 설정해야 한다.

1.14 PAM 디버그 사용

PAM (Pluggable Authentication Modules) 라이브러리는 실행 중에 디버그 정보를 제공 할 수 있다.

시스템이 디버그 출력을 수집하도록 설정한 후에는 수집된 정보를 사용하여 PAM API 호출을 추적하고 현재 PAM 설정에서 오류 지점을 확인할 수 있다.

PAM 디버그 출력을 사용하려면 다음 단계를 완료 해야 한다.

step 1) /etc/syslog.conf 파일을 편집하여 원하는 우선 순위 수준에서 syslog 메시지를 로깅 할 파일을 식별한다.

예를 들어 PAM 디버그 수준 메시지를 /var/log/auth.log 파일에 보내려면 다음 텍스트를 syslog.conf 파일의 새 행으로 추가해야 한다.

```
*.debug /var/log/auth.log
```

step 2) touch 명령을 사용하여 1 단계에서 참조한 출력 파일 (/var/log/auth.log)이 존재하지 않으면 작성해야 한다.

step 3) 구성 변경 사항이 인식되도록 syslogd 데몬을 다시 시작하려면 다음 단계를 완료 해야 한다.

```
> /etc/init.d/syslog start | stop
```

PAM 응용 프로그램이 다시 시작되면 /etc/syslog.conf 구성 파일에 정의된 출력 파일에 디버그 메시지가 수집된다.

1.15 PAM 테스트 방법

PAM 프로그램 컴파일 또는 PAM 구성 환경 변경 후 sshd를 restart하지 않고 테스트 할 수 있는 방법은 다음과 같이 진행 할 수 있다.

1) sshd 데몬 기동하기

sshd 데몬(22000 포트)을 디버깅 모드로 띄우기 위하여 다음과 같은 명령어를 수행한다.

```
> ps -ef |grep sshd
  root 1339    1  0 05:12:11 ?        0:00 /usr/lib/ssh/sshd
> /usr/lib/ssh/sshd -D -p22000 -d
```

2) ssh로 접속하기

sshd 데몬(22000 포트)에 ssh로 접속하기 위하여 다음과 같은 명령어를 수행한다.

```
> ssh -v -p22000 root@1.234.83.169
```

2. FreeBSD

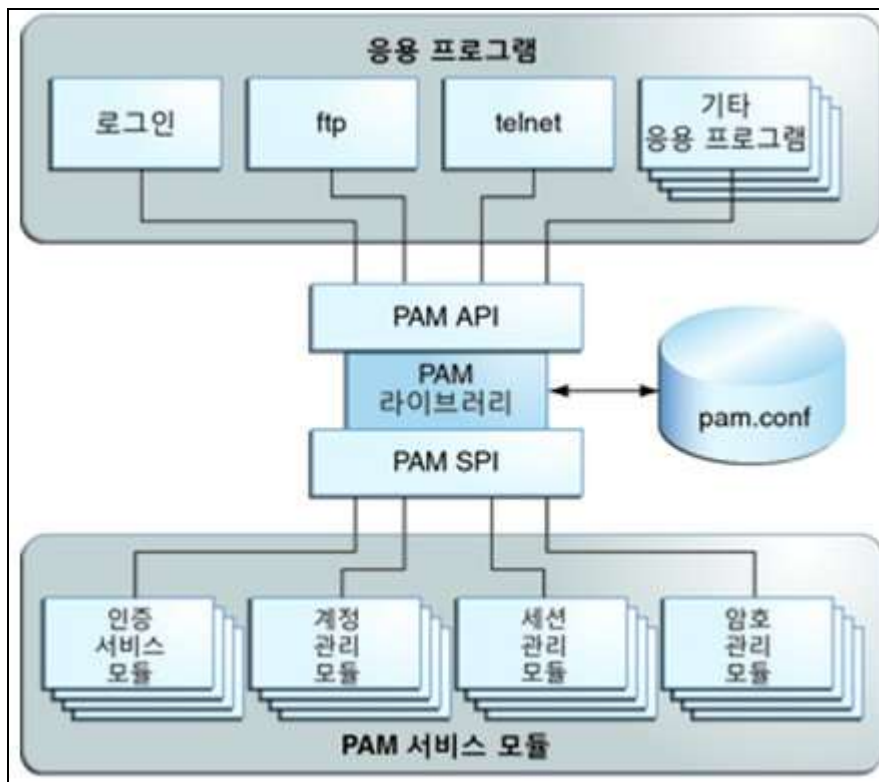
2.1 PAM 개요

PAM(Pluggable Authentication Module, 플러그 가능한 인증 모듈)은 Linux/Unix 시스템에서 서비스를 재 컴파일하지 않고, 다양한 인증 기술을 시스템 항목 서비스에 접목할 수 있도록 해주는 프레임워크로 중앙 집중적인 인증 매커니즘을 지원하는 것이다. 게다가 시스템의 기본적인 인증 기법을 제공하여 이것을 사용하면 응용 프로그램 개발자 뿐만 아니라 시스템 관리자들이 인증을 유연성 있게 관리할 수 있도록 도와 준다.

전통적으로 시스템 자원에 대한 접근을 관리하는 프로그램들은 내장된 매커니즘에 의해 사용자 인증 과정을 수행한다. 이러한 방식은 오랫동안 이루어졌지만 이러한 접근 방식은 확장성이 부족하고 매우 복잡하다. 그렇기 때문인지 이러한 인증 매커니즘을 끌어내기 위한 수많은 해킹 시도가 있었다.

Solaris의 방식을 따라서 Unix/Linux 사용자들은 그들만의 PAM을 구현하는 방식을 찾았다.

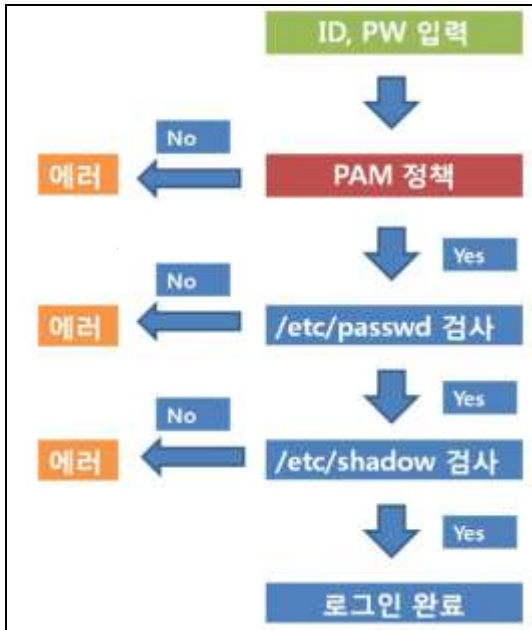
PAM의 아키텍처는 다음과 같다.



PAM의 기본 원리는 응용 프로그램이 password 파일을 읽어 오는 대신 PAM이 직접 인증을 수행 하도록 하는 것이다. PAM은 시스템 관리자가 원하는 인증 매커니즘이 무엇이든 상관하지 않는다.

여러 사이트에서 선택 받은 인증 매커니즘은 아직도 password 파일이다. 왜 그럴까? 우리가 원하는 것을 해주기 때문이다. 대부분의 사용자는 password 파일이 필요한 것이 무엇인지 이미 알고 있다. 그리고 원하는 작업을 수행하는데 있어 이미 그 기능이 검증되었기 때문일 것이다.

PAM의 인증 절차는 다음과 같다.



2.2 PAM 동작 원리

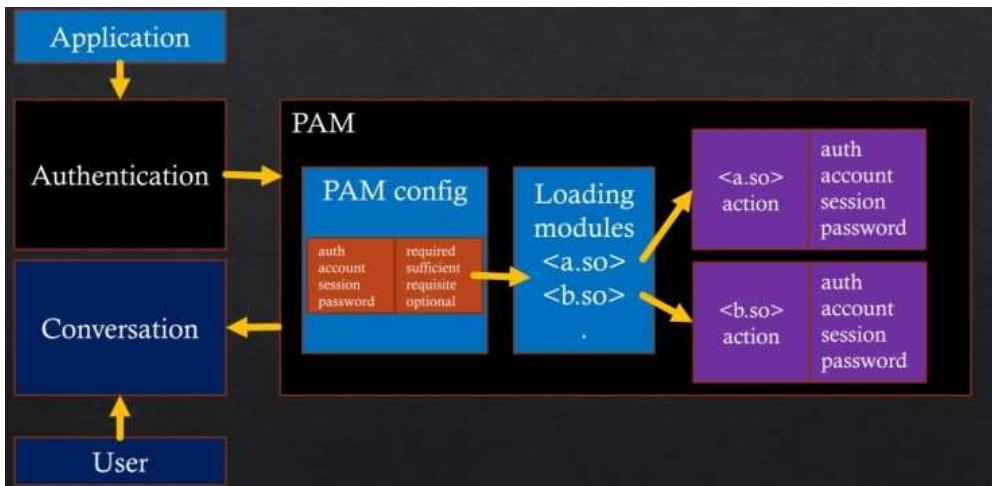
PAM은 Windows 환경의 DDL(Dynamic Link Library)과 같은 것이다. Library로 프로그램이 어떤 사용자에게 대한 인증을 수행하려면 PAM Library가 있는 함수를 호출한다. PAM은 해당 함수의 Library를 제공하여 응용 프로그램이 특정 사용자를 인증하도록 요청할 수 있다.

PAM을 통해 /etc/passwd 파일이나 /etc/shadow 파일을 확인하거나 또는 보다 복잡한 확인 작업을 수행하기도 하는데, 예로는 LDAP 서버에 접속하는 것이다.

확인 작업을 마치고, 인증 여부를 결정하게 되면 "인증됨/인증되지 않음"의 메시지를 자신을 호출한 응용 프로그램에 전송한다.

간단한 확인 작업이라고 했는데, 그 과정이 많아 보일 수가 있다. 여기에 나오는 각 모듈은 크기가 작고 작업 수행 시간이 매우 빠르다. 이것은 매우 놀랍기도 하지만 PAM을 사용하게 된다.

PAM의 동작은 프로그램의 실행 하자마자 실행되는 보안 설정 파일들의 프로그램들이다.



PAM의 동작 원리는 다음과 같다.

- 인증이 필요한 프로그램 동작 시 PAM 라이브러리 호출
- PAM설정 파일을 참조
- 참조한 파일의 내용을 바탕으로 모듈 동작
- 프로그램에 동작 결과를 반환하여 인증여부 결정.

2.3 PAM 사용 이점

PAM을 사용하면 사용자 인증을 위한 시스템 항목 서비스(예: ftp, login, telnet, rsh) 사용을 구성할 수 있다. PAM이 제공하는 몇 가지 이점은 다음과 같다.

- 유연한 구성 정책
- 응용 프로그램별 인증 정책
- 기본 인증방식을 선택할 수 있는 기능
- 높은 보안 시스템에서 여러 권한 부여를 요구할 수 있는 기능
- 최종 사용자의 사용 편의성
- 암호가 여러 인증 서비스에 대해 동일한 경우 암호 재입력 없음
- 사용자가 여러 명령을 입력할 필요 없이 여러 인증 서비스에 대해 사용자에게 암호를 요구할 수 있는 기능
- 사용자 인증 서비스에 선택적 옵션을 전달할 수 있는 기능
- 시스템 항목 서비스를 변경할 필요 없이 사이트별 보안 정책을 구현할 수 있는 기능

2.4 PAM 파일과 위치

파일 위치	설명	비고
/lib64/security /lib/security	제공 가능한 PAM 모듈 디렉토리로 실제 PAM Library를 동적으로 인증 모듈을 호출하여 실행한다. (*so)	
/etc/security	PAM 실행에 필요한 설정파일로 /lib64/security에 있는 모듈에 대한 설정 파일. (서비스명.conf)	
/etc/pam.d	PAM 데몬 파일(애플리케이션별 PAM의 설정 파일 위치)로 PAM을 사용하는 응용 프로그램이 설정 파일이 없다면 기본값으로 설정된 설정 파일을 자동으로 사용한다.	

2.5 PAM 라이브러리

라이브러리	설명	비고
pam_permit.so pam_deny.so	항상 성공/실패를 리턴해서 접근을 허용, 거부.	
pam_warn.so	호출한 사용자 및 호스트 정보를 syslog에 기록.	
pam_access.so	사용자 계정, 호스트/도메인을 통해서 시스템 접근을 허용. /etc/security/access.conf에서 설정한다.	
pam_unix.so	암호를 확인하고 변경되는데 사용되는 모듈.	

pam_pwdb.so	/etc/passwd나 shadow 사용시 /etc/shadow 파일에서 계정의 정보를 얻어온다.	
pam_env.so	환경 변수를 지정. /etc/security/pam_env.conf 설정 파일에서 환경 변수를 불러온다. /etc/environment와 /etc/security/pam_env.conf 설정 파일이 비었을 경우 변수 설정 없이 성공 값을 반환한다.	
pam_issue.so pam_motd.so	로그인 시 issue나 message를 출력하며, 성공적으로 로그인한 후 나타난다. /etc/issue, /etc/motd	
pam_tally2.so	로그인 시도 횟수를 세는 모듈로 일정 횟수 이상 실패 시 접근을 차단 및 관 리하는 역할을 수행.	
pam_limits.so	시스템의 자원에 대한 사용자 제한을 설정할 때 주로 사용되는 모듈로 시스 템 로그인 전/후에 수행되는 session 인터페이스와 많이 사용됨. /etc/security/limits.conf	
pam_listfile.so	임의의 파일을 기반으로 특정 서비스에 대해 허가 목록이나 거부 목록을 만 들 때 사용됨. Redhat 기준 /etc/pam.d/vsftpd 파일에 설정되어 ftp 사용자 허용/거부 리 스트를 관리할 때 사용.	
pam_mkhomedir.so	사용자 홈 디렉토리가 없다면 홈 디렉토리를 생성하고, /etc/skel 디렉토리 에 있는 파일들을 복사.	
pam_nologin.so	/etc/nologin 파일이 존재하면 root만 로그인 가능하고, 일반 사용자 로 그인 불가능하며, root로 로그인 시 nologin 파일의 내용이 보여진다.	
pam_rhosts_auth.so	rhost 방식으로 네트워크간 원격 세션을 허용.	
pam_rootok.so	UID가 0인 사용자를 인증하는 모듈로 보통 root가 암호 입력 없이 해당 서 비스에 대한 접근을 허용할 때 주로 사용됨.	
pam_securetty.so	접속하는 계정이 root인 경우 /etc/securetty 파일에 기록된 터미널을 통하 는 경우에만 허용, 나머지 사용자의 경우에는 항상 "성공"한 것으로 처리.	
pam_time.so	시간, 사용자, 그룹, 터미널, 셸 등으로 접근을 제어한다. /etc/security/time.conf	
pam_wheel.so	su와 관련된 /etc/pam.d/su에서 사용, 특정 그룹에 속하지 않은 사용자는 root로의 접근을 거부, deny 옵션을 사용하여 특정 그룹만 접근을 거부할 수 있게 설정 가능하다.	
pam_cracklib.so	입력 받은 암호를 /usr/lib/cracklib_dict에 있는 디렉토리와 비교하여 새 로운 암호를 /etc/security/opasswd에 저장되어 있는 이전 암호 목록과 비 교하여 이전 암호와 비슷한지 확인.	
pam_succeed_if.so	주어진 조건이 참일 경우 성공 값 반환, 인자로 quiet가 들어갈 경우 syslog에 알리지 않는다.	

2.6 PAM 설정하기

우리가 다룰 설정 파일은 /etc/pam.d 디렉토리에 있다. /etc/lib 디렉토리에 있는 특정 모듈에 적용되는
설정 파일을 변경하려면 모듈과 함께 있는 문서를 확인해야 한다.

/etc/pam.d 디렉토리에 있는 각각 PAM 설정 파일의 형식은 다음과 같다.

```
[module-type][control-flag][module-path][module-arguments]
```

1) module-type

PAM이 어떤 타입의 인증이 사용될 것인가를 지정한다.

모듈 타입	설명	비고
auth	사용자 인증에 사용하며, 올바른 비밀번호인지 확인하는 절차를 가진다. 응용 프로그램이 사용자에게 비밀번호를 입력하도록 안내하고, 사용자와 해당 사용자의 그룹에 대한 권한을 인증한다.	
account	사용자의 접근 허가 여부를 확인하며 계정 만료, 특정 시간대에 접근이 허용되었는지 여부를 확인한다. 인증하는 기능이 아니며 현재 시간, 사용자의 위치와 같은 다른 요소들의 접근 권한을 결정하는데, 예로 root 사용자의 로그인은 콘솔로만 한다. 이런 식으로 결정하는 것이다.	
password	비밀번호를 설정하고 확인한다. 비밀번호를 기준에 맞게 변경하도록 하는 모듈을 지정한다.	
session	사용자가 인증 받기 전후에 필요한 홈 디렉토리 마운트, 메일박스 생성 등의 유저 섹션을 구분하기 위해 추가적인 작업을 수행한다. 혹시라도 사용자의 로그인 전후에 수행해야 할 작업이 있다는 내용을 지정한다.	

2) contol-flag

PAM에서 사용되는 모듈들이 결과에 따라 어떠한 동작을 취해야 하는지를 지시한다.

flag	설명	비고
required	인증이 거부되기 전에 PAM이 이 서비스에 등록된 모든 모듈들을 요구함에도 불구하고 실패할 경우 인증을 거부하도록 한다. 해당 모듈은 개별 사용자에게 대한 인증을 반드시 진행해야 한다. 인증이 안될 경우 실패가 반드시 반환되어야 한다.	
requisite	이 모듈을 이용하는 인증이 실패할 경우, 즉시 인증을 거부하도록 한다. required와 비슷하지만 이 플래그가 인증을 실패할 경우 설정 파일에서 이 값 다음으로 나오는 모듈들을 호출되지 않는다. 실패한 결과는 즉시 응용 프로그램으로 전달된다.	
sufficient	이전에 요청되었던 모듈이 실패하더라도 이 모듈에 의해서 인증이 성공할 경우 PAM은 인증을 승인한다. 모듈이 성공 값을 반환하고 설정 파일에서 required와 sufficient 제어 플래그가 필요하지 않다면 PAM은 해당 응용 프로그램에서 성공을 반환한다.	
optional	이 모듈이 성공 또는 실패하는지는 그 모듈이 서비스에 대한 형식에 유일한 모듈일 경우에 해당한다. (성공 여부 상관 X) 이 제어 플래그는 모듈의 결과를 무시한다. PAM으로 다른 모듈을 지속적으로 확인하도록 한다. 확인 작업이 실패하게 되더라도 계속 진행하게 된다. 특정 모듈이 실패하더라도 사용자가 로그인하는 것을 허용하고자 할 때 이 플래그를 사용하면 된다.	
include	이 플래그는 인자(argument)에 지정된 또 다른 설정 파일의 내용이나 지침을 포함시키기 위해 사용된다. 즉, 서로 다른 PAM 설정 파일의 내용을 연결하고 구성하는 방식으로 사용된다.	
substack	이 플래그는 다른 PAM 관련 모듈을 불러오는 것은 include와 동일 하지만, substack은 substack의 동작 결과에 따라 나머지 모듈을 처리하지 않는다	

required/requisite 차이는 required와 requisite 인터페이스 모두 반드시 "성공" 되어야만 PAM을 이용한 인증이 완료되나 보안 및 가용성 측면에서는 분명한 차이가 존재한다. required의 경우 실패를 해도 실패한 지점이나 결과값에 대한 리턴을 하지 않는데, requisite의 경우 실패한 지점과 원인을 리턴하기에 보안 관점에서는 공격자에게 인증이 거부된 원인을 제공하게 되며, 가용성 측면에서는 실패한 원인을 리턴하여 원인 분석을 용이하게 해준다.

3) module-path

PAM에게 어떤 모듈을 사용할 것인지 그리고 그것을 어디서 찾을지를 알려준다. (인증 작업을 수행하는 모듈이 있는 실제 디렉토리 경로를 지정해 준다) 대부분 구성은 로그인 구성 파일의 경우 마찬가지로 모듈의 이름만 가지고 있다. 이와 같은 경우, PAM은 기본 PAM 모듈의 디렉토리(/lib64/security 또는 /lib/security) 모듈을 찾는다.

4) module-arguments

모듈에게 전달되는 인수를 나타낸다. 각각의 모듈들은 각각의 인수를 가지고 있다.

arguments	설명	비고
debug	시스템 로그에 디버깅 정보를 남기다.	
no_warn	응용 프로그램에 경고 메시지를 제공하지 않는다.	
use_first_pass	비밀번호를 두 번 확인하지 않는다. 대신 auth 모듈에서 입력한 비밀번호를 사용자 인증 과정 시에도 재사용 해야 한다. (이 옵션은 auth 및 password 모듈에 해당하는 옵션임)	
try_first_pass	이 옵션은 use_first_pass 옵션과 비슷한데, 사용자는 두 번 비밀번호를 입력할 필요가 없기 때문이다. 하지만 기존의 비밀번호를 다시 입력하도록 되어 있다.	
use_mapped_pass	이 인자는 이전 모듈에서 입력된 텍스트 인증 토큰을 입력 받도록 하는데, 이 값으로 암호화 또는 암호화가 해제된 키 값을 생성한다. 그 이유는 모듈에 대한 인증 토큰 값을 안전하게 저장하거나 불러오기 위함이다.	
expose_account	이 값은 모듈로 하여금 계정 정보를 중요하다고 판단하지 않게 한다. 시스템 관리자에 의해 임의로 설정한 것이라 여겨진다.	
nullok	이 인자는 호출된 PAM 모듈이 null 값의 비밀번호를 입력하는 것을 허용한다.	

2.7 PAM 사용예시(일반)

```
#%PAM-1.0
auth    required pam_securetty.so
auth    required pam_unix.so nullok
auth    required pam_nologin.so
account required pam_unix.so
password required pam_cracklib.so retry=3
password required pam_unix.so shadow nullok use_authtok
session required pam_unix.so
```

첫 번째 라인은 주석으로써 인증과는 무관하게 사용하고 있는 PAM 모듈의 버전을 표기하며, auth interface를 사용하는 2~4번째 라인은 해당 서비스에 대한 인증 시 사용된다.

1) auth required pam_securetty.so

pam_securetty.so 모듈의 경우 사용자가 root로 로그인할 경우 /etc/securetty 파일을 확인하여 해당 파일에 리스팅된 터미널에서 로그인을 시도하는 경우 "성공", 기타의 경우 "실패" 값을 리턴한다.

root외 사용자가 로그인을 시도하는 경우 무조건 "성공" 값을 리턴한다. 이 모듈에 대한 테스트의 결과가 "성공"으로 끝나는 경우 다음 auth 라인으로 이동하여 테스트를 진행한다.

2) auth required pam_unix.so nullok

인증 시도 시 /etc/passwd 및 /etc/shadow 파일의 내용을 기반으로 인증 허용여부를 결정한다. 일반적인 password를 통한 인증을 하는 경우, 이 모듈을 사용한다고 보면 된다.

nullok는 password 입력 시 아무것도 입력하지 않는, null password의 입력을 허용한다는 의미로 대부분의 사용자 인증은 이 라인에서 성공/실패 여부가 결정되며 인증이 성공할 경우 다음 auth 라인으로 진행한다.

3) auth required pam_nologin.so

위 예제에서의 인증작업의 마지막 단계에 해당하며, /etc/nologin 파일의 유무를 체크하여 /etc/nologin 파일이 존재하게 되면 시스템(서비스)에서는 root를 제외한 모든 사용자의 접속을 차단하게 되므로 해당 파일이 생성된 경우 root 사용자가 아니라면 인증이 실패하게 된다.

4) account required pam_unix.so

pam_unix.so 모듈의 경우 account interface 사용 시 사용자의 계정이 활성화/비활성화 되었는지 또는 password 사용기간이 만료되지 않았는지 등을 판별한다.

auth interface 모듈에서 인증이 성공한 경우 account interface에서 계정 사용이 가능한지 여부 등을 확인하는 단계이다.

5) password required pam_cracklib.so retry=3

password 사용기간 만료 등의 이유로 password 변경을 하게 되는 경우 pam_cracklib.so 모듈을 통해 작업이 진행된다.

이 모듈에서는 password 변경 시 정해진 규칙에 따라 취약한 password를 사용하는지, 사전에 등록된 알기 쉬운 단어를 password로 사용하는지 등을 체크하여 취약한 password로의 변경 시도 시 해당 변경 작업을 차단한다.

6) password required pam_unix.so shadow nullok use_authok

PAM을 사용하는 서비스에서 사용자 password를 변경하는 경우 해당되는 내용으로 arguments 중 shadow의 의미는 password 생성/변경 시 shadow password를 사용한다는 것이며, nullok는 null password의 사용을 허용한다는 뜻이다.

7) session required pam_unix.so

사용자가 로그인에 성공하면 사용자 이름, 서비스 타임 등을 로그파일(/var/log/secure) 등에 정상적으로 기록했는지 등을 체크한다.

2.8 PAM 사용예시(/etc/pam.d/su)

```
[root]# cat /etc/pam.d/su
##PAM-1.0
auth            sufficient      pam_rootok.so
# Uncomment the following line to implicitly trust users in the "wheel" group.
#auth           sufficient      pam_wheel.so trust use_uid
# Uncomment the following line to require a user to be in the "wheel" group.
```

#auth	required	pam_wheel.so use_uid
auth	include	system-auth
account	sufficient	pam_succeed_if.so uid = 0 use_uid quiet
account	include	system-auth
password	include	system-auth
session	include	system-auth
session	optional	pam_xauth.so

시스템 내에서 자주 사용하게 되는 su(Switching User) 명령에 대한 설정으로 su 명령의 경우 root에서 타 사용자로 전환 시 별도의 password를 확인하지 않으며, 일반 사용자에서 일반 사용자 또는 일반 사용자에서 root로의 전환 시에만 변경하고자 하는 계정의 password를 확인한다.

1) auth sufficient pam_rootok.so

사용자 전환 시 root의 경우 password 없이 인증을 통과 시켜준다는 의미로 이 모듈의 control-flag가 sufficient로 설정되어 있어 root가 su 명령을 실행할 경우 별도의 인증절차 없이 "성공" 값이 리턴되며 같은 인터페이스(auth)의 나머지 모듈들을 더이상 실행되지 않는다.

root가 아니면 "실패" 값을 리턴하나 sufficient의 경우 실패는 무시되기에 다음 라인으로 넘어 가게 된다.

2) auth include system-auth

root가 아닌 경우 다음 라인인 이 부분이 실행되며, include flag는 모듈이 아닌 다른 PAM 파일에 대한 결과를 사용한다.

system-auth 파일을 열어보면 auth 인터페이스에서 pam_unix.so 모듈이 사용되는 것을 알 수 있는데, 이 모듈이 사용자로부터 입력 받은 ID/PW를 시스템 내 DB와 비교하여 성공/실패 여부를 리턴하는 인증에 가장 중요한 모듈이다.

3) 기타

su 명령과 관련해서는 기타 인터페이스(account, password, session 등)는 거의 사용되지 않는다.

2.9 PAM 사용예시(/etc/pam.d/system-auth)

```
[root] /etc/pam.d > cat system-auth
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        required      pam_tally2.so deny=4 unlock_time=1800 no_magic_root reset
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid >= 500 quiet
auth        required      pam_deny.so

account     required      pam_unix.so
account     sufficient    pam_succeed_if.so uid < 500 quiet
account     required      pam_permit.so

password    requisite     pam_cracklib.so try_first_pass retry=3 type= minlen=8 lcredit=-1
```

dcredit=-1	ocredit=-1	
password	sufficient	pam_unix.so md5 shadow nullok try_first_pass use_authtok
password	required	pam_deny.so
session	optional	pam_keyinit.so revoke
session	required	pam_limits.so
session	[success=1 default=ignore]	pam_succeed_if.so service in crond quiet use_uid
session	required	pam_unix.so

다양한 PAM 관련 설정파일 중 가장 핵심이라 할 수 있는 파일로 Redhat 기준으로 6.0 이상 버전의 경우 system-auth와 password-auth 두 파일이 존재하는데, 둘 중 하나만 사용되는 것이 아닌, 타 PAM 관련 파일 들마다 참조하는 파일이 다르므로 그때 그때 확인을 해야 한다.

예를 들어, /etc/pam.d/su에서는 인증에 대한 모듈로 system-auth를 참조하며, /etc/pam.d/sshd에서는 인증에 대한 모듈로 password-auth를 참조한다.

1) auth required pam_env.so

pam_env.so 모듈을 사용하여 인증을 확인한다. 성공 시 남은 모듈 확인, 실패 시 남은 모듈을 확인 하지만 접근은 실패(pam_env.so는 설정 파일에서 환경 변수를 불러온다. /etc/environment, /etc/security/pam_env.conf)한다.

2) auth required pam_tally2.so deny=4 unlock_time=1800 no_magic_root reset

pam_tally2.so는 로그인 시도 횟수를 세는 모듈로 일정 횟수 이상 실패 시에는 접근을 차단 및 관리해 주는 역할을 담당한다.

deny=4 : 로그인 시도가 4번 실패하면 추가 시도를 차단함.

unlock_time=1800 : 미리 지정한 일정 횟수 이상 로그인에 실패했을 경우 1800초(30분) 동안 계정이 잠김.

no_magic_root reset : 로그인이 성공한 경우 기존 실패 횟수를 0으로 초기화.

3) auth sufficient pam_unix.so nullok try_first_pass

pam_unix.so는 입력 받은 password 등을 시스템 내에 DB와 비교하여 결과를 판별하는 역할을 수행하며, nullok는 password를 공백으로 입력하는 것을 허용한다는 뜻이다.

try_first_pass는 이전 단계에서 입력 받은 password가 있으면 그걸 사용해서 테스트를 수행하며, 입력 받은 password가 없을 경우 새롭게 password를 입력 받아 사용한다는 의미다.

이 모듈의 경우 sufficient로 flag가 설정되어 있어 테스트 결과가 "성공"으로 확인되면 다음에 위치한 auth requisite pam_succeed_if.so, auth required pam_deny.so에 대해서는 더이상 테스트를 수행하지 않는다.

4) auth required pam_deny.so

pam_deny.so는 테스트 시도에 대해 무조건 "실패" 값을 리턴한다. 즉, auth로 설정된 위 4개 라인에 대해 테스트가 실패할 경우 마지막으로 pam_deny.so에서 인증을 거부하게 된다.

5) auth requisite pam_succeed_if.so uid >= 500 quiet

uid가 500 보다 크거나 같으면 인증을 수행하며, quiet는 로그를 남기지 않는다.

6) password requisite pam_cracklib.so try_first_pass retry=3 type= minlen=8 lcredit=-1 dcredit=-1 ocredit=-1

pam_cracklib.so는 Dictionary에 등록된 단어를 이용한 password 설정 등을 방지하기 위한 비교 및 검사를 수행할 때 사용(password-auth, system-auth 등에 필수적으로 포함됨)한다.

retry=3 : 새로운 password를 설정할 때 물어 보는 횟수를 지정, 변경하고자 하는 password가 기준에 미달 (복잡도, 최소길이 등)할 경우 몇 번의 입력을 추가로 허용할지를 설정.

minlen=8 : password 최소길이를 지정, 글자수와 크레딧(credit) 값이 조합되어 계산됨.

type= : 새로운 password를 입력할 때 화면에 표기되는 문구를 지정, 없으면 기본값.

dcredit=-1 : 숫자가 갖는 기본 크레딧 값을 지정, 기본값은 1이며 -1로 설정된 경우 password에 반드시 숫자가 하나 이상 있어야 한다는 의미.

(ucredit: 대문자, lcredit: 소문자, ocredit: 기타 특수문자)

만약, 크레딧(예, ucredit) 값이 -1이 아닌 2로 지정되었다면 password 최소길이가 8글자로 설정되었을 때 대문자 하나는 2의 값을 갖게됨. 예를 들면 Abcdefg라는 password는 얼핏 보기에 총 길이가 7문자이지만 대문자 A는 2크리딧 값을 갖게 되어, 이 문자열은 8크리딧이 되고 이에 password 최소길이 8글자라는 조건을 충족하게 된다.

2.10 PAM 사용예시(/etc/pam.d/sshd)

SSH 접속을 제한하기 위해 다음과 같이 vi 명령어를 사용하여 다음과 같은 내용을 추가한다.

```
$ vi /etc/pam.d/sshd
auth required pam_listfile.so item=user sense=deny file=/etc/ssh/sshusers onerr=succeed
```

추가 후 /etc/ssh/sshusers 파일을 생성한 후 SSH 접속을 제한할 사용자를 다음과 같이 입력한다. (파일 경로와 파일명은 임의로 설정)

```
$ vi /etc/ssh/sshusers
nuriapp
barokey
baropam
barofds
```

저장 후 해당 계정으로 SSH 접속을 시도할 경우 "Access denied" 메시지를 출력하며 접속이 불가능하다.

2.11 SSH(Secure Shell)

Secure Shell은 개방형 소스 SSH 제품인 OpenSSH 제품을 기반으로 한다. (<http://www.openssh.org>)

Secure Shell은 비보안 네트워크에서 클라이언트와 원격 호스트 간의 보안 연결을 가능하게 한다.

이 보안 연결의 주요 속성은 다음과 같다.

- 클라이언트와 원격 호스트 둘 다에 대한 강력한 인증
- 클라이언트와 원격 호스트 간의 통신에 대한 강력한 암호화 및 공개 키 암호화
- 클라이언트와 원격 호스트에서 명령을 실행하는데 사용할 보안 연결

Secure Shell은 telnet, remsh, rlogin, ftp, 및 rcp와 같은 자주 사용하는 함수와 명령의 보안 대체 항목을 제공한다.

1) Secure Shell의 주요 보안 기능

Secure Shell의 주요 보안 기능은 다음과 같다.

① 강력한 암호화

클라이언트와 원격 호스트 간의 모든 통신은 Blowfish, 3DES, AES 및 arcfour와 같은 특허 없는 암호화 알고리즘을 사용하여 암호화된다. 암호 등의 인증 정보는 네트워크에서 일반 텍스트로 전송되지 않는다. 또한 암호화는 강력한 공개 키 기반 암호화와 더불어 잠재적 보안 공격을 차단한다.

② 강력한 인증

Secure shell은 클라이언트와 서버 간의 강력한 인증 방법 집합을 지원한다. 인증은 양방향일 수 있다. 서버는 클라이언트를 인증하고 클라이언트도 서버를 인증한다. 이렇게 하면 다양한 보안 문제로부터 세션을 보호할 수 있다.

③ 포트 전달

클라이언트와 원격 호스트 간의 TCP/IP 연결 리디렉션(및 그 반대)을 포트 전달 또는 SSH 터널링이라고 한다. Secure Shell은 포트 전달을 지원한다. 예를 들어, 포트 전달을 사용하여 클라이언트와 서버 간의 ftp 트래픽(또는 전자 메일 클라이언트와 POP/IMAP 서버 간의 전자 메일 트래픽)을 리디렉션할 수 있다. 클라이언트가 직접 서버와 통신하는 대신 보안 채널을 통해 트래픽을 sshd 서버로 리디렉션할 수 있으며, 그런 다음 sshd 서버에서 트래픽을 실제 서버 시스템의 지정된 포트로 전달할 수 있다.

2) Secure Shell의 소프트웨어 구성 요소

Secure Shell 소프트웨어는 클라이언트 및 서버 구성 요소로 집합으로 이루어져 있다.

구성요소	설명	위치	대응하는 비보안 구성요소
ssh	Secure Shell을 클라이언트는 telnet 및 remsh의 보안 대체 항목이며 보안 기능이 있는 remsh와 가장 유사함.	클라이언트	remsh, telnet, rlogin
slogin	ssh에 대한 심볼릭 링크임.	클라이언트	remsh, telnet, rlogin
scp	클라이언트 보안 복사 및 서버 보안 복사를 수행함,	클라이언트 및 서버	rcp
sftp	보안 ftp 클라이언트임	클라이언트	ftp
sshd	보안 Shell 데몬임.	서버	remshd, telnetd, rlogind
sftp-server	보안 ftp 데몬임.	서버	ftpd
ssh-rand-helper	sshd가 서버에서 /dev/random 또는 /dev/urandom을 찾을 수 없을 때는 사용되는 Random Number Generator. OS 커널에 상주하는 Random Number Generator인 rng가 포함되어 있다. rng가 구성 해제되어 있으면 sshd는 prngd를 사용함.	서버	해당 사항 없음
ssh-agent	클라이언트에서 서버로의 "자동" 키 기반 로그인 도구임.	클라이언트 및 서버	Rhosts 파일 메커니즘
ssh-add	클라이언트의 키 쌍을 ssh-agent에 알리는 도구임.	클라이언트	해당 사항 없음
ssh-keygen	공개 키 인증을 위해 키 쌍을 생성하는 도구임.	클라이언트	해당 사항 없음
ssh-keyscan	Secure Shell 데몬(sshd)을 실행하는 호스트 집합에 대한 공개 키를 수집하는 클라이언트 도구임.	클라이언트	해당 사항 없음

ssh-keysign	호스트 기반 인증 중에 필요한 디지털 서명을 생성하는 도구임. ssh()에서 로컬 호스트 키 호스트 기반 인증에 액세스하는데 사용됨.	클라이언트	해당 사항 없음
-------------	--	-------	----------

3) Secure Shell 실행

위 표에서 나열된 Secure Shell 클라이언트를 실행하기 전에 먼저 Secure Shell 서버 데몬 sshd를 시작한다.

sshd 데몬은 서버 시스템의 /etc/ssh 디렉토리에 있는 sshd_config 파일에서 초기 값을 가져온다.

sshd_config에 있는 가장 중요한 구성 지시어 중 하나는 sshd 데몬에서 지원하는 인증 방법 집합이다.

- ssh 클라이언트 실행

ssh 클라이언트 응용 프로그램은 sshd 서버와의 소켓 연결을 설정한다.

sshd 서버는 자식 sshd 프로세스를 시작한다.

이 자식은 연결 소켓을 상속 받고 선택된 인증 방법을 기반으로 클라이언트를 인증한다.

인증에 성공한 경우에만 성공적으로 보안 클라이언트 세션이 설정된다.

세션이 만들어진 후 모든 후속 통신은 클라이언트와 이 자식 sshd 프로세스 간에 직접 이루어진다.

이제 클라이언트는 서버에서 원격 명령을 실행할 수 있다.

ssh 클라이언트의 명령 요청이 있을 때마다 자식 sshd 프로세스에서 해당 명령을 실행할 Shell 프로세스를 시작한다.

간단히 말해서 실행 중인 ssh 클라이언트-서버 세션은 다음 프로세스로 구성된다.

① sshd 서버에 연결된 모든 클라이언트 시스템에는 현재 이 클라이언트 시스템에서 설정된 각 ssh 연결에 대한 하나의 ssh 클라이언트 프로세스가 있다.

② 서버 시스템에는 하나의 부모 sshd 프로세스와 서버에 연결된 동시 ssh 클라이언트 개수 만큼의 자식 sshd 프로세스가 있다. 서버에 권한 분리가 활성화되어 있으면 서버에서 실행되는 자식 sshd 프로세스의 수가 두 배로 증가한다.

③ ssh 클라이언트에서 명령 실행 요청이 있을 때마다 서버 시스템의 해당 자식 sshd 프로세스는 Shell 프로세스를 시작하고 Linux 파이프를 사용하여 명령 요청을 이 Shell 프로세스로 전달한다. 이 Shell 프로세스는 Linux 파이프를 사용하여 명령 실행 결과를 자식 sshd 프로세스로 반환하고 명령 실행이 완료되면 종료된다.

- sftp 클라이언트 실행

sftp 클라이언트 응용 프로그램은 sftp 클라이언트 응용 프로그램이 ssh 클라이언트를 시작 하도록 하고 Linux 파이프를 사용하여 이 클라이언트와 통신한다. 그런 다음 ssh 클라이언트는 sshd 서버와의 소켓 연결을 설정한다.

서버 상호 작용의 나머지 부분은 ssh 클라이언트의 경우와 유사하다. 차이점은 원격 명령을 실행한 Shell 을 시작하는 대신 자식 sshd 프로세스가 sftp-server 프로세스를 시작한다는 것이다. 이 sftp 세션 중의 모든 후속 통신은 다음 프로세스 간에 발생한다.

- ① Linux 파이프를 사용하여 클라이언트 시스템에서 sftp 클라이언트와 ssh 클라이언트 간에
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Linux 파이프를 사용하여 자식 sshd 프로세스와 sftp 서버 프로세스 간에

- scp 클라이언트 실행

scp 클라이언트의 경우는 sftp 클라이언트 실행과 거의 동일하다. 차이점은 sftp-server 프로세스를 시작하는 대신 자식 sshd 프로세스가 scp 프로세스를 시작한다는 것이다. scp 세션 중의 모든 후속 통신은 다음 프로세스간에 발생한다.

- ① 클라이언트 시스템에서 scp 클라이언트와 ssh 클라이언트 간에, Linux 파이프 사용
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Linux 파이프를 사용하여 자식 sshd 프로세스와 scp 서버 프로세스 간에

4) Secure Shell 권한 분리

Secure Shell은 권한 분리 기능을 통해 보다 향상된 수준의 보안을 제공한다. 부모 sshd 및 자식 sshd 프로세스는 권한이 부여된 사용자로 실행된다. 권한 분리를 활성화하면 사용자 연결당 하나의 추가 프로세스가 시작된다.

ssh 클라이언트가 권한 분리에 대해 구성된 sshd 서버에 연결하면 부모 sshd 프로세스가 권한이 부여된 자식 sshd 프로세스를 시작한다. 권한 분리를 활성화하면 자식 sshd 프로세스는 권한이 없는 자식 sshd 프로세스를 추가로 시작한다. 권한이 없는 자식 sshd 프로세스는 연결 소켓을 상속 받는다. 클라이언트와 서버 간의 모든 후속 통신은 권한이 없는 이 자식 sshd 프로세스를 사용하여 이루어진다.

클라이언트의 원격 명령 실행 요청은 대부분 비권한이며 권한이 없는 이 자식 sshd 프로세스에서 시작된 Shell에 의해 처리된다. 권한이 없는 자식 sshd 프로세스에서 권한이 부여된 기능을 실행해야 하는 경우 Linux 파이프를 사용하여 권한이 부여된 부모 sshd 프로세스와 통신한다.

권한 분리는 침입자에 의한 잠재적 손상을 제한하는데 도움이 된다. 예를 들어, Shell 명령을 실행하는 동안 버퍼 오버플로 공격이 발생할 경우 권한이 없는 프로세스 내에서 제어되므로 잠재적 보안 위험이 제한된다.

권한 분리는 Secure Shell의 기본 구성이다. sshd_config 파일에서 "UsePrivilegeSeparation NO"를 설정하여 권한 분리를 해제할 수 있다. 잠재적 보안 위험이 있으므로 권한 분리를 해제할 경우 신중하게 고려해야 한다.

5) Secure shell 인증

Secure Shell은 다음 인증 방법을 지원한다.

- GSS-API(Kerberos 기반 클라이언트 인증)
- 공개 키 인증
- 호스트 기반 인증
- 암호 인증

클라이언트는 원격 sshd 데몬과 연결될 때 원하는 인증 방법(앞에 나열된 방법 중 하나)을 선택하고 연결 요청의 일부로 적절한 자격 증명을 제공하거나 서버에서 보낸 프롬프트에 응답한다. 모든 인증 방법이 이

런 방식으로 작동한다.

서버가 성공적으로 연결을 설정하려면 클라이언트로부터 적절한 키, 암호구, 암호 또는 자격 증명을 받아야 한다.

sshd 인스턴스에서 보안 요구 사항을 기반으로 지원되는 인증 방법 중 일부만 지원하도록 선택할 수 있다.

Secure Shell은 앞에 나열된 인증 방법을 지원하지만 시스템 관리자가 환경의 특정 보안 요구 사항을 기반으로 sshd 인스턴스에서 제공되는 인증 방법을 제한할 수 있다. 예를 들어, Secure Shell 환경에서 모든 클라이언트가 공개 키 또는 Kerberos 방법을 사용하여 인증해야 한다고 지정할 수 있으며, 이로 인해 나머지 방법은 비활성화될 수 있다. 지원되는 인증 방법 활성화 및 비활성화는 sshd_config 파일에 지정된 구성 지시어를 통해 이루어진다.

ssh 클라이언트 연결 요청이 있으면 서버는 먼저 지원되는 인증 방법 목록으로 응답한다. 이 목록은 sshd 서버에서 지원하는 인증 방법과 이러한 방법이 시도되는 시퀀스를 나타낸다. 클라이언트는 이러한 인증 방법을 하나 이상 생각할 수 있다. 클라이언트에서 방법이 시도되는 시퀀스를 변경할 수도 있다. 이렇게 하려면 클라이언트 구성 파일 /etc/ssh/ssh_config의 구성 지시어를 사용한다.

Secure Shell에서 지원하는 인증 방법은 다음과 같이 요약되어 있다.

① GSS-API(Kerberos 기반 클라이언트 인증)

Kerberos 기반 클라이언트 인증인 GSS-API(Generic Security Sservice application Programming Interface)를 사용하는 경우 클라이언트가 미리 Kerberos 자격 증명을 받아야 하며, 해당 클라이언트 디렉토리에 Kerberos 구성 파일이 있어야 한다.

클라이언트는 sshd 데몬과 연결될 때 연결 시 자격 증명을 제공한다.

서버는 이러한 자격 증명을 이 특정 사용자의 자격 증명 복사본과 일치 시킨다.

또한 선택적으로 클라이언트 호스트 환경의 타당성을 설정할 수 있다.

② 공개 키 인증

공개 키 인증을 사용하려면 Secure Shell 환경에 다음 설정이 있어야 한다..

클라이언트와 서버 둘 다에 키 쌍이 있어야 한다. 모든 ssh 클라이언트와 모든 sshd 서버가 ssh-keygen 유틸리티를 사용하여 자체적으로 키 쌍을 생성해야 한다.

클라이언트는 통신해야 하는 모든 sshd 서버에 해당 공개 키를 알려야 한다. 이렇게 하려면 각 클라이언트의 공개 키를 모든 관련 서버의 미리 지정된 디렉토리에 복사한다.

클라이언트는 통신해야 하는 모든 서버의 공개 키를 구해야 한다. 클라이언트는 ssh-keyscan 유틸리티를 사용하여 공개 키를 받는다.

이 설정이 완료 되면 sshd 서버에 연결하는 ssh 클라이언트가 공개 키와 개인 키를 사용하여 인증된다.

Secure Shell은 공개 키 인증을 단순화하는 추가 기능을 제공한다. 일부 환경에서는 항상 암호 프롬프트에 응답할 필요가 없도록 설정할 수 있다. 둘 다 클라이언트 시스템에서 실행되는 ssh-agent 및 ssh-add 프로세스를 조합해서 사용하면 암호에 응답하지 않아도 된다. 클라이언트는 ssh-add 유틸리티를 통해 ssh-agent 프로세스에 모든 키 정보를 등록한다. 그런 다음 클라이언트와 서버 간의 공개 키 인증은 sshd 데몬이 클라이언트와 상호 작용할 필요 없이 ssh-agent에 의해 수행된다.

③ 호스트 기반 및 공개 키 인증

호스트 기반 및 공개 키 인증은 보다 안전한 공개 키 인증 방법의 확장이다.

클라이언트와 서버 둘 다의 키 쌍이 있는 것 외에도 이 방법을 사용하면 클라이언트 환경에서 통신할 서버를 제한할 수 있다.

클라이언트의 홈 디렉토리에 `.rhosts` 파일을 만들어 이 제한을 구현한다.

④ 암호 인증

암호 인증 방법은 단일 사용자-ID 및 암호 기반 로그인을 사용한다. 이 로그인은 `/etc/passwd`에 지정된 사용자 로그인을 기반으로 하거나 PAM 기반일 수 있다.

Secure Shell은 서버 시스템에서 사용할 수 있는 PAM 모듈과 완전히 통합된다. 이 목적을 위해 `/etc/ssh/sshd_config` 파일에 UsePAM 구성 지시어가 있다. YES로 설정하면 클라이언트에서 암호 인증 요청이 있을 때마다 sshd는 PAM 구성 파일(`/etc/pam.conf`)을 확인한다. 그런 다음 구성된 PAM 모듈을 통해 암호 인증이 성공할 때까지 순서대로 수행한다.

PAM 인증을 무시하려면 UsePAM 지시어를 NO로 설정한다. 그러면 클라이언트에서 암호 인증 요청이 있을 때 sshd가 서버의 PAM 구성 설정을 무시한다. 대신 sshd는 `gwtppnam()` 라이브러리 호출을 직접 호출하여 사용자 암호 정보를 가져온다.

Secure Shell은 PAM_UNIX, PAM_LDAP 및 PAM_KERBEROS를 사용하여 테스트 되었다. 또한 PAM_DCE 및 PAM_NTLM과 같은 다른 PAM 모듈에서 작동한다.

6) 통신 프로토콜

Secure Shell 사용자는 SSH-1 또는 SSH-2 프로토콜을 사용하여 원격 sshd 데몬과 연결할 수 있다. SSH-2가 더 안전하므로 SSH-1 대신 권장한다.

7) Secure Shell에서 사용하는 기능의 일부 목록

Secure Shell은 실제로 Shell이 아니라 호스트에서 안전하게 원격 shell 세션을 실행하기 위해 클라이언트와 원격 호스트 간에 보안 연결을 만드는 매커니즘이다. 보안 연결을 설정하기 위해 Secure Shell에서 인증과 세션 생성을 대부분 수행한다. Secure Shell에서 사용하는 기능의 일부 목록은 다음과 같다.

① login 시도 기록

telnet 또는 remsh와 마찬가지로 Secure Shell은 성공한 세션과 실패한 세션을 각각 `/var/log/wtmp` 및 `/var/log/btmp` 파일에 기록한다.

② PAM 모듈

Secure Shell은 클라이언트 세션에 대해 PAM 인증을 사용할 수 있다. PAM 인증을 선택하면 Secure Shell은 `/etc/pam.conf` 파일을 사용하고 인증을 위해 해당 PAM 모듈을 호출한다.

③ `/etc/lib` 파일 사용

로그인 동작 암호 및 기타 보안 구성을 정의하는 속성이 들어 있는 시스템 범위 구성 파일이다. 몇 가지 제한은 있지만 Secure Shell에서 이러한 속성을 사용할 수 있다.

④ Shadow passwd

Secure Shell은 Shadow passwd 기능과 통합된다.

⑤ 컨트롤 시스템 로그(syslog)

Secure Shell은 syslog를 사용하여 중요한 메시지를 남긴다.

⑥ 감사 로깅

Secure shell은 해당 코드로 감사 로깅(트러스트된 모드)을 구현했다.

8) 비밀번호 없이 SSH에 접속하는 방법

SSH 접속 시 비밀번호 없이 접속하는 방법은 다음과 같다.

① ssh key 생성 (없는 경우만)

```
$ /etc/rc.d/sshd keygen
```

② sshd 데몬 활성화

```
$ /etc/rc.d/sshd enabled
```

③ sshd 데몬 start|stop|restart

```
$ /etc/rc.d/sshd start|stop|restart
```

④ 서버 접속

```
$ ssh -i ~/.ssh/id_rsa [id]@[address]
```

2.12 PAM 프로그래밍

아래 소스 코드는 PAM 모듈의 기본 형태로 "#if 0"으로 막혀 있는 부분에 통제를 할 수 있는 코드를 넣어 두면 원하는 동작(차단 또는 허용, 로그 기록)을 쉽게 처리할 수 있다.

```
/* pam_test.c */

#include <stdio.h>
#include <stdlib.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>

#ifdef PAM_MODULE_ENTRY
PAM_MODULE_ENTRY("pam_test");
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN extern
#endif

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}
```

```

}

PAM_EXTERN int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

/* expected hook, this is where custom stuff happens */
PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    #if 0
        // 아래 조건을 만족하지 못할 경우 로그인 실패로 처리함
        if (check_auth_pam(pamh) < 0)
            return PAM_PERM_DENIED;
    #endif
    return PAM_SUCCESS;
}

```

위에 있는 pam_sm...으로 된 함수들이 사용자가 임의로 추가한 함수가 아니라 PAM에서 기본 함수들로서 해당 함수 부분에 원하는 동작 코드를 넣도록 되어 있다.

PAM 모듈 자체가 기존에 동작하는 telnet이나, ssh, login 등의 바이너리에 동적 라이브러리 형태로 붙기 때문에 위 코드에 삽입한 코드들은 로그인과 동시에 해당 서버로의 접속자의 모든 상황들을 감시하고 제어할 수 있다.

2.13 PAM 컴파일 방법

PAM 인증 모듈은 FreeBSD의 특성별 OS에서 제공하는 최적화 옵션에 따라 컴파일 하는 방법은 다음과 같다.

```

sudo /usr/bin/cc -std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o baro_auth.o baro_auth.c
sudo /usr/bin/cc -g -o baro_auth baro_auth.o -lc
sudo /usr/bin/cc -std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o base64.o base64.c
sudo /usr/bin/cc -std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o barocrypt.o barocrypt.c
sudo /usr/bin/cc -std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o pam_baro_auth.o pam_baro_auth.c
sudo /usr/bin/cc -shared -g -o pam_baro_auth.so pam_baro_auth.o base64.o barocrypt.o -lpam -
L/usr/lib64 -lssl -lcrypto -lc -lz

```

PAM 프로그램 컴파일할 때 반드시 sudo 명령어를 사용하여 컴파일 해야 한다. 그렇지 않으면 다음과 같은 오류가 발생한다.

```
logmsg: pri 43, flags 0, from baropam, msg Apr 12 13:18:40 sshd[26319]: in
openpam_check_desc_owner_perms(): /usr/baropam/pam_baro_auth.so: insecure ownership or
permissions
Apr 12 13:18:40 baropam sshd[26319]: in try_module(): /usr/baropam/pam_baro_auth.so: Operation
not permitted
```

sudo 명령어는 유닉스 및 유닉스 계열 운영 체제에서 다른 사용자의 보안권한과 관련된 프로그램을 구동할 수 있게 해주는 프로그램이다.

이것은 substitute user do (다른 사용자의 권한으로 명령을 이행하라, 는 뜻이다.) 의 줄임말이다.

기본적으로 Sudo는 사용자 비밀번호를 요구하지만 루트 비밀번호(root password)가 필요할 수 도 있고, 한 터미널에 한번만 입력하고 그 다음부터는 비밀번호가 필요 없다.

sudo는 각 명령줄에 사용할 수 있으며 일부 상황에서는 관리자 권한을 위한 슈퍼유저 로그인(superuser login)을 완벽히 대신하며, 주로 우분투, 리눅스와 애플의 OS X 에서 볼 수 있다.

PAM을 컴파일하기 위해서는 암호화 모듈에 "openssl" 라이브러리를 사용하므로 반드시 "openssl"을 다음과 같은 명령어로 설치해야 한다.

```
> pkg install openssl
```

2.14 PAM 환경설정

보통 Linux의 경우는 /etc/pam.d/ 디렉토리 안에 서비스마다 별도 설정을 해줘야 한다. 단 설정 양식은 아래와 비슷하다.

```
login auth required /usr/lib/security/pam_test.so common
ftp auth required /usr/lib/security/pam_test.so ftp
sshd auth required /usr/lib/security/pam_test.so ssh
telnet auth required /usr/lib/security/pam_test.so telnet
rsh auth required /usr/lib/security/pam_test.so rsh
rlogin auth required /usr/lib/security/pam_test.so rlogin
dtlogin auth required /usr/lib/security/pam_test.so dtlogin
```

PAM설정이 되어 있는 경우 PAM모듈 프로그램이 잘못되거나 환경설정 파일 수정에 실수가 있는 경우, 그 순간부터 신규 접속은 모두 차단된다.

따라서 반드시 작업 전에 telnet/ssh 창을 접속해 둔 다음 테스트 하시고, 테스트 중에 문제가 발생한 경우 이전에 붙여 놓았던 창을 이용하여 복원해야 한다.

2.15 PAM 디버그 사용

PAM (Pluggable Authentication Modules) 라이브러리는 실행 중에 디버그 정보를 제공 할 수 있다.

시스템이 디버그 출력을 수집하도록 설정한 후에는 수집된 정보를 사용하여 PAM API 호출을 추적하고 현재 PAM 설정에서 오류 지점을 확인할 수 있다.

PAM 디버그 출력을 사용하려면 다음 단계를 완료 해야 한다.

step 1) /etc/syslog.conf 파일을 편집하여 원하는 우선 순위 수준에서 syslog 메시지를 로깅 할 파일을 식별한다.

예를 들어 PAM 디버그 수준 메시지를 /var/log/auth.log 파일에 보내려면 다음 텍스트를 syslog.conf 파일의 새 행으로 추가해야 한다.

```
*.debug /var/log/auth.log
```

step 2) touch 명령을 사용하여 1 단계에서 참조한 출력 파일 (/var/log/auth.log)이 존재하지 않으면 작성해야 한다.

step 3) 구성 변경 사항이 인식되도록 syslogd 데몬을 다시 시작하려면 다음 단계를 완료 해야 한다.

```
> /etc/rc.d/syslogd start | stop | restart
```

PAM 응용 프로그램이 다시 시작되면 /etc/syslog.conf 구성 파일에 정의된 출력 파일에 디버그 메시지가 수집된다.

2.16 PAM 테스트 방법

PAM 프로그램 컴파일 또는 PAM 구성 환경 변경 후 sshd를 restart하지 않고 테스트 할 수 있는 방법은 다음과 같이 진행 할 수 있다.

1) sshd 데몬 기동하기

sshd 데몬(22000 포트)을 디버깅 모드로 띄우기 위하여 다음과 같은 명령어를 수행한다.

```
> /usr/sbin/sshd -D -p22000 -d
```

2) ssh로 접속하기

sshd 데몬(22000 포트)에 ssh로 접속하기 위하여 다음과 같은 명령어를 수행한다.

```
> ssh -v -p22000 root@10.0.2.15
```

3. HP-UX

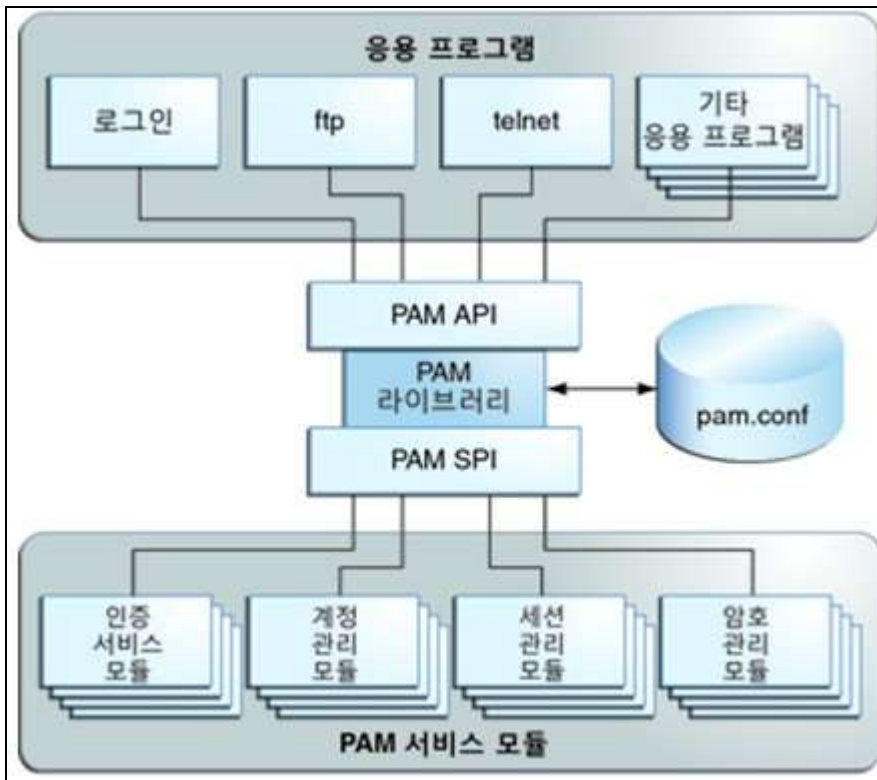
3.1 PAM 개요

PAM(Pluggable Authentication Module, 플러그 가능한 인증 모듈)은 Linux/Unix 시스템에서 서비스를 재 컴파일하지 않고, 다양한 인증 기술을 시스템 항목 서비스에 접목할 수 있도록 해주는 프레임워크로 중앙 집중적인 인증 매커니즘을 지원하는 것이다. 게다가 시스템의 기본적인 인증 기법을 제공하여 이것을 사용하면 응용 프로그램 개발자 뿐만 아니라 시스템 관리자들이 인증을 유연성 있게 관리할 수 있도록 도와 준다.

전통적으로 시스템 자원에 대한 접근을 관리하는 프로그램들은 내장된 메커니즘에 의해 사용자 인증 과정을 수행한다. 이러한 방식은 오랫동안 이루어졌지만 이러한 접근 방식은 확장성이 부족하고 매우 복잡하다. 그렇기 때문인지 이러한 인증 매커니즘을 끌어내기 위한 수많은 해킹 시도가 있었다.

Solaris의 방식을 따라서 Unix/Linux 사용자들은 그들만의 PAM을 구현하는 방식을 찾았다.

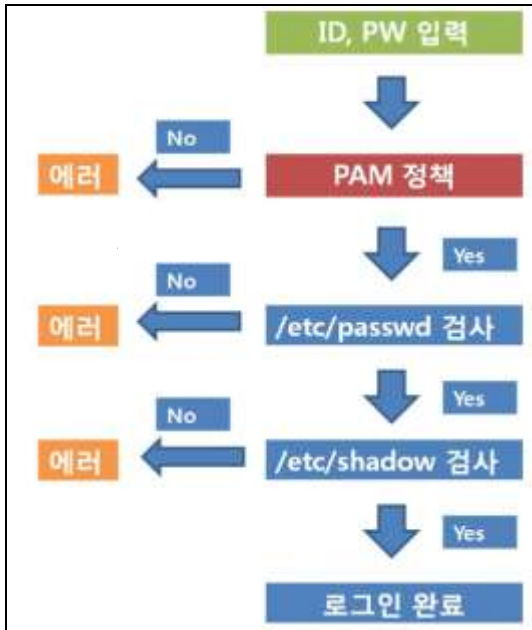
PAM의 아키텍처는 다음과 같다.



PAM의 기본 원리는 응용 프로그램이 password 파일을 읽어 오는 대신 PAM이 직접 인증을 수행 하도록 하는 것이다. PAM은 시스템 관리자가 원하는 인증 매커니즘이 무엇이든 상관하지 않는다.

여러 사이트에서 선택 받은 인증 매커니즘은 아직도 password 파일이다. 왜 그럴까? 우리가 원하는 것을 해주기 때문이다. 대부분의 사용자는 password 파일이 필요한 것이 무엇인지 이미 알고 있다. 그리고 원하는 작업을 수행하는데 있어 이미 그 기능이 검증되었기 때문일 것이다.

PAM의 인증 절차는 다음과 같다.



3.2 PAM 동작 원리

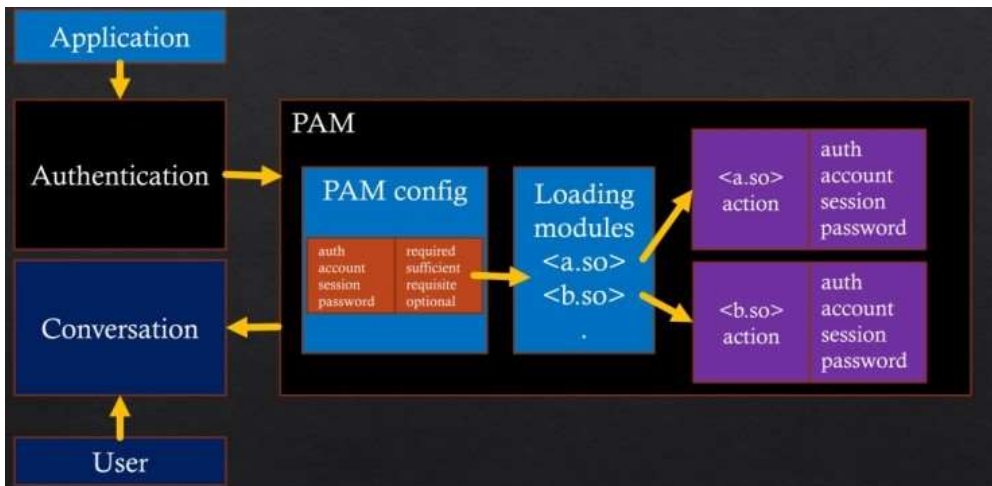
PAM은 Windows 환경의 DDL(Dynamic Link Library)과 같은 것이다. Library로 프로그램이 어떤 사용자에게 대한 인증을 수행하려면 PAM Library가 있는 함수를 호출한다. PAM은 해당 함수의 Library를 제공하여 응용 프로그램이 특정 사용자를 인증하도록 요청할 수 있다.

PAM을 통해 /etc/passwd 파일이나 /etc/shadow 파일을 확인하거나 또는 보다 복잡한 확인 작업을 수행하기도 하는데, 예로는 LDAP 서버에 접속하는 것이다.

확인 작업을 마치고, 인증 여부를 결정하게 되면 "인증됨/인증되지 않음"의 메시지를 자신을 호출한 응용 프로그램에 전송한다.

간단한 확인 작업이라고 했는데, 그 과정이 많아 보일 수가 있다. 여기에 나오는 각 모듈은 크기가 작고 작업 수행 시간이 매우 빠르다. 이것은 매우 놀랍기도 하지만 PAM을 사용하게 된다.

PAM의 동작은 프로그램의 실행 하자마자 실행되는 보안 설정 파일들의 프로그램들이다.



PAM의 동작 원리는 다음과 같다.

- 인증이 필요한 프로그램 동작 시 PAM 라이브러리 호출
- PAM설정 파일을 참조
- 참조한 파일의 내용을 바탕으로 모듈 동작
- 프로그램에 동작 결과를 반환하여 인증여부 결정.

3.3 PAM 사용 이점

PAM을 사용하면 사용자 인증을 위한 시스템 항목 서비스(예: ftp, login, telnet, rsh) 사용을 구성할 수 있다. PAM이 제공하는 몇 가지 이점은 다음과 같다.

- 유연한 구성 정책
- 응용 프로그램별 인증 정책
- 기본 인증방식을 선택할 수 있는 기능
- 높은 보안 시스템에서 여러 권한 부여를 요구할 수 있는 기능
- 최종 사용자의 사용 편의성
- 암호가 여러 인증 서비스에 대해 동일한 경우 암호 재입력 없음
- 사용자가 여러 명령을 입력할 필요 없이 여러 인증 서비스에 대해 사용자에게 암호를 요구할 수 있는 기능
- 사용자 인증 서비스에 선택적 옵션을 전달할 수 있는 기능
- 시스템 항목 서비스를 변경할 필요 없이 사이트별 보안 정책을 구현할 수 있는 기능

3.4 PAM 파일과 위치

인증 방법은 다음 PAM 시스템 파일을 사용하여 시스템 범위 및 개별 사용자별로 지정한다.

파일 위치	설명	비고
/usr/lib/security	제공 가능한 PAM 모듈 디렉토리로 실제 PAM Library를 동적으로 인증 모듈을 호출하여 실행한다. (*so)	
/etc/pam.conf	PAM 데몬 파일(애플리케이션별 PAM의 설정 파일 위치)로 PAM을 사용하는 응용 프로그램이 설정 파일이 없다면 기본값으로 설정된 설정 파일을 자동으로 사용한다.	
/etc/pam_user.conf	개별 사용자 파일로 특정 사용자의 서비스 모듈에서 사용할 옵션을 정의한다. (옵션)	

3.5 PAM 라이브러리

PAM 서비스 모듈은 공유 라이브러리에 의해 구현된다. PAM을 사용하면 HP-UX에서 여러 인증 기술을 함께 사용할 수 있다. /etc/pam.conf 구성 파일은 사용할 인증 모듈을 결정한다. PAM 라이브러리는 다음과 같다.

라이브러리	설명	비고
PAM_HPSEC	HP-UX와 관련해서 인증, 계정관리, 암호 관리 및 세션 관리 확장을 관리한다.	

	login, dtlogin, ftp, su, remsh, rexec, ssh 등의 서비스에는 /usr/lib/security/\$ISA/libpam_hpsec.so.1을 사용한다. 이러한 서비스는 하나 이상 비선택적 모듈 위의 스택 맨 위에 libpam_hpsec.so.1을 배치해야 한다. 또한 pam_hpsec 모듈은 /etc/default/security에 정의된 여러 속성을 강제로 시행한다.	
PAM_KRB5	일반 텍스트로 암호를 전송하지 않고 네트워크에서 보안 통신을 활성화하는 네트워크 인증 프로토콜이다. KDC(Key Distribution Center)는 암호를 인증한 다음 TGT(Ticket Granting Ticket)를 발급한다. PAM Kerberos 고스유 라이브러리는 /usr/lib/security/libpam_krb5.1이다.	
PAM_LDAP	LDAP(Lightweight Directory Access Protocol)는 디렉토리 서비스를 통해 사용자, 그룹 및 네트워크 관리 정보를 중앙 집중화하는 표준이다. 인증은 LDAP 디렉토리 서버에서 수행된다.	
PAM_NTLM	PAM NT LAN Manager를 사용하면 시스템 로그인 도중 Windows 서버에 대한 HP-UX 사용자를 인증할 수 있다. PAM NTLM은 NT 서버를 사용하여 HP-UX 시스템에 로그인하는 사용자를 인증한다.	
PAM_UNIX	인증, 계정관리, 세션관리, 암호 관리 등 네 가지 PAM 모듈에 대한 기능을 모두 제공한다. 모듈은 PAM UNIX 라이브러리 /usr/lib/security/libpam_unix.1을 통해 지원한다.	
PAM_UPDBE	PAM에 대한 사용자 정책 정의 서비스 모듈 /usr/lib/security/libpam_updbe.1은 사용자 구성 파일 /etc/pam_user.conf에 정의된 옵션을 읽고 후속 서비스 모듈에서 사용할 수 있도록 이 정보를 PAM 핸들에 저장한다.	

3.6 PAM 구성 파일

1) /etc/pam.conf를 사용한 시스템 범위 구성

PAM 구성 파일 /etc/pam.conf는 사용자를 인증 하는데 사용되는 보안 메커니즘을 정의한다. 기본값을 사용하면 표준 HP-UX 및 트러스트된 시스템 모두에서 일반적인 시스템 작업을 수행할 수 있다. 또한 개별 사용자 제어 및 DEC 통합 로그인 기능이 지원된다.

사용자가 로그인하거나 암호를 변경할 수 있으려면 libpam 및 libpam_unix PAM 라이브러리와 /etc/pam.conf 구성 파일이 시스템에 있어야 한다.

HP-UX 인증은 /etc/pam.conf 파일에 따라 달라진다. 이 파일의 소유자는 root여야 하고 파일 사용 권한은 다음과 같아야 한다.

```
-r--r--r-- 1 root sys 1050 nov 8 10:16 /etc/pam.conf
```

이 파일이 손상되거나 시스템에서 이 파일을 찾을 수 없소는 경우 root가 단일 사용자 모드로 콘솔에 로그인하여 문제를 해결할 수 있다.

보호되는 서비스 이름은 시스템 제어 파일 /etc/pam.conf에서 네가지 테스트 범주(module-type)인 인증, 계정, 세션 및 암호 아래에 나열된다.

/etc/pam.conf의 항목은 다음 구문을 사용한다.

```
[service-name] [module-type] [control-flag] [module-path] [module-options]
```

① service_name

서비스의 이름(대소문자 구분 안함)다(예:login 또는 ssh). 응용 프로그램에서는 응용 프로그램이 제공하는 서비스에 대해 서로 다른 서비스 이름을 사용할 수 있다. 예를 들어 sshd 데몬이 제공하는 다양한 서비스의 서비스 이름을 확인하려면 sshd(1M) 매뉴얼 페이지에서 PAM을 검색하라.

미리 정의된 서비스 이름 "other"는 특정 서비스 구성이 제공되지 않은 경우 사용되는 기본 서비스 이름이다.

② module-type

PAM이 어떤 타입의 인증이 사용될 것인지 서비스의 유형(즉, auth, account, session 또는 password)을 나타낸다.

모듈 타입	설명	비고
auth	사용자 인증에 사용하며, 올바른 비밀번호인지 확인하는 절차를 가진다. 응용 프로그램이 사용자에게 비밀번호를 입력하도록 안내하고, 사용자와 해당 사용자의 그룹에 대한 권한을 인증한다.	
account	사용자의 접근 허가 여부를 확인하며 계정 만료, 특정 시간대에 접근이 허용되었는지 여부를 확인한다. 인증하는 기능이 아니며 현재 시간, 사용자의 위치와 같은 다른 요소들의 접근 권한을 결정하는데, 예로 root 사용자의 로그인은 콘솔로만 한다. 이런 식으로 결정하는 것이다.	
password	비밀번호를 설정하고 확인한다. 비밀번호를 기준에 맞게 변경하도록 하는 모듈을 지정한다.	
session	사용자가 인증 받기 전후에 필요한 홈 디렉토리 마운트, 메일박스 생성 등의 유저 섹션을 구분하기 위해 추가적인 작업을 수행한다. 혹시라도 사용자의 로그인 전후에 수행해야 할 작업이 있다는 내용을 지정한다.	

③ contol-flag

PAM에서 사용되는 모듈들이 결과에 따라 어떠한 동작을 취해야 하는지를 지시하는 서비스에 대한 성공 또는 실패 값을 결정하는 모듈의 역할을 나타낸다.

flag	설명	비고
required	인증이 거부되기 전에 PAM이 이 서비스에 등록된 모든 모듈들을 요구함에도 불구하고 실패할 경우 인증을 거부하도록 한다. 해당 모듈은 개별 사용자에게 대한 인증을 반드시 진행해야 한다. 인증이 안될 경우 실패가 반드시 반환되어야 한다.	
requisite	이 모듈을 이용하는 인증이 실패할 경우, 즉시 인증을 거부하도록 한다. required와 비슷하지만 이 플래그가 인증을 실패할 경우 설정 파일에서 이 값 다음으로 나오는 모듈들을 호출되지 않는다. 실패한 결과는 즉시 응용 프로그램으로 전달된다.	
sufficient	이전에 요청되었던 모듈이 실패하더라도 이 모듈에 의해서 인증이 성공할 경우 PAM은 인증을 승인한다. 모듈이 성공 값을 반환하고 설정 파일에서 required와 sufficient 제어 플래그가 필요하지 않다면 PAM은 해당 응용 프로그램에서 성공을 반환한다.	
optional	이 모듈이 성공 또는 실패하는지는 그 모듈이 서비스에 대한 형식에 유일한 모듈일 경우에 해당한다. (성공 여부 상관 X) 이 제어 플래그는 모듈의 결과를 무시한다. PAM으로 다른 모듈을 지속적으로 확인하도록 한다. 확인 작업이 실패하게 되더라도 계속 진행하게 된다. 특정 모듈이 실패하더라도 사용자가 로그인하는 것을 허용하고자 할 때 이 플래그를 사용하면 된다.	
include	이 플래그는 인자(argument)에 지정된 또 다른 설정 파일의 내용이나 지침을 포함시	

	키기 위해 사용된다. 즉, 서로 다른 PAM 설정 파일의 내용을 연결하고 구성하는 방식으로 사용된다.	
substack	이 플래그는 다른 PAM 관련 모듈을 불러오는 것은 include와 동일 하지만, substack 은 substack의 동작 결과에 따라 나머지 모듈을 처리하지 않는다	

required/requisite 차이는 required와 requisite 인터페이스 모두 반드시 "성공" 되어야만 PAM을 이용한 인증이 완료되나 보안 및 가용성 측면에서는 분명한 차이가 존재한다. required의 경우 실패를 해도 실패한 지점이나 결과값에 대한 리턴을 하지 않는데, requisite의 경우 실패한 지점과 원인을 리턴하기에 보안 관점에서는 공격자에게 인증이 거부된 원인을 제공하게 되며, 가용성 측면에서는 실패한 원인을 리턴하여 원인 분석을 용이하게 해준다.

④ module-path

PAM에게 어떤 모듈을 사용할 것인지 그리고 그것을 어디서 찾을지의 모듈 유형을 구현하는 모듈의 경로다. 경로 이름이 절대 경로가 아닌 경우 경로 이름은 /usr/lib/security/\$ISA/ 경로에 대한 상대 경로로 간주 된다. \$ISA 매크로 또는 토큰은 PAM 프레임워크가 모듈 경로의 아키텍처 특정 디렉토리를 검색하도록 지시한다.

⑤ module-option

서비스 모듈로 전달될 수 있는 nowarn 및 debug와 같은 옵션이다. 모듈의 매뉴얼 페이지에서는 해당 모듈에 대한 옵션을 설명한다. 각각의 모듈들은 각각의 인수를 가지고 있다.

arguments	설명	비고
debug	시스템 로그에 디버깅 정보를 남기다.	
no_warn	응용 프로그램에 경고 메시지를 제공하지 않는다.	
use_first_pass	비밀번호를 두 번 확인하지 않는다. 대신 auth 모듈에서 입력한 비밀번호를 사용자 인증 과정 시에도 재사용 해야 한다. (이 옵션은 auth 및 password 모듈에 해당하는 옵션임)	
try_first_pass	이 옵션은 use_first_pass 옵션과 비슷하데, 사용자는 두 번 비밀번호를 입력할 필요가 없기 때문이다. 하지만 기존의 비밀번호를 다시 입력하도록 되어 있다.	
use_mapped_pass	이 인자는 이전 모듈에서 입력된 텍스트 인증 토큰을 입력 받도록 하는데, 이 값으로 암호화 또는 암호화가 해제된 키 값을 생성한다. 그 이유는 모듈에 대한 인증 토큰 값을 안전하게 저장하거나 불러오기 위함이다.	
expose_account	이 값은 모듈로 하여금 계정 정보를 중요하다고 판단하지 않게 한다. 시스템 관리자에 의해 임의로 설정한 것이라 여겨진다.	
nullok	이 인자는 호출된 PAM 모듈이 null 값의 비밀번호를 입력하는 것을 허용한다.	

다음은 샘플 /etc/pam.conf 파일의 일부 목록이다. #로 시작하는 줄은 주석 줄이다. /etc/pam.conf에는 인증관리, 계정관리, 세션 관리 및 암호 관리 섹션이 있다.

```
#
# Authentication management
#
login    auth required libpam_hpsec.so.1
login    auth required libpam_unix.so.1
su       auth required libpam_hpsec.so.1 bypass_setaud
su       auth required libpam_unix.so.1
dtlogin  auth required libpam_hpsec.so.1
dtlogin  auth required libpam_unix.so.1
dtaction auth required libpam_hpsec.so.1
```

```
dtaction auth required libpam_unix.so.1
ftp      auth required libpam_hpsec.so.1
ftp      auth required libpam_unix.so.1
rcomds  auth required libpam_hpsec.so.1
rcomds  auth required libpam_unix.so.1
sshd    auth required libpam_hpsec.so.1
sshd    auth required libpam_unix.so.1
OTHER   auth required libpam_hpsec.so.1
OTHER   auth required libpam_unix.so.1
```

2) /etc/pam_user.conf 사용자 구성 파일

PAM 구성 파일 /etc/pam_user.conf는 사용자별로 PAM을 구성한다. 이 파일은 선택 사항이며, PAM 응용 프로그램이 사용자에게 따라 다르게 동작해야 하는 경우에만 필요하다.

/etc/pam_user.conf에 나열하여 개별 사용자에게 다른 옵션을 할당한다. 여기에 나열된 login-name에 대해 지정된 options는 /etc/pam.conf의 module-type 및 module-path에 대해 지정된 모든 options를 대체한다.

/etc/pam_user.conf의 항목은 다음 구문을 사용한다.

```
[login-name] [module-type] [module-path] [module-options]
```

여기서 각 항목에 대한 설명은 다음과 같다.

항목	설명	비고
login-name	사용자의 로그인 이름.	
module-type	/etc/pam.conf에 지정된 module-type.	
module-path	/etc/pam.conf의 module-type과 연관된 module-path.	
Module-options	모듈에 의해 인식되는 0개 이상의 옵션.	

/etc/pam_user.conf의 기본 내용은 주석이다.

3.7 PAM이 로그인에 대해 작동하는 방식

다음 예제에서는 /etc/pam.conf 파일이 구성된 방식에 따라 login의 auth 프로세스에 대해 설명한다.

1) /etc/pam.conf에 다음과 같은 단일 표준 login auth가 포함되어 있으면 login이 정상적으로 진행된다.

```
login auth required /usr/lib/security/libpam_unix.1
```

2) 다음과 같은 시스템 범위 login auth 항목이 두 개 이상 있으면 순서대로 수행된다.

```
login auth required /usr/lib/security/libpam_unix.1
login auth required /usr/lib/security/libpam_dce.1
```

이 경우 표준 HP-UX login 프로세스가 실행된 다음 DCE 인증 프로세스가 발생한다. 둘 다 제대로 실행되면 로그인이 성공적으로 완료된다. 둘 중 하나가 실패해도 두 프로세스가 모두 수행된다.

3) 서로 다른 사용자에게 대해 서로 다른 인증 방법이 필요한 경우에는 /etc/pam.conf의 인증 모듈 앞에 특수한 항목인 libpam_ulpbe를 지정한다. (쉽게 참조할 수 있도록 줄에 번호가 매겨져 있음)

```

#/etc/pam.conf
#1
login auth required /usr/lib/security/libpam_udpbe.1
#2
login auth required /usr/lib/security/libpam_unix.1
#3
login auth required /usr/lib/security/libpam_dce.1

```

그런 다음 /etc/pam_user.conf에 영향을 받는 각 사용자에게 대한 항목을 지정한다.

```

#/etc/pam_user.conf
#4
allan auth /usr/lib/security/libpam_unix.1 debug
#5
allan auth /usr/lib/security/libpam_dce.1 try_first_pass
#6
isabel auth /usr/lib/security/libpam_unix.1 debug use_psd

```

allan이 로그인할 때 /etc/pam.conf의 첫 번째 줄로 인해 PAM이 /etc/pam_user.conf를 읽게 된다. /etc/pam_user.conf의 네 번째 및 다섯 번째 줄에 있는 모듈 경로가 /etc/pam.conf의 두 번째 및 세 번째 줄에 있는 모듈 경로와 일치하므로 PAM은 임시로 /etc/pam.conf의 두 번째 및 세 번째 줄에 있는 null options 필드를 각각 debug 및 try_first_pass로 대체한다. 그런 다음 두 번째 및 세 번째 줄에 지정된 모듈이 변경된 옵션으로 실행된다.

isabel이 로그인할 때 /etc/pam.conf의 첫 번째 줄로 인해 PAM이 /etc/pam_user.conf를 읽게 되고 /etc/pam.conf의 두 번째 줄에 있는 options 필드를 임시로 debug, use_psd로 대체하며 세 번째 줄은 변경되지 않는다. 그런 다음 두 번째 및 세 번째 줄에 지정된 모듈이 변경된 옵션으로 실행된다.

george가 로그인할 때 /etc/pam.conf의 첫 번째 줄로 인해 PAM이 /etc/pam_user.conf를 읽게 된다. george에 대한 항목이 없으므로 /etc/pam_user.conf의 두 번째 및 세 번째 줄은 변경되지 않는다. 두 번째 및 세 번째 줄에 지정된 모듈이 변경 없이 실행된다.

3.8 SSH(Secure Shell)

Secure Shell은 개방형 소스 SSH 제품인 OpenSSH 제품을 기반으로 한다. (<http://www.openssh.org>)

Secure Shell은 비보안 네트워크에서 클라이언트와 원격 호스트 간의 보안 연결을 가능하게 한다.

이 보안 연결의 주요 속성은 다음과 같다.

- 클라이언트와 원격 호스트 둘 다에 대한 강력한 인증
- 클라이언트와 원격 호스트 간의 통신에 대한 강력한 암호화 및 공개 키 암호화
- 클라이언트와 원격 호스트에서 명령을 실행하는데 사용할 보안 연결

Secure Shell은 telnet, remsh, rlogin, ftp, 및 rcp와 같은 자주 사용하는 함수와 명령의 보안 대체 항목을 제공한다.

1) Secure Shell의 주요 보안 기능

Secure Shell의 주요 보안 기능은 다음과 같다.

① 강력한 암호화

클라이언트와 원격 호스트 간의 모든 통신은 Blowfish, 3DES, AES 및 arcfour와 같은 특허 없는 암호화 알고리즘을 사용하여 암호화된다. 암호 등의 인증 정보는 네트워크에서 일반 텍스트로 전송되지 않는다. 또한 암호화는 강력한 공개 키 기반 암호화와 더불어 잠재적 보안 공격을 차단한다.

② 강력한 인증

Secure shell은 클라이언트와 서버 간의 강력한 인증 방법 집합을 지원한다. 인증은 양방향일 수 있다. 서버는 클라이언트를 인증하고 클라이언트도 서버를 인증한다. 이렇게 하면 다양한 보안 문제로부터 세션을 보호할 수 있다.

③ 포트 전달

클라이언트와 원격 호스트 간의 TCP/IP 연결 리디렉션(및 그 반대)을 포트 전달 또는 SSH 터널링이라고 한다. Secure Shell은 포트 전달을 지원한다. 예를 들어, 포트 전달을 사용하여 클라이언트와 서버 간의 ftp 트래픽(또는 전자 메일 클라이언트와 POP/IMAP 서버 간의 전자 메일 트래픽)을 리디렉션할 수 있다. 클라이언트가 직접 서버와 통신하는 대신 보안 채널을 통해 트래픽을 sshd 서버로 리디렉션할 수 있으며, 그런 다음 sshd 서버에서 트래픽을 실제 서버 시스템의 지정된 포트로 전달할 수 있다.

2) Secure Shell의 소프트웨어 구성 요소

Secure Shell 소프트웨어는 클라이언트 및 서버 구성 요소로 이루어져 있다.

구성요소	설명	위치	대응하는 비보안 구성요소
ssh	Secure Shell을 클라이언트는 telnet 및 remsh의 보안 대체 항목이며 보안 기능이 있는 remsh와 가장 유사함.	클라이언트	remsh, telnet, rlogin
slogin	ssh에 대한 심볼릭 링크임.	클라이언트	remsh, telnet, rlogin
scp	클라이언트 보안 복사 및 서버 보안 복사를 수행함,	클라이언트 및 서버	rcp
sftp	보안 ftp 클라이언트임	클라이언트	ftp
sshd	보안 Shell 데몬임.	서버	remshd, telnetd, rlogind
sftp-server	보안 ftp 데몬임.	서버	ftpd
ssh-rand-helper	sshd가 서버에서 /dev/random 또는 /dev/urandom을 찾을 수 없을 때는 사용되는 Random Number Generator. OS 커널에 상주하는 Random Number Generator인 rng가 포함되어 있다. rng가 구성 해제되어 있으면 sshd는 prngd를 사용함.	서버	해당 사항 없음
ssh-agent	클라이언트에서 서버로의 "자동" 키 기반 로그인 도구임.	클라이언트 및 서버	Rhosts 파일 메커니즘
ssh-add	클라이언트의 키 쌍을 ssh-agent에 알리는 도구임.	클라이언트	해당 사항 없음
ssh-keygen	공개 키 인증을 위해 키 쌍을 생성하는 도구임.	클라이언트	해당 사항 없음
ssh-keyscan	Secure Shell 데몬(sshd)을 실행하는 호스트 집합에 대한 공개 키를 수집하는 클라이언트 도구임.	클라이언트	해당 사항 없음
ssh-keysign	호스트 기반 인증 중에 필요한 디지털 서명을 생성하는 도구임. ssh()에서 로컬 호스트 키 호스트 기반	클라이언트	해당 사항 없음

인증에 액세스하는데 사용됨.		
-----------------	--	--

3) Secure Shell 실행

위 표에서 나열된 Secure Shell 클라이언트를 실행하기 전에 먼저 Secure Shell 서버 데몬 sshd를 시작한다.

sshd 데몬은 서버 시스템의 /etc/ssh 디렉토리에 있는 sshd_config 파일에서 초기 값을 가져온다.

sshd_config에 있는 가장 중요한 구성 지시어 중 하나는 sshd 데몬에서 지원하는 인증 방법 집합이다.

- ssh 클라이언트 실행

ssh 클라이언트 응용 프로그램은 sshd 서버와의 소켓 연결을 설정한다.

sshd 서버는 자식 sshd 프로세스를 시작한다.

이 자식은 연결 소켓을 상속 받고 선택된 인증 방법을 기반으로 클라이언트를 인증한다.

인증에 성공한 경우에만 성공적으로 보안 클라이언트 세션이 설정된다.

세션이 만들어진 후 모든 후속 통신은 클라이언트와 이 자식 sshd 프로세스 간에 직접 이루어진다.

이제 클라이언트는 서버에서 원격 명령을 실행할 수 있다.

ssh 클라이언트의 명령 요청이 있을 때마다 자식 sshd 프로세스에서 해당 명령을 실행할 Shell 프로세스를 시작한다.

간단히 말해서 실행 중인 ssh 클라이언트-서버 세션은 다음 프로세스로 구성된다.

① sshd 서버에 연결된 모든 클라이언트 시스템에는 현재 이 클라이언트 시스템에서 설정된 각 ssh 연결에 대한 하나의 ssh 클라이언트 프로세스가 있다.

② 서버 시스템에는 하나의 부모 sshd 프로세스와 서버에 연결된 동시 ssh 클라이언트 개수 만큼의 자식 sshd 프로세스가 있다. 서버에 권한 분리가 활성화되어 있으면 서버에서 실행되는 자식 sshd 프로세스의 수가 두 배로 증가한다.

③ ssh 클라이언트에서 명령 실행 요청이 있을 때마다 서버 시스템의 해당 자식 sshd 프로세스는 Shell 프로세스를 시작하고 Unix 파이프를 사용하여 명령 요청을 이 Shell 프로세스로 전달한다. 이 Shell 프로세스는 Unix 파이프를 사용하여 명령 실행 결과를 자식 sshd 프로세스로 반환하고 명령 실행이 완료되면 종료된다.

- sftp 클라이언트 실행

sftp 클라이언트 응용 프로그램은 sftp 클라이언트 응용 프로그램이 ssh 클라이언트를 시작 하도록 하고 Unix 파이프를 사용하여 이 클라이언트와 통신한다. 그런 다음 ssh 클라이언트는 sshd 서버와의 소켓 연결을 설정한다.

서버 상호 작용의 나머지 부분은 ssh 클라이언트의 경우와 유사하다. 차이점은 원격 명령을 실행한 Shell 을 시작하는 대신 자식 sshd 프로세스가 sftp-server 프로세스를 시작한다는 것이다. 이 sftp 세션 중의 모든 후속 통신은 다음 프로세스 간에 발생한다.

① Unix 파이프를 사용하여 클라이언트 시스템에서 sftp 클라이언트와 ssh 클라이언트 간에

- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Unix 파이프를 사용하여 자식 sshd 프로세스와 sftp 서버 프로세스 간에

- scp 클라이언트 실행

scp 클라이언트의 경우는 sftp 클라이언트 실행과 거의 동일하다. 차이점은 sftp-server 프로세스를 시작하는 대신 자식 sshd 프로세스가 scp 프로세스를 시작한다는 것이다. scp 세션 중의 모든 후속 통신은 다음 프로세스간에 발생한다.

- ① 클라이언트 시스템에서 scp 클라이언트와 ssh 클라이언트 간에, Unix 파이프 사용
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Unix 파이프를 사용하여 자식 sshd 프로세스와 scp 서버 프로세스 간에

4) Secure Shell 권한 분리

Secure Shell은 권한 분리 기능을 통해 보다 향상된 수준의 보안을 제공한다. 부모 sshd 및 자식 sshd 프로세스는 권한이 부여된 사용자로 실행된다. 권한 분리를 활성화하면 사용자 연결당 하나의 추가 프로세스가 시작된다.

ssh 클라이언트가 권한 분리에 대해 구성된 sshd 서버에 연결하면 부모 sshd 프로세스가 권한이 부여된 자식 sshd 프로세스를 시작한다. 권한 분리를 활성화하면 자식 sshd 프로세스는 권한이 없는 자식 sshd 프로세스를 추가로 시작한다. 권한이 없는 자식 sshd 프로세스는 연결 소켓을 상속 받는다. 클라이언트와 서버 간의 모든 후속 통신은 권한이 없는 이 자식 sshd 프로세스를 사용하여 이루어진다.

클라이언트의 원격 명령 실행 요청은 대부분 비권한이며 권한이 없는 이 자식 sshd 프로세스에서 시작된 Shell에 의해 처리된다. 권한이 없는 자식 sshd 프로세스에서 권한이 부여된 기능을 실행해야 하는 경우 Unix 파이프를 사용하여 권한이 부여된 부모 sshd 프로세스와 통신한다.

권한 분리는 침입자에 의한 잠재적 손상을 제한하는데 도움이 된다. 예를 들어, Shell 명령을 실행하는 동안 버퍼 오버플로 공격이 발생할 경우 권한이 없는 프로세스 내에서 제어되므로 잠재적 보안 위험이 제한된다.

권한 분리는 Secure Shell의 기본 구성이다. sshd_config 파일에서 "UsePrivilegeSeparation NO"를 설정하여 권한 분리를 해제할 수 있다. 잠재적 권한 위험이 있으므로 권한 분리를 해제할 경우 신중하게 고려해야 한다.

5) Secure shell 인증

Secure Shell은 다음 인증 방법을 지원한다.

- GSS-API(Kerberos 기반 클라이언트 인증)
- 공개 키 인증
- 호스트 기반 인증
- 암호 인증

클라이언트는 원격 sshd 데몬과 연결될 때 원하는 인증 방법(앞에 나열된 방법 중 하나)을 선택하고 연결 요청의 일부로 적절한 자격 증명을 제공하거나 서버에서 보낸 프롬프트에 응답한다. 모든 인증 방법이 이런 방식으로 작동한다.

서버가 성공적으로 연결을 설정하려면 클라이언트로부터 적절한 키, 암호구, 암호 또는 자격 증명을 받아야 한다.

sshd 인스턴스에서 보안 요구 사항을 기반으로 지원되는 인증 방법 중 일부만 지원하도록 선택할 수 있다.

Secure Shell은 앞에 나열된 인증 방법을 지원하지만 시스템 관리자가 환경의 특정 보안 요구 사항을 기반으로 sshd 인스턴스에서 제공되는 인증 방법을 제한할 수 있다. 예를 들어, Secure Shell 환경에서 모든 클라이언트가 공개 키 또는 Kerberos 방법을 사용하여 인증해야 한다고 지정할 수 있으며, 이로 인해 나머지 방법은 비활성화될 수 있다. 지원되는 인증 방법 활성화 및 비활성화는 sshd_config 파일에 지정된 구성 지시어를 통해 이루어진다.

ssh 클라이언트 연결 요청이 있으면 서버는 먼저 지원되는 인증 방법 목록으로 응답한다. 이 목록은 sshd 서버에서 지원하는 인증 방법과 이러한 방법이 시도되는 시퀀스를 나타낸다. 클라이언트는 이러한 인증 방법을 하나 이상 생략할 수 있다. 클라이언트에서 방법이 시도되는 시퀀스를 변경할 수도 있다. 이렇게 하려면 클라이언트 구성 파일 /etc/ssh/ssh_config의 구성 지시어를 사용한다.

Secure Shell에서 지원하는 인증 방법은 다음과 같이 요약되어 있다.

① GSS-API(Kerberos 기반 클라이언트 인증)

Kerberos 기반 클라이언트 인증인 GSS-API(Generic Security Sservice application Programming Interface)를 사용하는 경우 클라이언트가 미리 Kerberos 자격 증명을 받아야 하며, 해당 클라이언트 디렉토리에 Kerberos 구성 파일이 있어야 한다.

클라이언트는 sshd 데몬과 연결될 때 연결 시 자격 증명을 제공한다.

서버는 이러한 자격 증명을 이 특정 사용자의 자격 증명 복사본과 일치 시킨다.

또한 선택적으로 클라이언트 호스트 환경의 타당성을 설정할 수 있다.

② 공개 키 인증

공개 키 인증을 사용하려면 Secure Shell 환경에 다음 설정이 있어야 한다..

클라이언트와 서버 둘 다에 키 쌍이 있어야 한다. 모든 ssh 클라이언트와 모든 sshd 서버가 ssh-keygen 유틸리티를 사용하여 자체적으로 키 쌍을 생성해야 한다.

클라이언트는 통신해야 하는 모든 sshd 서버에 해당 공개 키를 알려야 한다. 이렇게 하려면 각 클라이언트의 공개 키를 모든 관련 서버의 미리 지정된 디렉토리에 복사한다.

클라이언트는 통신해야 하는 모든 서버의 공개 키를 구해야 한다. 클라이언트는 ssh-keyscan 유틸리티를 사용하여 공개 키를 받는다.

이 설정이 완료 되면 sshd 서버에 연결하는 ssh 클라이언트가 공개 키와 개인 키를 사용하여 인증된다.

Secure Shell은 공개 키 인증을 단순화하는 추가 기능을 제공한다. 일부 환경에서는 항상 암호 프롬프트에 응답할 필요가 없도록 설정할 수 있다. 둘 다 클라이언트 시스템에서 실행되는 ssh-agent 및 ssh-add 프로세스를 조합해서 사용하면 암호에 응답하지 않아도 된다. 클라이언트는 ssh-add 유틸리티를 통해 ssh-agent 프로세스에 모든 키 정보를 등록한다. 그런 다음 클라이언트와 서버 간의 공개 키 인증은 sshd 데몬이 클라이언트와 상호 작용할 필요 없이 ssh-agent에 의해 수행된다.

③ 호스트 기반 및 공개 키 인증

호스트 기반 및 공개 키 인증은 보다 안전한 공개 키 인증 방법의 확장이다.

클라이언트와 서버 둘 다의 키 쌍이 있는 것 외에도 이 방법을 사용하면 클라이언트 환경에서 통신할 서버를 제한할 수 있다.

클라이언트의 홈 디렉토리에 `.rhosts` 파일을 만들어 이 제한을 구현한다.

④ 암호 인증

암호 인증 방법은 단일 사용자-ID 및 암호 기반 로그인을 사용한다. 이 로그인은 `/etc/passwd`에 지정된 사용자 로그인을 기반으로 하거나 PAM 기반일 수 있다.

Secure Shell은 서버 시스템에서 사용할 수 있는 PAM 모듈과 완전히 통합된다. 이 목적을 위해 `/etc/ssh/sshd_config` 파일에 UsePAM 구성 지시어가 있다. YES로 설정하면 클라이언트에서 암호 인증 요청이 있을 때마다 sshd는 PAM 구성 파일(`/etc/pam.conf`)을 확인한다. 그런 다음 구성된 PAM 모듈을 통해 암호 인증이 성공할 때까지 순서대로 수행한다.

PAM 인증을 무시하려면 UsePAM 지시어를 NO로 설정한다. 그러면 클라이언트에서 암호 인증 요청이 있을 때 sshd가 서버의 PAM 구성 설정을 무시한다. 대신 sshd는 `gwtppnam()` 라이브러리 호출을 직접 호출하여 사용자 암호 정보를 가져온다.

Secure Shell은 PAM_UNIX, PAM_LDAP 및 PAM_KERBEROS를 사용하여 테스트 되었다. 또한 PAM_DCE 및 PAM_NTLM과 같은 다른 PAM 모듈에서 작동한다.

6) 통신 프로토콜

Secure Shell 사용자는 SSH-1 또는 SSH-2 프로토콜을 사용하여 원격 sshd 데몬과 연결할 수 있다. SSH-2가 더 안전하므로 SSH-1 대신 권장한다.

7) Secure Shell에서 사용하는 기능의 일부 목록

Secure Shell은 실제로 Shell이 아니라 호스트에서 인전하게 원격 shell 세션을 실행하기 위해 클라이언트와 원격 호스트 간에 보안 연결을 만드는 매커니즘이다. 보안 연결을 설정하기 위해 Secure Shell에서 인증과 세션 생성을 대부분 수행한다. Secure Shell에서 사용하는 기능의 일부 목록은 다음과 같다.

① login 시도 기록

telnet 또는 remsh와 마찬가지로 Secure Shell은 성공한 세션과 실패한 세션을 각각 `/var/adm/wtmp` 및 `/var/adm/btmp` 파일에 기록한다.

② PAM 모듈

Secure Shell은 클라이언트 세션에 대해 PAM 인증을 사용할 수 있다. PAM 인증을 선택하면 Secure Shell은 `/etc/pam.conf` 파일을 사용하고 인증을 위해 해당 PAM 모듈을 호출한다.

③ `/etc/default/security` 파일 사용

로그인 동작 암호 및 기타 보안 구성을 정의하는 속성이 들어 있는 시스템 범위 구성 파일이다. 몇 가지 제한은 있지만 Secure Shell에서 이러한 속성을 사용할 수 있다.

④ Shadow passwd

Secure Shell은 Shadow passwd 기능과 통합된다.

⑤ 컨트롤 시스템 로그(syslog)

Secure Shell은 syslog를 사용하여 중요한 메시지를 남긴다.

⑥ 감사 로깅

Secure shell은 해당 코드로 감사 로깅(트러스트된 모드)을 구현했다.

8) 비밀번호 없이 SSH에 접속하는 방법

SSH 접속 시 비밀번호 없이 접속하는 방법은 다음과 같다.

① ssh key 생성 (없는 경우만)

```
$ ssh-keygen -t rsa
```

② 서버로 로컬에서 만든 공개키 복사

```
$ scp ~/.ssh/id_rsa.pub [id@[address]:id_rsa.pub
```

③ 서버 접속 (일반접속)

```
$ ssh [id@[address]
```

④ 로그인 후 개인폴더에 .ssh 폴더 생성 (있으면 pass)

```
$ mkdir .ssh
```

⑤ .ssh폴더 권한 변경

```
$ chmod 700 .ssh
```

⑥ 공개키 인증키 목록에 추가

```
$ cat id_rsa.pub >> .ssh/authorized_keys
```

⑦ 필요없는 공개키 제거

```
$ rm -f id_rsa.pub
```

⑧ 인증키 목록 권한 수정

```
$ chmod 644 .ssh/authorized_keys
```

⑨ 서버 접속

```
$ ssh -i ~/.ssh/id_rsa [id@[address]
```

3.9 PAM 프로그래밍

아래 소스 코드는 PAM 모듈의 기본 형태로 "#if 0"으로 막혀 있는 부분에 통제를 할 수 있는 코드를 넣어 두면 원하는 동작(차단 또는 허용, 로그 기록)을 쉽게 처리할 수 있다.

```
/* pam_test.c */
```

```

#include <stdio.h>
#include <stdlib.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>

#ifdef PAM_MODULE_ENTRY
PAM_MODULE_ENTRY("pam_test");
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN extern
#endif

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

/* expected hook, this is where custom stuff happens */
PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    #if 0
        // 아래 조건을 만족하지 못할 경우 로그인 실패로 처리함
        if (check_auth_pam(pamh) < 0)
            return PAM_PERM_DENIED;
    #endif
    return PAM_SUCCESS;
}

```

위에 있는 pam_sm...으로 된 함수들이 사용자가 임의로 추가한 함수가 아니라 PAM에서 기본 함수들로서 해당 함수 부분에 원하는 동작 코드를 넣도록 되어 있다.

PAM 모듈 자체가 기존에 동작하는 telnet이나, ssh, login 등의 바이너리에 동적 라이브러리 형태로 붙기 때문에 위 코드에 삽입한 코드들은 로그인과 동시에 해당 서버로의 접속자의 모든 상황들을 감시하고 제어할 수 있다.

3.10 PAM 컴파일 방법

PAM 인증 모듈은 HP-UX의 특성별 OS에서 제공하는 최적화 옵션에 따라 컴파일 하는 방법은 다음과 같다.

```
> sudo cc +DD64 -Wl,+s +z -o baro_auth baro_auth.c -L/opt/openssl/lib/hpux64
> sudo cc +DD64 -Wl,+s +z -c base64.c -L/opt/openssl/lib/hpux64 -lssl -lcrypto -
L/opt/openssl/include
> sudo cc +DD64 -Wl,+s +z -c xxtea.c -L/opt/openssl/lib/hpux64 -lssl -lcrypto -
L/opt/openssl/include
> sudo cc +DD64 -Wl,+s +z -b -o pam_baro_auth.so pam_baro_auth.c xxtea.c base64.c -
L/opt/openssl/lib/hpux64 -lssl -lcrypto -lpam -lm -lc -ldl -L/opt/openssl/include
```

PAM 프로그램 컴파일할 때 반드시 sudo 명령어를 사용하여 컴파일 해야 한다. 그렇지 않으면 다음과 같은 오류가 발생한다.

```
Oct 25 17:26:34 project sshd[11609]: open_module: module /usr/baropam/pam_baro_auth.so writable
by group
Oct 25 17:26:34 project sshd[11609]: load_modules: can not open module
/usr/baropam/pam_baro_auth.so
```

sudo 명령어는 유닉스 및 유닉스 계열 운영 체제에서 다른 사용자의 보안권한과 관련된 프로그램을 구동할 수 있게 해주는 프로그램이다.

이것은 substitute user do (다른 사용자의 권한으로 명령을 이행하라, 는 뜻이다.) 의 줄임말이다.

기본적으로 Sudo는 사용자 비밀번호를 요구하지만 루트 비밀번호(root password)가 필요할 수 도 있고, 한 터미널에 한번만 입력하고 그 다음부터는 비밀번호가 필요 없다.

Sudo는 각 명령줄에 사용할 수 있으며 일부 상황에서는 관리자 권한을 위한 슈퍼유저 로그인(superuser login)을 완벽히 대신하며, 주로 우분투, 리눅스와 애플의 OS X 에서 볼 수 있다.

참고로 PAM 프로그램 컴파일할 때 필요한 기본적인 환경변수인 PATH(명령어 입력시 경로를 입력하지 않고 실행가능), SHLIB_PATH(외부 라이브러리를 링크할 때 참조할 경로)를 다음과 같이 설정한다.

```
export
PATH=$HOME/bin:/usr/bin:/usr/ccs/bin:/usr/local/bin:/usr/ucb:/usr/sfw/bin:$HOME/shell:/usr/sbin:.$PATH
export SHLIB_PATH=/usr/lib:/usr/ccs/lib:/lib:/usr/ucb:/usr/local/ssl/lib:/usr/sfw/lib:/usr/dt/lib:
/usr/openwin/lib:.$SHLIB_PATH
```

PAM을 컴파일하기 위해서는 암호화 모듈에 "openssl" 라이브러리를 사용하므로 반드시 "openssl"을 다음과 같은 명령어로 설치해야 한다.

```
> swinstall -s /tmp/OpenSSL*.depot openssl
```

3.11 PAM 디버그 사용

PAM (Pluggable Authentication Modules) 라이브러리는 실행 중에 디버그 정보를 제공 할 수 있다.

시스템이 디버그 출력을 수집하도록 설정한 후에는 수집된 정보를 사용하여 PAM API 호출을 추적하고 현재 PAM 설정에서 오류 지점을 확인할 수 있다.

PAM 디버그 출력을 사용하려면 다음 단계를 완료 해야 한다.

step 1) /etc/syslog.conf 파일을 편집하여 원하는 우선 순위 수준에서 syslog 메시지를 로깅 할 파일을 식별한다.

예를 들어 PAM 디버그 수준 메시지를 /var/log/auth.log 파일에 보내려면 다음 텍스트를 syslog.conf 파일의 새 행으로 추가해야 한다.

```
*.debug /var/log/auth.log
```

step 2) touch 명령을 사용하여 1 단계에서 참조한 출력 파일 (/var/log/auth.log)이 존재하지 않으면 작성해야 한다.

step 2) 구성 변경 사항이 인식되도록 syslogd 데몬을 다시 시작하려면 다음 단계를 완료 해야 한다.

```
> /sbin/init.d/syslogd start | stop
```

PAM 응용 프로그램이 다시 시작되면 /etc/syslog.conf 구성 파일에 정의된 출력 파일에 디버그 메시지가 수집된다.

3.12 PAM 테스트 방법

PAM 프로그램 컴파일 또는 PAM 구성 환경 변경 후 sshd를 restart하지 않고 테스트 할 수 있는 방법은 다음과 같이 진행 할 수 있다.

1) sshd 데몬 기동하기

sshd 데몬(22000 포트)을 디버깅 모드로 띄우기 위하여 다음과 같은 명령어를 수행한다.

```
> /usr/lib/sshd -D -p22000 -d
```

2) ssh로 접속하기

sshd 데몬(22000 포트)에 ssh로 접속하기 위하여 다음과 같은 명령어를 수행한다.

```
> ssh -v -p22000 root@10.0.2.15
```

4. AIX

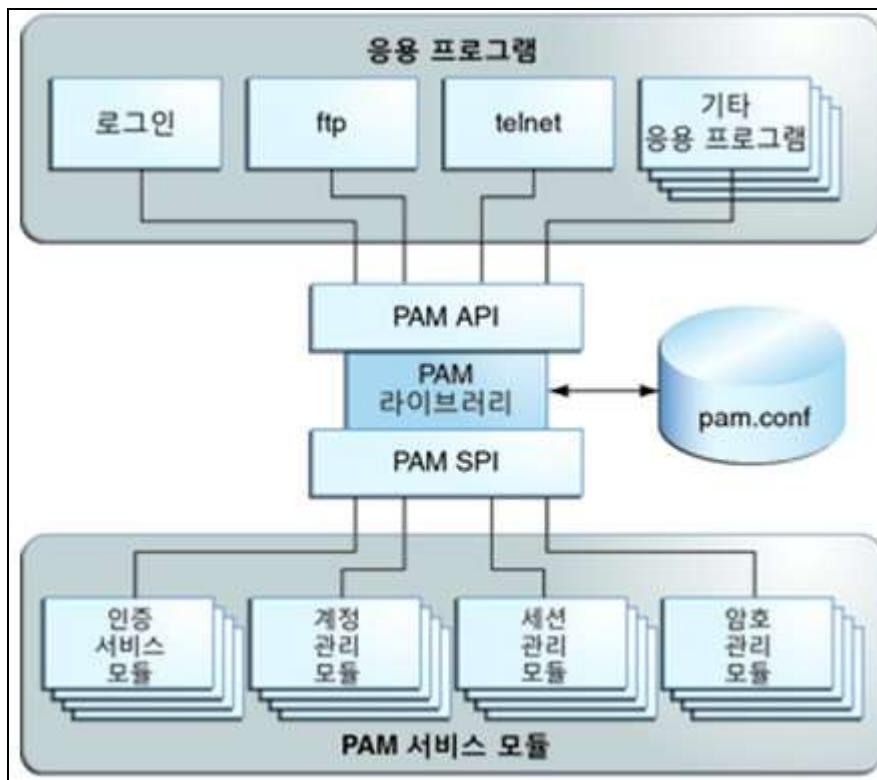
4.1 PAM 개요

PAM(Pluggable Authentication Module, 플러그 가능한 인증 모듈)은 Linux/Unix 시스템에서 서비스를 재 컴파일하지 않고, 다양한 인증 기술을 시스템 항목 서비스에 접목할 수 있도록 해주는 프레임워크로 중앙 집중적인 인증 매커니즘을 지원하는 것이다. 게다가 시스템의 기본적인 인증 기법을 제공하여 이것을 사용하면 응용 프로그램 개발자 뿐만 아니라 시스템 관리자들이 인증을 유연성 있게 관리할 수 있도록 도와 준다.

전통적으로 시스템 자원에 대한 접근을 관리하는 프로그램들은 내장된 매커니즘에 의해 사용자 인증 과정을 수행한다. 이러한 방식은 오랫동안 이루어졌지만 이러한 접근 방식은 확장성이 부족하고 매우 복잡하다. 그렇기 때문인지 이러한 인증 매커니즘을 끌어내기 위한 수많은 해킹 시도가 있었다.

Solaris의 방식을 따라서 Unix/Linux 사용자들은 그들만의 PAM을 구현하는 방식을 찾았다.

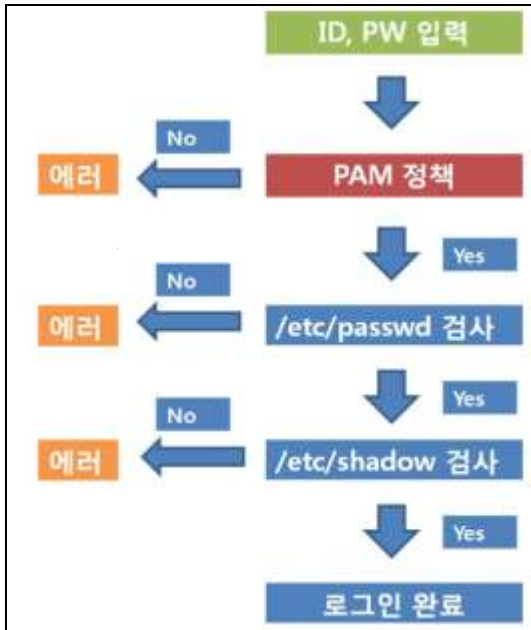
PAM의 아키텍처는 다음과 같다.



PAM의 기본 원리는 응용 프로그램이 password 파일을 읽어 오는 대신 PAM이 직접 인증을 수행하도록 하는 것이다. PAM은 시스템 관리자가 원하는 인증 매커니즘이 무엇이든 상관하지 않는다.

여러 사이트에서 선택받은 인증 매커니즘은 아직도 password 파일이다. 왜 그럴까? 우리가 원하는 것을 해주기 때문이다. 대부분의 사용자는 password 파일이 필요한 것이 무엇인지 이미 알고 있다. 그리고 원하는 작업을 수행하는데 있어 이미 그 기능이 검증되었기 때문일 것이다.

PAM의 인증 절차는 다음과 같다.



4.2 PAM 동작 원리

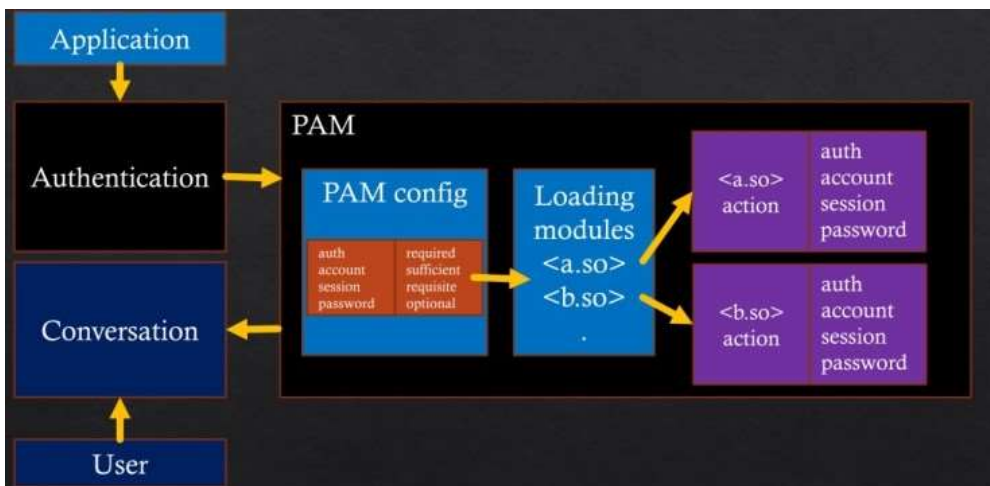
PAM은 Windows 환경의 DDL(Dynamic Link Library)과 같은 것이다. Library로 프로그램이 어떤 사용자에게 대한 인증을 수행하려면 PAM Library가 있는 함수를 호출한다. PAM은 해당 함수의 Library를 제공하여 응용 프로그램이 특정 사용자를 인증하도록 요청할 수 있다.

PAM을 통해 /etc/passwd 파일이나 /etc/shadow 파일을 확인하거나 또는 보다 복잡한 확인 작업을 수행하기도 하는데, 예로는 LDAP 서버에 접속하는 것이다.

확인 작업을 마치고, 인증 여부를 결정하게 되면 "인증됨/인증되지 않음"의 메시지를 자신을 호출한 응용 프로그램에 전송한다.

간단한 확인 작업이라고 했는데, 그 과정이 많아 보일 수가 있다. 여기에 나오는 각 모듈은 크기가 작고 작업 수행 시간이 매우 빠르다. 이것은 매우 놀랍기도 하지만 PAM을 사용하게 된다.

PAM의 동작은 프로그램의 실행하자마자 실행되는 보안 설정 파일들의 프로그램들이다.



PAM의 동작 원리는 다음과 같다.

- 인증이 필요한 프로그램 동작 시 PAM 라이브러리 호출
- PAM설정 파일을 참조
- 참조한 파일의 내용을 바탕으로 모듈 동작
- 프로그램에 동작 결과를 반환하여 인증여부 결정.

4.3 PAM 사용 이점

PAM을 사용하면 사용자 인증을 위한 시스템 항목 서비스(예: ftp, login, telnet, rsh) 사용을 구성할 수 있다. PAM이 제공하는 몇 가지 이점은 다음과 같다.

- 유연한 구성 정책
- 응용 프로그램별 인증 정책
- 기본 인증방식을 선택할 수 있는 기능
- 높은 보안 시스템에서 여러 권한 부여를 요구할 수 있는 기능
- 최종 사용자의 사용 편의성
- 암호가 여러 인증 서비스에 대해 동일한 경우 암호 재입력 없음
- 사용자가 여러 명령을 입력할 필요 없이 여러 인증 서비스에 대해 사용자에게 암호를 요구할 수 있는 기능
- 사용자 인증 서비스에 선택적 옵션을 전달할 수 있는 기능
- 시스템 항목 서비스를 변경할 필요 없이 사이트별 보안 정책을 구현할 수 있는 기능

4.4 PAM 파일과 위치

파일 위치	설명	비고
/usr/lib/security	제공 가능한 PAM 모듈 디렉토리로 실제 PAM Library를 동적으로 인증 모듈을 호출하여 실행한다. (*so)	
/etc/security/unix	PAM 실행에 필요한 설정파일로 /usr/lib/security에 있는 모듈에 대한 설정 파일. (서비스명.conf)	
/etc/pam.d	PAM 데몬 파일(애플리케이션별 PAM의 설정 파일 위치)로 PAM을 사용하는 응용 프로그램이 설정 파일이 없다면 기본값으로 설정된 설정 파일을 자동으로 사용한다.	

4.5 PAM 라이브러리

PAM 라이브러리 /usr/lib/libpam.a에는 모든 PAM 응용 프로그램에 대한 공통 인터페이스 역할을 하며 모듈 로드를 제어하는 PAM API가 포함되어 있다.

모듈은 /etc/pam.conf 파일에 정의된 스택킹 동작을 기반으로 PAM 라이브러리에 의해 로드 된다.

다음 PAM API 함수는 PAM 모듈이 제공하는 해당 PAM SPI를 호출한다.

예를 들어, pam_authenticate API는 PAM 모듈에서 pam_sm_authenticate SPI를 호출한다.

라이브러리	설명	비고
pam_authenticate	PAM 프레임워크 내에서 인증 수행	
pam_setcred	인증 서비스에 대한 사용자 자격 증명 수정 또는 삭제	
pam_acct_mgmt	PAM 계정 유효성 검사 절차 수행	
pam_open_session	PAM 세션 생성 작업 수행	
pam_close_session	PAM 세션 종료 작업 수행	
pam_chauthtok	PAM 프레임 워크 내에서 암호 관련 기능 수행	

또한 PAM 라이브러리에는 응용 프로그램이 PAM 모듈을 호출하고 PAM 모듈에 정보를 전달할 수 있게 해주는 여러 프레임 워크 API가 포함되어 있다.

다음 표는 AIX 및 그 기능에서 구현된 PAM 프레임워크 API를 보여준다.

라이브러리	설명	비고
pam_start	PAM 세션 설정	
pam_end	PAM 세션 종료	
pam_get_data	모듈 특정 데이터 검색	
pam_set_data	모듈별 데이터 설정	
pam_getenv	정의된 모든 PAM 환경 변수와 그 값 목록을 검색	
pam_putenv	PAM 환경 변수를 설정	
pam_get_item	일반적인 PAM 정보를 가져옴	
pam_set_item	공통 PAM 정보 설정	
pam_get_user	사용자 이름 검색	
pam_strerror	PAM 표준 오류 메시지 가져오기	

4.6 PAM 모듈

AIX 메커니즘은 getty, login, rlogin, rsh, telnet 및 tsm과 같은 명령을 통해 로그인을 처리한다.

AIX 메커니즘은 pam_aix 모듈을 사용하여 STD_AUTH 인증과 PAM_AUTH 인증을 지원한다.

usw 스탠자의 mkhomeatlogin 속성을 true로 설정하여 /etc/security/login.cfg 파일에서 AIX 메커니즘을 사용 가능하게 한다(파일에 대한 추가 정보는 /etc/security/login.cfg 파일을 참조하라).

chsec 명령을 사용하여 로그인시 자동 홈 디렉토리 생성 기능을 활성화 또는 비활성화 한다.

예를 들어 이 기능을 사용하려면 다음 명령을 실행한다.

```
> chsec -f /etc/security/login.cfg -s usw -a mkhomeatlogin=true
```

또는

```
usw:
shells = /bin/sh,/bin/bsh,/bin/csh,/bin/ksh,/bin/tsh,/bin/ksh93
maxlogins = 32767
logintimeout = 60
maxroles = 8
```

```
auth_type = PAM_AUTH
pwd_algorithm = sha256
```

활성화되면 로그인 프로세스는 인증이 성공한 후 사용자의 홈 디렉토리를 확인하고 사용자의 홈 디렉토리가 없으면 하나가 만들어진다.

Note : mkhomeatlogin 속성은 AIX 버전 6.1 (6100-02 Technology Level 이상)에서만 지원된다.

AIX는 또한 PAM 메커니즘을 위한 홈 디렉토리를 생성하기 위한 pam_mkuserhome 모듈을 제공한다.

pam_mkuserhome 모듈은 로그인 서비스를 위한 다른 세션 모듈과 쌓일 수 있다.

이 PAM 모듈을 서비스에 사용하려면 해당 항목에 항목을 추가해야 한다.

예를 들어, PAM을 사용하여 telnet 명령을 통해 홈 디렉토리를 만들려면 /etc/pam.cfg 파일에 다음 항목을 추가한다.

```
telnet session optional pam_mkuserhome
```

PAM 모듈을 사용하면 여러 인증 메커니즘을 시스템에서 집합적으로 또는 독립적으로 사용할 수 있다.

주어진 PAM 모듈은 네 가지 모듈 유형 중 적어도 하나를 구현해야 한다.

모듈 유형은 모듈 유형을 준수하는데 필요한 해당 PAM SPI와 함께 다음과 같이 설명된다.

1) 인증 모듈

사용자를 인증하고 자격 증명을 설정, 새로 고치거나 삭제한다. 이 모듈은 인증 및 자격 증명을 기반으로 사용자를 식별한다.

인증 모듈 기능 :

- pam_sm_authenticate
- pam_sm_setcred

2) 계정 관리 모듈

인증 모듈에서 식별한 후 사용자 계정의 유효성과 후속 액세스를 결정한다. 이러한 모듈에서 수행하는 검사에는 일반적으로 계정 만료 및 암호 제한이 포함된다.

계정 관리 모듈 기능 :

- pam_sm_acct_mgmt

3) 세션 관리 모듈

사용자 세션을 시작하고 종료한다. 또한 세션 감사 지원이 제공될 수도 있다.

세션 관리 모듈 기능 :

- pam_sm_open_session
- pam_sm_close_session

4) 암호 관리 모듈

암호 변경 및 관련 속성 관리를 수행한다.

암호 관리 모듈 기능 :

- pam_sm_chauthtok

4.7 PAM 구성 파일

/etc/pam.conf 구성 파일은 각 PAM 모듈 유형에 대한 서비스 항목으로 구성되며 정의된 모듈 경로를 통해 서비스를 라우팅한다.

파일의 항목은 공백으로 구분된 다음 필드로 구성된다.

```
[service_name] [module_type] [control_flag] [module_path] [module_options]
```

이 필드에 대한 설명은 다음과 같다.

1) service_name

서비스의 이름을 지정한다. 키워드 OTHER는 항목에 지정되지 않은 응용 프로그램에 사용할 기본 모듈을 정의하는데 사용된다.

2) module-type

서비스의 모듈 유형을 지정한다. 유효한 모듈 유형은 auth, account, session 또는 password이다. 주어진 모듈은 하나 이상의 모듈 유형을 지원한다.

모듈 타입	설명	비고
auth	사용자 인증에 사용하며, 올바른 비밀번호인지 확인하는 절차를 가진다. 응용 프로그램이 사용자에게 비밀번호를 입력하도록 안내하고, 사용자와 해당 사용자의 그룹에 대한 권한을 인증한다.	
account	사용자의 접근 허가 여부를 확인하며 계정 만료, 특정 시간대에 접근이 허용되었는지 여부를 확인한다. 인증하는 기능이 아니며 현재 시간, 사용자의 위치와 같은 다른 요소들의 접근 권한을 결정하는데, 예로 root 사용자의 로그인은 콘솔로만 한다. 이런 식으로 결정하는 것이다.	
password	비밀번호를 설정하고 확인한다. 비밀번호를 기준에 맞게 변경하도록 하는 모듈을 지정한다.	
session	사용자가 인증 받기 전후에 필요한 홈 디렉토리 마운트, 메일박스 생성 등의 유저 섹션을 구분하기 위해 부가적인 작업을 수행한다. 혹시라도 사용자의 로그인 전후에 수행해야 할 작업이 있다는 내용을 지정한다.	

3) control-flag

모듈의 스택킹 동작을 지정한다. 지원되는 제어 플래그는 required, requisite, sufficient 또는 optional이다.

control_flag 필드에 유효한 값과 스택의 해당 동작은 다음과 같습니다.

flag	설명	비고
required	스택의 모든 필수 모듈이 성공해야 성공한다. 하나 이상의 필수 모듈이 실패하면 스택의 모든 필수 모듈이 시도되지만 첫 번째 실패한 필수 모듈의 오류가 반환된다.	

requisite	필요한 모듈이 실패한 경우 스택의 다른 모듈이 처리되지 않고 필요한 모듈에서 첫 번째 오류 코드를 즉시 반환한다는 점을 제외하고는 요구 사항과 유사하다.	
sufficient	충분하다고 플래그가 지정된 모듈이 성공하고 이전의 필수 또는 충분한 모듈이 실패하지 않은 경우 스택의 나머지 모든 모듈이 무시되고 성공이 반환된다.	
optional	스택에 있는 모듈 충분하다고 플래그가 지정된 모듈이 성공하고 이전의 필수 또는 충분한 모듈이 실패하지 않은 경우 스택의 나머지 모든 모듈이 무시되고 성공이 반환된다. 하나도 없으며 충분한 모듈이 성공하지 못하면 서비스에 대한 하나 이상의 선택적 모듈이 성공해야 한다. 스택의 다른 모듈이 성공하면 선택적 모듈의 오류가 무시된다.	

required/requisite 차이는 required와 requisite 인터페이스 모두 반드시 "성공" 되어야만 PAM을 이용한 인증이 완료되나 보안 및 가용성 측면에서는 분명한 차이가 존재한다. required의 경우 실패를 해도 실패한 지점이나 결과값에 대한 리턴을 하지 않는데, requisite의 경우 실패한 지점과 원인을 리턴하기에 보안 관점에서는 공격자에게 인증이 거부된 원인을 제공하게 되며, 가용성 측면에서는 실패한 원인을 리턴하여 원인 분석을 용이하게 해준다.

4) module-path

서비스에 로드 할 모듈을 지정한다. module_path의 유효한 값은 모듈의 전체 경로 또는 모듈 이름으로 지정할 수 있다.

모듈에 대한 전체 경로가 지정되면 PAM 라이브러리는 해당 module_path를 사용하여 32bit 서비스를 로드하거나 64bit 서비스 용으로 64 서브 디렉토리를 사용한다.

모듈에 대한 전체 경로가 지정되지 않은 경우 PAM 라이브러리는 /usr/lib/security 점두사(32bit 서비스 용) 또는 /usr/lib/security/64(64bit 서비스 용)를 모듈 이름에 추가한다.

5) module-option

서비스 모듈에 전달할 수 있는 공백으로 구분된 옵션 목록을 지정한다. 이 필드의 값은 module_path 필드에 정의된 모듈에서 지원하는 옵션에 따라 다르다. 이 필드는 선택 사항이다.

module_type 또는 control_flag 필드에 대한 잘못된 값을 가진 잘못된 항목이나 항목은 PAM 라이브러리에서 무시된다. 줄의 시작 부분에 숫자 기호 (#)로 시작하는 항목도 주석을 나타 내기 때문에 무시된다.

PAM은 일반적으로 "스태킹"이라고 하는 개념을 지원하므로 각 서비스에 여러 메커니즘을 사용할 수 있다.

스태킹은 동일한 module_type 필드를 가진 서비스에 대한 여러 항목을 작성하여 구성 파일에 구현된다.

모듈은 지정된 서비스의 파일에 나열된 순서대로 호출되며 최종 결과는 각 항목에 지정된 control_flag 필드에 의해 결정된다.

arguments	설명	비고
debug	시스템 로그에 디버깅 정보를 남기다.	
no_warn	응용 프로그램에 경고 메시지를 제공하지 않는다.	
use_first_pass	비밀번호를 두 번 확인하지 않는다. 대신 auth 모듈에서 입력한 비밀번호를 사용자 인증 과정 시에도 재사용 해야 한다. (이 옵션은 auth 및 password 모듈에 해당하는 옵션임)	
try_first_pass	이 옵션은 use_first_pass 옵션과 비슷한데, 사용자는 두 번 비밀번호를 입력할 필요가 없기 때문이다. 하지만 기존의 비밀번호를 다시 입력하도록 되어 있다.	
use_mapped_pass	이 인자는 이전 모듈에서 입력된 텍스트 인증 토큰을 입력 받도록 하는데, 이 값으로 암호화 또는 암호화가 해제된 키 값을 생성한다. 그 이유는 모듈	

	에 대한 인증 토큰 값을 안전하게 저장하거나 불러오기 위함이다.	
expose_account	이 값은 모듈로 하여금 계정 정보를 중요하다고 판단하지 않게 한다. 시스템 관리자에 의해 임의로 설정한 것이라 여겨진다.	
nullok	이 인자는 호출된 PAM 모듈이 null 값의 비밀번호를 입력하는 것을 허용한다.	

다음 /etc/pam.conf 하위 세트는 로그인 서비스의 인증 모듈 유형에서 스택하는 예제입니다.

```
##
PAM configuration file /etc/pam.conf
# Authentication Management
login auth required /usr/lib/security/pam_ckfile file=/etc/nologin
login auth required /usr/lib/security/pam_aix
login auth optional /usr/lib/security/pam_test use_first_pass
OTHER auth required /usr/lib/security/pam_prohibit
```

구성 파일의 예에는 로그인 서비스에 대한 세 가지 항목이 들어 있다. 필요에 따라 pam_ckfile과 pam_aix를 모두 지정하면 두 모듈이 모두 실행되고 전체 결과가 성공적으로 이루어져야 한다.

가상의 pam_test 모듈에 대한 세 번째 항목은 선택 사항이며 성공 또는 실패 여부는 사용자가 로그인 할 수 있는지 여부에 영향을 주지 않는다.

pam_test 모듈에 대한 use_first_pass 옵션은 새로운 암호를 요구하는 대신 이전에 입력한 암호를 사용해야 한다.

OTHER 키워드를 서비스 이름으로 사용하면 구성 파일에 명시적으로 선언되지 않은 다른 서비스에 대해 기본값을 설정할 수 있다.

기본값을 설정하면 주어진 모듈 유형에 대한 모든 케이스가 하나 이상의 모듈에 의해 보호된다.

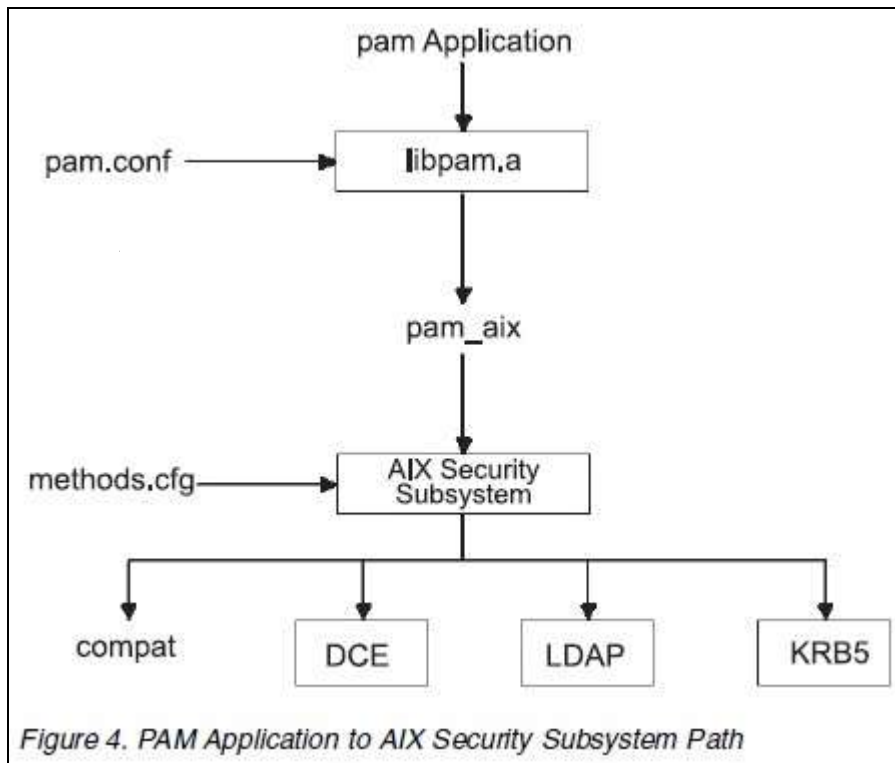
이 예제의 경우 pam_prohibit 모듈이 모든 호출에 대해 PAM 실패를 리턴하기 때문에 login이 아닌 모든 서비스가 항상 실패한다.

4.8 pam_aix 모듈

pam_aix 모듈은 AIX 보안 서비스에 대한 PAM 사용 가능 응용 프로그램 액세스를 제공하는 PAM 모듈로서, 해당 AIX 서비스를 호출하는 인터페이스를 제공하여 AIX 보안 서비스에 액세스한다.

이러한 서비스는 로드 가능한 인증 모듈 또는 사용자 정의 및 methods.cfg 파일의 해당 설정에 따라 AIX 내장 함수에 의해 차례로 수행된다.

AIX 서비스 실행 중 생성된 모든 오류 코드는 해당 PAM 오류 코드에 맵핑된다.



이 그림은 pam_aix 모듈을 사용하도록 /etc/pam.conf 파일을 구성한 경우 PAM 응용 프로그램 API 호출이 따르는 경로를 보여준다.

다이어그램에 표시된 대로 통합을 통해 로드 가능한 인증 모듈(DCE, LDAP 또는 KRB5) 또는 AIX 파일 (compat) 중 하나를 통해 사용자를 인증할 수 있다.

pam_aix 모듈은 /usr/lib/security 디렉토리에 설치된다.

pam_aix 모듈을 통합하려면 모듈을 사용하도록 /etc/pam.conf 파일을 구성해야 한다.

스태킹은 계속 사용할 수 있지만 다음과 같은 /etc/pam.conf 파일에는 표시되지 않는다.

```

##
Authentication management
#
OTHER auth required /usr/lib/security/pam_aix
##
Account management
#
OTHER account required /usr/lib/security/pam_aix
##
Session management
#
OTHER session required /usr/lib/security/pam_aix
##
Password management
#
OTHER password required /usr/lib/security/pam_aix
  
```

pam_aix 모듈에는 pam_sm_authenticate, pam_sm_chauthok 및 pam_sm_acct_mgmt SPI 함수에 대한 구현이

있다.

pam_sm_setcred, pam_sm_open_session 및 pam_sm_close_session SPI는 pam_aix 모듈에도 구현되지만 이러한 SPI 함수는 PAM_SUCCESS 호출을 반환한다.

다음은 AIX 보안 서브 시스템에 대한 PAM SPI 호출의 대략적인 맵핑이다.

PAM SPI	AIX
pam_sm_authenticate	→ authenticate
pam_sm_chauthtok	→ passwdexpired, chpass Note: passwdexpired는 PAM_CHANGE_EXPIRED_AUTHTOK 플래그가 전달된 경우에만 검사된다.
pam_sm_acct_mgmt	→ loginrestrictions, passwdexpired
pam_sm_setcred	→ 비교 가능한 매핑이 없으며, PAM_SUCCESS가 반환된다.
pam_sm_open_session	→ 비교 가능한 매핑이 없으며, PAM_SUCCESS가 반환된다.
pam_sm_close_session	→ 비교 가능한 매핑이 없으며, PAM_SUCCESS가 반환된다.

AIX 보안 서브 시스템으로 전달되는 데이터는 모듈 사용 이전의 pam_set_item 기능 또는 데이터가 존재하지 않는 경우 pam_aix 모듈을 사용하여 설정할 수 있다.

4.9 PAM 모듈 추가

PAM 모듈을 추가하여 여러 인증 메커니즘을 사용할 수 있다.

step 1) 모듈의 32 비트 버전을 /usr/lib/security 디렉토리에 두고 모듈의 64 비트 버전을 /usr/lib/security/ 64 디렉토리에 위치시켜야 한다.

step 2) 파일 소유권을 루트 및 사용 권한으로 555로 설정해야 한다. PAM 라이브러리는 루트 사용자가 소유하지 않은 모듈을 로드하지 않는다.

step 3) /etc/pam.conf 구성 파일을 갱신하여 원하는 서비스 이름의 항목에 모듈을 포함시켜야 한다.

step 4) 영향을 받은 서비스를 테스트하여 기능을 확인해야 한다. 로그인 테스트가 수행될 때까지 시스템을 로그 오프 하지 말아야 한다.

4.10 /etc/pam.conf 파일 변경

/etc/pam.conf 파일을 변경하기 전에 고려해야 할 사항이 몇 가지 있는데, /etc/pam.conf 구성 파일을 변경할 때 다음 요구 사항을 고려해야 한다.

1) 파일은 항상 루트 사용자 및 그룹 보안이 소유해야 한다. 모든 사용자가 읽기 액세스를 허용하지만 루트 만 수정할 수 있게 하려면 파일에 대한 사용 권한이 644 여야 한다.

2) 보안을 강화하려면 각 PAM 사용 가능 서비스를 명시적으로 구성한 다음 OTHER 서비스 키워드에 대해 pam_prohibit 모듈을 사용하는 것이 좋다.

3) 선택한 모듈에 대해 제공되는 모든 설명서를 읽고 지원되는 제어 플래그 및 옵션과 그 영향에 대해 결정해야 한다.

4) 스택 모듈의 required, requisite, sufficient 및 optional 제어 플래그의 동작을 염두에 두고 모듈 및 제어 플래그의 순서를 신중하게 선택해야 한다.

주 : PAM 구성 파일을 잘못 구성하면 루트를 포함하여 모든 사용자에게 구성이 적용되므로 로그인 할 수 없는 시스템이 발생할 수 있다.

파일을 변경한 후에는 시스템에서 로그 아웃하기 전에 영향을 받는 응용 프로그램을 항상 테스트해야 한다.

로그인 할 수 없는 시스템은 시스템을 유지 보수 모드로 부팅하고 /etc/pam.conf 구성 파일을 수정하여 복구할 수 있다.

4.11 PAM 디버그 사용

PAM (Pluggable Authentication Modules) 라이브러리는 실행 중에 디버그 정보를 제공할 수 있다.

시스템이 디버그 출력을 수집하도록 설정한 후에는 수집된 정보를 사용하여 PAM API 호출을 추적하고 현재 PAM 설정에서 오류 지점을 확인할 수 있다.

PAM 디버그 출력을 사용하려면 다음 단계를 완료해야 한다.

step 1) touch 명령을 사용하여 파일이 없으면 /etc/pam_debug 디렉토리에 pam_debug라는 빈 파일을 작성한다. PAM 라이브러리는 /etc/pam_debug 파일을 검사하고 syslog를 활성화하며, 결과가 발견되면 출력한다.

step 2) /etc/syslog.conf 파일을 편집하여 원하는 우선 순위 수준에서 syslog 메시지를 로깅 할 파일을 식별한다.

예를 들어 PAM 디버그 수준 메시지를 /var/log/auth.log 파일에 보내려면 다음 텍스트를 syslog.conf 파일의 새 행으로 추가해야 한다.

```
*.debug /var/log/auth.log
```

step 3) touch 명령을 사용하여 2 단계에서 참조한 출력 파일 (/var/log/auth.log)이 존재하지 않으면 작성해야 한다.

step 4) 구성 변경 사항이 인식되도록 syslogd 데몬을 다시 시작하려면 다음 단계를 완료해야 한다.

```
> stopsrc -s syslogd
> startsrc -s syslogd
```

PAM 응용 프로그램이 다시 시작되면 /etc/syslog.conf 구성 파일에 정의된 출력 파일에 디버그 메시지가 수집된다.

4.12 SSH(Secure Shell)

Secure Shell은 개방형 소스 SSH 제품인 OpenSSH 제품을 기반으로 한다. (<http://www.openssh.org>)

Secure Shell은 비보안 네트워크에서 클라이언트와 원격 호스트 간의 보안 연결을 가능하게 한다.

이 보안 연결의 주요 속성은 다음과 같다.

- 클라이언트와 원격 호스트 둘 다에 대한 강력한 인증
- 클라이언트와 원격 호스트 간의 통신에 대한 강력한 암호화 및 공개 키 암호화
- 클라이언트와 원격 호스트에서 명령을 실행하는데 사용할 보안 연결

Secure Shell은 telnet, remsh, rlogin, ftp, 및 rcp와 같은 자주 사용하는 함수와 명령의 보안 대체 항목을 제공한다.

1) Secure Shell의 주요 보안 기능

Secure Shell의 주요 보안 기능은 다음과 같다.

① 강력한 암호화

클라이언트와 원격 호스트 간의 모든 통신은 Blowfish, 3DES, AES 및 arcfour와 같은 특허 없는 암호화 알고리즘을 사용하여 암호화된다. 암호 등의 인증 정보는 네트워크에서 일반 텍스트로 전송되지 않는다. 또한 암호화는 강력한 공개 키 기반 암호화와 더불어 잠재적 보안 공격을 차단한다.

② 강력한 인증

Secure shell은 클라이언트와 서버 간의 강력한 인증 방법 집합을 지원한다. 인증은 양방향일 수 있다. 서버는 클라이언트를 인증하고 클라이언트도 서버를 인증한다. 이렇게 하면 다양한 보안 문제로부터 세션을 보호할 수 있다.

③ 포트 전달

클라이언트와 원격 호스트 간의 TCP/IP 연결 리디렉션(및 그 반대)을 포트 전달 또는 SSH 터널링이라고 한다. Secure Shell은 포트 전달을 지원한다. 예를 들어, 포트 전달을 사용하여 클라이언트와 서버 간의 ftp 트래픽(또는 전자 메일 클라이언트와 POP/IMAP 서버 간의 전자 메일 트래픽)을 리디렉션할 수 있다. 클라이언트가 직접 서버와 통신하는 대신 보안 채널을 통해 트래픽을 sshd 서버로 리디렉션할 수 있으며, 그런 다음 sshd 서버에서 트래픽을 실제 서버 시스템의 지정된 포트에 전달할 수 있다.

2) Secure Shell의 소프트웨어 구성 요소

Secure Shell 소프트웨어는 클라이언트 및 서버 구성 요소로 이루어져 있다.

구성요소	설명	위치	대응하는 비보안 구성요소
ssh	Secure Shell을 클라이언트는 telnet 및 remsh의 보안 대체 항목이며 보안 기능이 있는 remsh와 가장 유사함.	클라이언트	remsh, telnet, rlogin
slogin	ssh에 대한 심볼릭 링크임.	클라이언트	remsh, telnet, rlogin
scp	클라이언트 보안 복사 및 서버 보안 복사를 수행함,	클라이언트 및 서버	rcp
sftp	보안 ftp 클라이언트임	클라이언트	ftp

sshd	보안 Shell 데몬임.	서버	remshd, telnetd, rlogind
sftp-server	보안 ftp 데몬임.	서버	ftpd
ssh-rand-helper	sshd가 서버에서 /dev/random 또는 /dev/urandom을 찾을 수 없을 때는 사용되는 Random Number Generator. OS 커널에 상주하는 Random Number Generator인 rng가 포함되어 있다. rng가 구성 해제되어 있으면 sshd는 prngd를 사용함.	서버	해당 사항 없음
ssh-agent	클라이언트에서 서버로의 "자동" 키 기반 로그인 도구임.	클라이언트 및 서버	Rhosts 파일 메커니즘
ssh-add	클라이언트의 키 쌍을 ssh-agent에 알리는 도구임.	클라이언트	해당 사항 없음
ssh-keygen	공개 키 인증을 위해 키 쌍을 생성하는 도구임.	클라이언트	해당 사항 없음
ssh-keyscan	Secure Shell 데몬(sshd)을 실행하는 호스트 집합에 대한 공개 키를 수집하는 클라이언트 도구임.	클라이언트	해당 사항 없음
ssh-keysign	호스트 기반 인증 중에 필요한 디지털 서명을 생성하는 도구임. ssh()에서 로컬 호스트 키 호스트 기반 인증에 액세스하는데 사용됨.	클라이언트	해당 사항 없음

3) Secure Shell 실행

위 표에서 나열된 Secure Shell 클라이언트를 실행하기 전에 먼저 Secure Shell 서버 데몬 sshd를 시작한다.

sshd 데몬은 서버 시스템의 /opt/ssh/etc 디렉토리에 있는 sshd_config 파일에서 초기 값을 가져온다.

sshd_config에 있는 가장 중요한 구성 지시어 중 하나는 sshd 데몬에서 지원하는 인증 방법 집합이다.

- ssh 클라이언트 실행

ssh 클라이언트 응용 프로그램은 sshd 서버와의 소켓 연결을 설정한다.

sshd 서버는 자식 sshd 프로세스를 시작한다.

이 자식은 연결 소켓을 상속받고 선택된 인증 방법을 기반으로 클라이언트를 인증한다.

인증에 성공한 경우에만 성공적으로 보안 클라이언트 세션이 설정된다.

세션이 만들어진 후 모든 후속 통신은 클라이언트와 이 자식 sshd 프로세스 간에 직접 이루어진다.

이제 클라이언트는 서버에서 원격 명령을 실행할 수 있다.

ssh 클라이언트의 명령 요청이 있을 때마다 자식 sshd 프로세스에서 해당 명령을 실행할 Shell 프로세스를 시작한다.

간단히 말해서 실행 중인 ssh 클라이언트-서버 세션은 다음 프로세스로 구성된다.

① sshd 서버에 연결된 모든 클라이언트 시스템에는 현재 이 클라이언트 시스템에서 설정된 각 ssh 연결에 대한 하나의 ssh 클라이언트 프로세스가 있다.

② 서버 시스템에는 하나의 부모 sshd 프로세스와 서버에 연결된 동시 ssh 클라이언트 개수 만큼의 자식 sshd 프로세스가 있다. 서버에 권한 분리가 활성화되어 있으면 서버에서 실행되는 자식 sshd 프로세스의 수가 두 배로 증가한다.

③ ssh 클라이언트에서 명령 실행 요청이 있을 때마다 서버 시스템의 해당 자식 sshd 프로세스는 Shell 프로세스를 시작하고 Unix 파이프를 사용하여 명령 요청을 이 Shell 프로세스로 전달한다. 이 Shell 프로세스는 Unix 파이프를 사용하여 명령 실행 결과를 자식 sshd 프로세스로 반환하고 명령 실행이 완료되면 종료된다.

- sftp 클라이언트 실행

sftp 클라이언트 응용 프로그램은 sftp 클라이언트 응용 프로그램이 ssh 클라이언트를 시작하도록 하고 Unix 파이프를 사용하여 이 클라이언트와 통신한다. 그런 다음 ssh 클라이언트는 sshd 서버와의 소켓 연결을 설정한다.

서버 상호 작용의 나머지 부분은 ssh 클라이언트의 경우와 유사하다. 차이점은 원격 명령을 실행한 Shell을 시작하는 대신 자식 sshd 프로세스가 sftp-server 프로세스를 시작한다는 것이다. 이 sftp 세션 중의 모든 후속 통신은 다음 프로세스 간에 발생한다.

- ① Unix 파이프를 사용하여 클라이언트 시스템에서 sftp 클라이언트와 ssh 클라이언트 간에
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Unix 파이프를 사용하여 자식 sshd 프로세스와 sftp 서버 프로세스 간에

- scp 클라이언트 실행

scp 클라이언트의 경우는 sftp 클라이언트 실행과 거의 동일하다. 차이점은 sftp-server 프로세스를 시작하는 대신 자식 sshd 프로세스가 scp 프로세스를 시작한다는 것이다. scp 세션 중의 모든 후속 통신은 다음 프로세스간에 발생한다.

- ① 클라이언트 시스템에서 scp 클라이언트와 ssh 클라이언트 간에, Unix 파이프 사용
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Unix 파이프를 사용하여 자식 sshd 프로세스와 scp 서버 프로세스 간에

4) Secure Shell 권한 분리

Secure Shell은 권한 분리 기능을 통해 보다 향상된 수준의 보안을 제공한다. 부모 sshd 및 자식 sshd 프로세스는 권한이 부여된 사용자로 실행된다. 권한 분리를 활성화하면 사용자 연결당 하나의 추가 프로세스가 시작된다.

ssh 클라이언트가 권한 분리에 대해 구성된 sshd 서버에 연결하면 부모 sshd 프로세스가 권한이 부여된 자식 sshd 프로세스를 시작한다. 권한 분리를 활성화하면 자식 sshd 프로세스는 권한이 없는 자식 sshd 프로세스를 추가로 시작한다. 권한이 없는 자식 sshd 프로세스는 연결 소켓을 상속받는다. 클라이언트와 서버 간의 모든 후속 통신은 권한이 없는 이 자식 sshd 프로세스를 사용하여 이루어진다.

클라이언트의 원격 명령 실행 요청은 대부분 비권한이며 권한이 없는 이 자식 sshd 프로세스에서 시작된 Shell에 의해 처리된다. 권한이 없는 자식 sshd 프로세스에서 권한이 부여된 기능을 실행해야 하는 경우 Unix 파이프를 사용하여 권한이 부여된 부모 sshd 프로세스와 통신한다.

권한 분리는 침입자에 의한 잠재적 손상을 제한하는데 도움이 된다. 예를 들어, Shell 명령을 실행하는 동안 버퍼 오버플로 공격이 발생할 경우 권한이 없는 프로세스 내에서 제어되므로 잠재적 보안 위험이 제한된다.

권한 분리는 Secure Shell의 기본 구성이다. sshd_config 파일에서 "UsePrivilegeSeparation NO"를 설정하여 권한 분리를 해제할 수 있다. 잠재적 권한 위험이 있으므로 권한 분리를 해제할 경우 신중하게 고려해야 한다.

5) Secure shell 인증

Secure Shell은 다음 인증 방법을 지원한다.

- GSS-API(Kerberos 기반 클라이언트 인증)
- 공개 키 인증
- 호스트 기반 인증
- 암호 인증

클라이언트는 원격 sshd 데몬과 연결될 때 원하는 인증 방법(앞에 나열된 방법 중 하나)을 선택하고 연결 요청의 일부로 적절한 자격 증명을 제공하거나 서버에서 보낸 프롬프트에 응답한다. 모든 인증 방법이 이런 방식으로 작동한다.

서버가 성공적으로 연결을 설정하려면 클라이언트로부터 적절한 키, 암호구, 암호 또는 자격 증명을 받아야 한다.

sshd 인스턴스에서 보안 요구 사항을 기반으로 지원되는 인증 방법 중 일부만 지원하도록 선택할 수 있다.

Secure Shell은 앞에 나열된 인증 방법을 지원하지만 시스템 관리자가 환경의 특정 보안 요구 사항을 기반으로 sshd 인스턴스에서 제공되는 인증 방법을 제한할 수 있다. 예를 들어, Secure Shell 환경에서 모든 클라이언트가 공개 키 또는 Kerberos 방법을 사용하여 인증해야 한다고 지정할 수 있으며, 이로 인해 나머지 방법은 비활성화될 수 있다. 지원되는 인증 방법 활성화 및 비활성화는 sshd_config 파일에 지정된 구성 지시어를 통해 이루어진다.

ssh 클라이언트 연결 요청이 있으면 서버는 먼저 지원되는 인증 방법 목록으로 응답한다. 이 목록은 sshd 서버에서 지원하는 인증 방법과 이러한 방법이 시도되는 시퀀스를 나타낸다. 클라이언트는 이러한 인증 방법을 하나 이상 생략할 수 있다. 클라이언트에서 방법이 시도되는 시퀀스를 변경할 수도 있다. 이렇게 하려면 클라이언트 구성 파일 /etc/ssh/etc/ssh_config의 구성 지시어를 사용한다.

Secure Shell에서 지원하는 인증 방법은 다음과 같이 요약되어 있다.

① GSS-API(Kerberos 기반 클라이언트 인증)

Kerberos 기반 클라이언트 인증인 GSS-API(Generic Security Sservice Application Programming Interface)를 사용하는 경우 클라이언트가 미리 Kerberos 자격 증명을 받아야 하며, 해당 클라이언트 디렉토리에 Kerberos 구성 파일이 있어야 한다.

클라이언트는 sshd 데몬과 연결될 때 연결 시 자격 증명을 제공한다.

서버는 이러한 자격 증명을 이 특정 사용자의 자격 증명 복사본과 일치시킨다.

또한 선택적으로 클라이언트 호스트 환경의 타당성을 설정할 수 있다.

② 공개 키 인증

공개 키 인증을 사용하려면 Secure Shell 환경에 다음 설정이 있어야 한다.

클라이언트와 서버 둘 다에 키 쌍이 있어야 한다. 모든 ssh 클라이언트와 모든 sshd 서버가 ssh-keygen 유틸리티를 사용하여 자체적으로 키 쌍을 생성해야 한다.

클라이언트는 통신해야 하는 모든 sshd 서버에 해당 공개 키를 알려야 한다. 이렇게 하려면 각 클라이언트의 공개 키를 모든 관련 서버의 미리 지정된 디렉토리에 복사한다.

클라이언트는 통신해야 하는 모든 서버의 공개 키를 구해야 한다. 클라이언트는 ssh-keyscan 유틸리티를 사용하여 공개 키를 받는다.

이 설정이 완료되면 sshd 서버에 연결하는 ssh 클라이언트가 공개 키와 개인 키를 사용하여 인증된다.

Secure Shell은 공개 키 인증을 단순화하는 추가 기능을 제공한다. 일부 환경에서는 항상 암호 프롬프트에 응답할 필요가 없도록 설정할 수 있다. 둘 다 클라이언트 시스템에서 실행되는 ssh-agent 및 ssh-add 프로세스를 조합해서 사용하면 암호에 응답하지 않아도 된다. 클라이언트는 ssh-add 유틸리티를 통해 ssh-agent 프로세스에 모든 키 정보를 등록한다. 그런 다음 클라이언트와 서버 간의 공개 키 인증은 sshd 데몬이 클라이언트와 상호 작용할 필요 없이 ssh-agent에 의해 수행된다.

③ 호스트 기반 및 공개 키 인증

호스트 기반 및 공개 키 인증은 보다 안전한 공개 키 인증 방법의 확장이다.

클라이언트와 서버 둘 다의 키 쌍이 있는 것 외에도 이 방법을 사용하면 클라이언트 환경에서 통신할 서버를 제한할 수 있다.

클라이언트의 홈 디렉토리에 .rhosts 파일을 만들어 이 제한을 구현한다.

④ 암호 인증

암호 인증 방법은 단일 사용자-ID 및 암호 기반 로그인을 사용한다. 이 로그인은 /etc/passwd에 지정된 사용자 로그인을 기반으로 하거나 PAM 기반일 수 있다.

Secure Shell은 서버 시스템에서 사용할 수 있는 PAM 모듈과 완전히 통합된다. 이 목적을 위해 /opt/ssh/etc/sshd_config 파일에 UsePAM 구성 지시어가 있다. YES로 설정하면 클라이언트에서 암호 인증 요청이 있을 때마다 sshd는 PAM 구성 파일(/etc/pam.conf)을 확인한다. 그런 다음 구성된 PAM 모듈을 통해 암호 인증이 성공할 때까지 순서대로 수행한다.

PAM 인증을 무시하려면 UsePAM 지시어를 NO로 설정한다. 그러면 클라이언트에서 암호 인증 요청이 있을 때 sshd가 서버의 PAM 구성 설정을 무시한다. 대신 sshd는 gwtppnam() 라이브러리 호출을 직접 호출하여 사용자 암호 정보를 가져온다.

Secure Shell은 PAM_UNIX, PAM_LDAP 및 PAM_KERBEROS를 사용하여 테스트 되었다. 또한 PAM_DCE 및 PAM_NTLM과 같은 다른 PAM 모듈에서 작동한다.

6) 통신 프로토콜

Secure Shell 사용자는 SSH-1 또는 SSH-2 프로토콜을 사용하여 원격 sshd 데몬과 연결할 수 있다. SSH-2가 더 안전하므로 SSH-1 대신 권장한다.

7) Secure Shell에서 사용하는 기능의 일부 목록

Secure Shell은 실제로 Shell이 아니라 호스트에서 인전하게 원격 shell 세션을 실행하기 위해 클라이언트와 원격 호스트 간에 보안 연결을 만드는 매커니즘이다. 보안 연결을 설정하기 위해 Secure Shell에서 인증과 세션 생성을 대부분 수행한다. Secure Shell에서 사용하는 기능의 일부 목록은 다음과 같다.

① login 시도 기록

telnet 또는 remsh와 마찬가지로 Secure Shell은 성공한 세션과 실패한 세션을 각각 /var/adm/wtmp 및 /var/adm/btmp 파일에 기록한다.

② PAM 모듈

Secure Shell은 클라이언트 세션에 대해 PAM 인증을 사용할 수 있다. PAM 인증을 선택하면 Secure Shell은 /etc/pam.conf 파일을 사용하고 인증을 위해 해당 PAM 모듈을 호출한다.

③ /etc/default/security 파일 사용

로그인 동작 암호 및 기타 보안 구성을 정의하는 속성이 들어 있는 시스템 범위 구성 파일이다. 몇 가지 제한은 있지만 Secure Shell에서 이러한 속성을 사용할 수 있다.

④ Shadow passwd

Secure Shell은 Shadow passwd 기능과 통합된다.

⑤ 컨트롤 시스템 로그(syslog)

Secure Shell은 syslog를 사용하여 중요한 메시지를 남긴다.

⑥ 감사 로깅

Secure shell은 해당 코드로 감사 로깅(트러스트된 모드)을 구현했다.

8) 비밀번호 없이 SSH에 접속하는 방법

SSH 접속 시 비밀번호 없이 접속하는 방법은 다음과 같다.

① ssh key 생성 (없는 경우만)

```
$ ssh-keygen -t rsa
```

② 서버로 로컬에서 만든 공개키 복사

```
$ scp ~/.ssh/id_rsa.pub [id]@[address]:id_rsa.pub
```

③ 서버 접속 (일반접속)

```
$ ssh [id]@[address]
```

④ 로그인 후 개인폴더에 .ssh 폴더 생성 (있으면 pass)

```
$ mkdir .ssh
```

⑤ .ssh폴더 권한 변경

```
$ chmod 700 .ssh
```

⑥ 공개키 인증키 목록에 추가

```
$ cat id_rsa.pub >> .ssh/authorized_keys
```

⑦ 필요없는 공개키 제거

```
$ rm -f id_rsa.pub
```

⑧ 인증키 목록 권한 수정

```
$ chmod 644 .ssh/authorized_keys
```

⑨ 서버 접속

```
$ ssh -i ~/.ssh/id_rsa [id]@[address]
```

4.13 PAM 프로그래밍

아래 소스 코드는 PAM 모듈의 기본 형태로 "#if 0"으로 막혀 있는 부분에 통제를 할 수 있는 코드를 넣어 두면 원하는 동작(차단 또는 허용, 로그 기록)을 쉽게 처리할 수 있다.

```
/* pam_test.c */

#include <stdio.h>
#include <stdlib.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>

#ifdef PAM_MODULE_ENTRY
PAM_MODULE_ENTRY("pam_test");
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN extern
#endif

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

/* expected hook, this is where custom stuff happens */
PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    #if 0
```

```
// 아래 조건을 만족하지 못할 경우 로그인 실패로 처리함
if (check_auth_pam(pamh) < 0)
    return PAM_PERM_DENIED;
#endif
return PAM_SUCCESS;
}
```

위에 있는 pam_sm...으로 된 함수들이 사용자가 임의로 추가한 함수가 아니라 PAM에서 기본 함수들로서 해당 함수 부분에 원하는 동작 코드를 넣도록 되어 있다.

PAM 모듈 자체가 기존에 동작하는 telnet이나, ssh, login 등의 바이너리에 동적 라이브러리 형태로 붙기 때문에 위 코드에 삽입한 코드들은 로그인과 동시에 해당 서버로의 접속자의 모든 상황들을 감시하고 제어할 수 있다.

4.14 PAM 컴파일 방법

PAM 인증 모듈은 Aix의 특성별 OS에서 제공하는 최적화 옵션에 따라 컴파일 하는 방법은 다음과 같다.

```
> sudo gcc -g -o baro_auth baro_auth.c -lc -ldl
> sudo gcc --std=gnu99 -Wall -O2 -g -fPIC -c -g -fno-strict-aliasing -I/usr/include -o base64.o base64.c
> sudo gcc --std=gnu99 -Wall -O2 -g -fPIC -c -g -fno-strict-aliasing -I/usr/include -o xxtea.o xxtea.c
> sudo gcc --std=gnu99 -Wall -O2 -g -fPIC -c -g -fno-strict-aliasing -I/usr/include -o pam_baro_auth.o pam_baro_auth.c
> sudo gcc -shared -g -o pam_baro_auth.so pam_baro_auth.o base64.o xxtea.o -lpam -L/user/lib64 -lssl -lcrypto -ldl -lz -lc -lcfg -lodm -lm
> chmod 555 pam_baro_auth.so
> chown root pam_baro_auth.so
```

PAM 프로그램 컴파일할 때 반드시 sudo 명령어를 사용하여 컴파일 해야 한다. 그렇지 않으면 다음과 같은 오류가 발생한다.

```
Oct 25 17:26:34 project sshd[11609]: open_module: module /usr/baropam/pam_baro_auth.so writable by group
Oct 25 17:26:34 project sshd[11609]: load_modules: can not open module /usr/baropam/pam_baro_auth.so
```

sudo 명령어는 유닉스 및 유닉스 계열 운영 체제에서 다른 사용자의 보안권한과 관련된 프로그램을 구동할 수 있게 해주는 프로그램이다.

이것은 substitute user do (다른 사용자의 권한으로 명령을 이행하라, 는 뜻이다.) 의 줄임말이다.

기본적으로 Sudo는 사용자 비밀번호를 요구하지만 루트 비밀번호(root password)가 필요할 수도 있고, 한 터미널에 한번만 입력하고 그 다음부터는 비밀번호가 필요 없다.

Sudo는 각 명령줄에 사용할 수 있으며 일부 상황에서는 관리자 권한을 위한 슈퍼유저 로그인(superuser login)을 완벽히 대신하며, 주로 우분투, 리눅스와 애플의 OS X에서 볼 수 있다.

참고로 PAM 프로그램 컴파일할 때 필요한 기본적인 환경변수인 PATH(명령어 입력시 경로를 입력하지 않고

실행가능), LIBPATH(외부 라이브러리를 링크할 때 참조할 경로)를 다음과 같이 설정한다.

```
export LIBPATH=$LIBPATH:/opt/freeware/lib:/usr/lib64:
export PATH=$PATH:/usr/bin:/etc/alternatives
```

PAM을 컴파일하기 위해서는 암호화 모듈에 "openssl" 라이브러리를 사용하므로 반드시 "openssl"을 다음과 같은 명령어로 설치해야 한다.

```
> installp -ac -Y -d . openssl.base openssl.base openssl.man.en_US openssl.man.en_US
```

4.15 PAM 테스트 방법

PAM 프로그램 컴파일 또는 PAM 구성 환경 변경 후 sshd를 restart하지 않고 테스트할 수 있는 방법은 다음과 같이 진행할 수 있다.

1) sshd 데몬 기동하기

sshd 데몬(22000 포트)을 디버깅 모드로 띄우기 위하여 다음과 같은 명령어를 수행한다.

```
> /usr/sbin/sshd -D -p22000 -d
```

2) ssh로 접속하기

sshd 데몬(22000 포트)에 ssh로 접속하기 위하여 다음과 같은 명령어를 수행한다.

```
> ssh -vvv -p22000 root@1.234.83.169
```

5. Linux

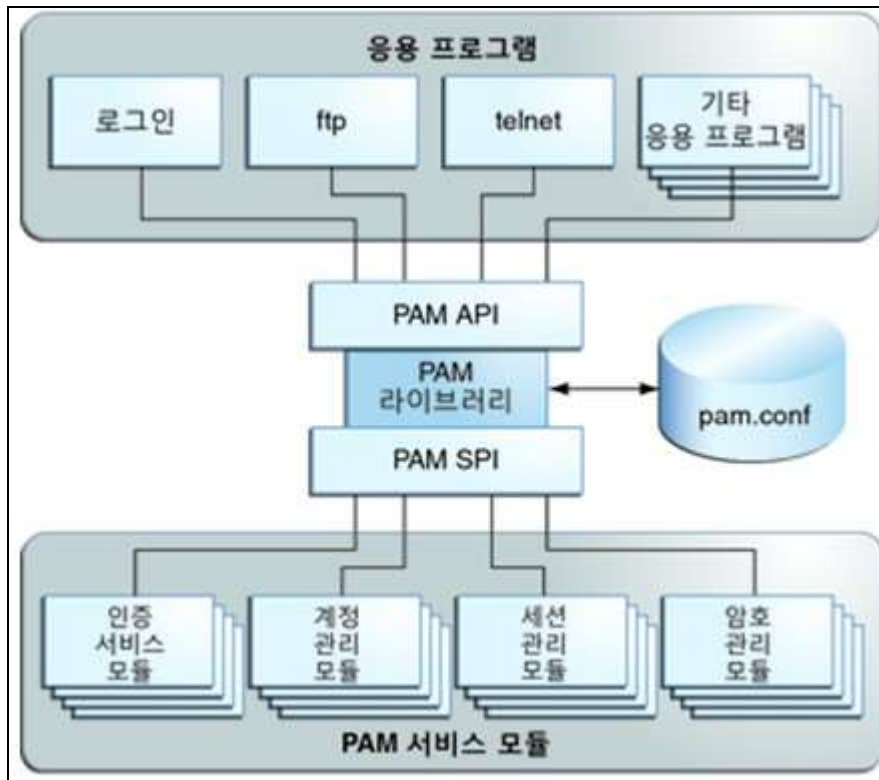
5.1 PAM 개요

PAM(Pluggable Authentication Module, 플러그 가능한 인증 모듈)은 Linux/Unix 시스템에서 서비스를 재 컴파일하지 않고, 다양한 인증 기술을 시스템 항목 서비스에 접목할 수 있도록 해주는 프레임워크로 중앙 집중적인 인증 매커니즘을 지원하는 것이다. 게다가 시스템의 기본적인 인증 기법을 제공하여 이것을 사용하면 응용 프로그램 개발자 뿐만 아니라 시스템 관리자들이 인증을 유연성 있게 관리할 수 있도록 도와 준다.

전통적으로 시스템 자원에 대한 접근을 관리하는 프로그램들은 내장된 메커니즘에 의해 사용자 인증 과정을 수행한다. 이러한 방식은 오랫동안 이루어졌지만 이러한 접근 방식은 확장성이 부족하고 매우 복잡하다. 그렇기 때문인지 이러한 인증 매커니즘을 끌어내기 위한 수많은 해킹 시도가 있었다.

Solaris의 방식을 따라서 Unix/Linux 사용자들은 그들만의 PAM을 구현하는 방식을 찾았다.

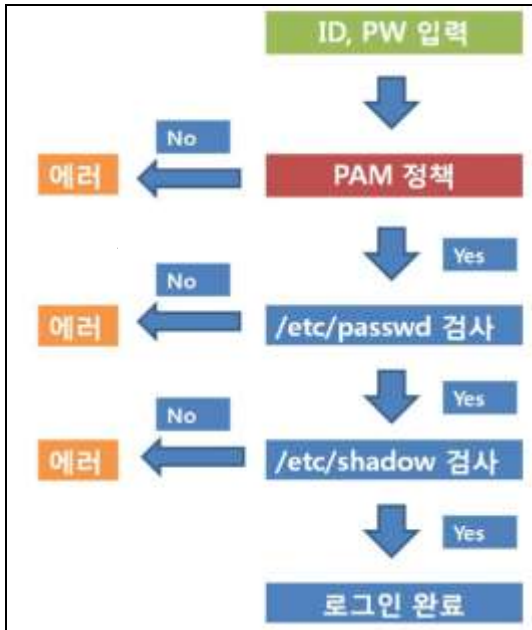
PAM의 아키텍처는 다음과 같다.



PAM의 기본 원리는 응용 프로그램이 password 파일을 읽어 오는 대신 PAM이 직접 인증을 수행 하도록 하는 것이다. PAM은 시스템 관리자가 원하는 인증 매커니즘이 무엇이든 상관하지 않는다.

여러 사이트에서 선택 받은 인증 매커니즘은 아직도 password 파일이다. 왜 그럴까? 우리가 원하는 것을 해주기 때문이다. 대부분의 사용자는 password 파일이 필요한 것이 무엇인지 이미 알고 있다. 그리고 원하는 작업을 수행하는데 있어 이미 그 기능이 검증되었기 때문일 것이다.

PAM의 인증 절차는 다음과 같다.



5.2 PAM 동작 원리

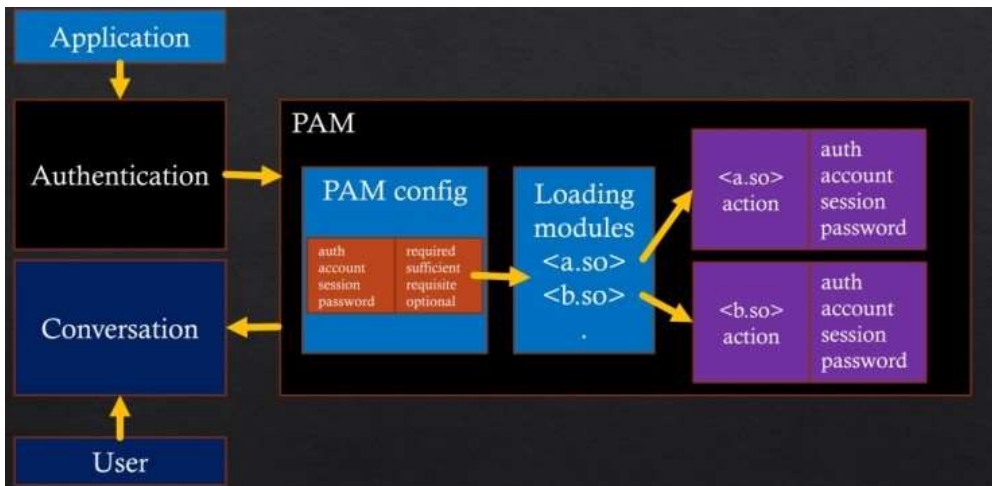
PAM은 Windows 환경의 DDL(Dynamic Link Library)과 같은 것이다. Library로 프로그램이 어떤 사용자에게 대한 인증을 수행하려면 PAM Library가 있는 함수를 호출한다. PAM은 해당 함수의 Library를 제공하여 응용 프로그램이 특정 사용자를 인증하도록 요청할 수 있다.

PAM을 통해 /etc/passwd 파일이나 /etc/shadow 파일을 확인하거나 또는 보다 복잡한 확인 작업을 수행하기도 하는데, 예로는 LDAP 서버에 접속하는 것이다.

확인 작업을 마치고, 인증 여부를 결정하게 되면 “인증됨/인증되지 않음”의 메시지를 자신을 호출한 응용 프로그램에 전송한다.

간단한 확인 작업이라고 했는데, 그 과정이 많아 보일 수가 있다. 여기에 나오는 각 모듈은 크기가 작고 작업 수행 시간이 매우 빠르다. 이것은 매우 놀랍기도 하지만 PAM을 사용하게 된다.

PAM의 동작은 프로그램의 실행 하자마자 실행되는 보안 설정 파일들의 프로그램들이다.



PAM의 동작 원리는 다음과 같다.

- 인증이 필요한 프로그램 동작 시 PAM 라이브러리 호출
- PAM설정 파일을 참조
- 참조한 파일의 내용을 바탕으로 모듈 동작
- 프로그램에 동작 결과를 반환하여 인증여부 결정.

5.3 PAM 사용 이점

PAM을 사용하면 사용자 인증을 위한 시스템 항목 서비스(예: ftp, login, telnet, rsh) 사용을 구성할 수 있다. PAM이 제공하는 몇 가지 이점은 다음과 같다.

- 유연한 구성 정책
- 응용 프로그램별 인증 정책
- 기본 인증방식을 선택할 수 있는 기능
- 높은 보안 시스템에서 여러 권한 부여를 요구할 수 있는 기능
- 최종 사용자의 사용 편의성
- 암호가 여러 인증 서비스에 대해 동일한 경우 암호 재입력 없음
- 사용자가 여러 명령을 입력할 필요 없이 여러 인증 서비스에 대해 사용자에게 암호를 요구할 수 있는 기능
- 사용자 인증 서비스에 선택적 옵션을 전달할 수 있는 기능
- 시스템 항목 서비스를 변경할 필요 없이 사이트별 보안 정책을 구현할 수 있는 기능

5.4 PAM 파일과 위치

파일 위치	설명	비고
/lib64/security /lib/security	제공 가능한 PAM 모듈 디렉토리로 실제 PAM Library를 동적으로 인증 모듈을 호출하여 실행한다. (*so)	
/etc/security	PAM 실행에 필요한 설정파일로 /lib64/security에 있는 모듈에 대한 설정 파일. (서비스명.conf)	
/etc/pam.d	PAM 데몬 파일(애플리케이션별 PAM의 설정 파일 위치)로 PAM을 사용하는 응용 프로그램이 설정 파일이 없다면 기본값으로 설정된 설정 파일을 자동으로 사용한다.	

5.5 PAM 라이브러리

라이브러리	설명	비고
pam_permit.so pam_deny.so	항상 성공/실패를 리턴해서 접근을 허용, 거부.	
pam_warn.so	호출한 사용자 및 호스트 정보를 syslog에 기록.	
pam_access.so	사용자 계정, 호스트/도메인을 통해서 시스템 접근을 허용. /etc/security/access.conf에서 설정한다.	
pam_unix.so	암호를 확인하고 변경되는데 사용되는 모듈.	

pam_pwdb.so	/etc/passwd나 shadow 사용시 /etc/shadow 파일에서 계정의 정보를 얻어온다.	
pam_env.so	환경 변수를 지정. /etc/security/pam_env.conf 설정 파일에서 환경 변수를 불러온다. /etc/environment와 /etc/security/pam_env.conf 설정 파일이 비었을 경우 변수 설정 없이 성공 값을 반환한다.	
pam_issue.so pam_motd.so	로그인 시 issue나 message를 출력하며, 성공적으로 로그인한 후 나타난다. /etc/issue, /etc/motd	
pam_tally2.so	로그인 시도 횟수를 세는 모듈로 일정 횟수 이상 실패 시 접근을 차단 및 관 리하는 역할을 수행.	
pam_limits.so	시스템의 자원에 대한 사용자 제한을 설정할 때 주로 사용되는 모듈로 시스 템 로그인 전/후에 수행되는 session 인터페이스와 많이 사용됨. /etc/security/limits.conf	
pam_listfile.so	임의의 파일을 기반으로 특정 서비스에 대해 허가 목록이나 거부 목록을 만 들 때 사용됨. Redhat 기준 /etc/pam.d/vsftpd 파일에 설정되어 ftp 사용자 허용/거부 리 스트를 관리할 때 사용.	
pam_mkhomedir.so	사용자 홈 디렉토리가 없다면 홈 디렉토리를 생성하고, /etc/skel 디렉토리 에 있는 파일들을 복사.	
pam_nologin.so	/etc/nologin 파일이 존재하면 root만 로그인 가능하고, 일반 사용자 로그 인 불가능하며, root로 로그인 시 nologin 파일의 내용이 보여진다.	
pam_rhosts_auth.so	rhost 방식으로 네트워크간 원격 세션을 허용.	
pam_rootok.so	UID가 0인 사용자를 인증하는 모듈로 보통 root가 암호 입력 없이 해당 서 비스에 대한 접근을 허용할 때 주로 사용됨.	
pam_securetty.so	접속하는 계정이 root인 경우 /etc/securetty 파일에 기록된 터미널을 통하 는 경우에만 허용, 나머지 사용자의 경우에는 항상 “성공” 한 것으로 처 리.	
pam_time.so	시간, 사용자, 그룹, 터미널, 쉘 등으로 접근을 제어한다. /etc/security/time.conf	
pam_wheel.so	su와 관련된 /etc/pam.d/su에서 사용, 특정 그룹에 속하지 않은 사용자는 root로의 접근을 거부, deny 옵션을 사용하여 특정 그룹만 접근을 거부할 수 있게 설정 가능하다.	
pam_cracklib.so	입력 받은 암호를 /usr/lib/cracklib_dict에 있는 디렉토리와 비교하여 새 로운 암호를 /etc/security/opasswd에 저장되어 있는 이전 암호 목록과 비 교하여 이전 암호와 비슷한지 확인.	
pam_succeed_if.so	주어진 조건이 참일 경우 성공 값 반환, 인자로 quiet가 들어갈 경우 syslog에 알리지 않는다.	

5.6 PAM 설정하기

우리가 다룰 설정 파일은 /etc/pam.d 디렉토리에 있다. /etc/security 디렉토리에 있는 특정 모듈에 적용
되는 설정 파일을 변경하려면 모듈과 함께 있는 문서를 확인해야 한다.

/etc/pam.d 디렉토리에 있는 각각 PAM 설정 파일의 형식은 다음과 같다.

[module-type][control-flag][module-path][module-arguments]
--

1) module-type

PAM이 어떤 타입의 인증이 사용될 것인가를 지정한다.

모듈 타입	설명	비고
auth	사용자 인증에 사용하며, 올바른 비밀번호인지 확인하는 절차를 가진다. 응용 프로그램이 사용자에게 비밀번호를 입력하도록 안내하고, 사용자와 해당 사용자의 그룹에 대한 권한을 인증한다.	
account	사용자의 접근 허가 여부를 확인하며 계정 만료, 특정 시간대에 접근이 허용되었는지 여부를 확인한다. 인증하는 기능이 아니며 현재 시간, 사용자의 위치와 같은 다른 요소들의 접근 권한을 결정하는데, 예로 root 사용자의 로그인은 콘솔로만 한다. 이런 식으로 결정하는 것이다.	
password	비밀번호를 설정하고 확인한다. 비밀번호를 기준에 맞게 변경하도록 하는 모듈을 지정한다.	
session	사용자가 인증 받기 전후에 필요한 홈 디렉토리 마운트, 메일박스 생성 등의 유저 섹션을 구분하기 위해 부가적인 작업을 수행한다. 혹시라도 사용자의 로그인 전후에 수행해야 할 작업이 있다는 내용을 지정한다.	

2) contol-flag

PAM에서 사용되는 모듈들이 결과에 따라 어떠한 동작을 취해야 하는지를 지시한다.

flag	설명	비고
required	인증이 거부되기 전에 PAM이 이 서비스에 등록된 모든 모듈들을 요구함에도 불구하고 실패할 경우 인증을 거부하도록 한다. 해당 모듈은 개별 사용자에게 대한 인증을 반드시 진행해야 한다. 인증이 안될 경우 실패가 반드시 반환되어야 한다.	
requisite	이 모듈을 이용하는 인증이 실패할 경우, 즉시 인증을 거부하도록 한다. required와 비슷하지만 이 플래그가 인증을 실패할 경우 설정 파일에서 이 값 다음으로 나오는 모듈들을 호출되지 않는다. 실패한 결과는 즉시 응용 프로그램으로 전달된다.	
sufficient	이전에 요청되었던 모듈이 실패하더라도 이 모듈에 의해서 인증이 성공할 경우 PAM은 인증을 승인한다. 모듈이 성공 값을 반환하고 설정 파일에서 required와 sufficient 제어 플래그가 필요하지 않다면 PAM은 해당 응용 프로그램에서 성공을 반환한다.	
optional	이 모듈이 성공 또는 실패하는지는 그 모듈이 서비스에 대한 형식에 유일한 모듈일 경우에 해당한다. (성공 여부 상관 X) 이 제어 플래그는 모듈의 결과를 무시한다. PAM으로 다른 모듈을 지속적으로 확인하도록 한다. 확인 작업이 실패하게 되더라도 계속 진행하게 된다. 특정 모듈이 실패하더라도 사용자가 로그인하는 것을 허용하고자 할 때 이 플래그를 사용하면 된다.	
include	이 플래그는 인자(argument)에 지정된 또 다른 설정 파일의 내용이나 지침을 포함시키기 위해 사용된다. 즉, 서로 다른 PAM 설정 파일의 내용을 연결하고 구성하는 방식으로 사용된다.	
substack	이 플래그는 다른 PAM 관련 모듈을 불러오는 것은 include와 동일 하지만, substack은 substack의 동작 결과에 따라 나머지 모듈을 처리하지 않는다	

required/requisite 차이는 required와 requisite 인터페이스 모두 반드시 "성공" 되어야만 PAM을 이용한 인증이 완료되나 보안 및 가용성 측면에서는 분명한 차이가 존재한다. required의 경우 실패를 해도 실패한 지점이나 결과값에 대한 리턴을 하지 않는데, requisite의 경우 실패한 지점과 원인을 리턴하기에 보안 관점에서는 공격자에게 인증이 거부된 원인을 제공하게 되며, 가용성 측면에서는 실패한 원인을 리턴하여 원인 분석을 용이하게 해준다.

3) module-path

PAM에게 어떤 모듈을 사용할 것인지 그리고 그것을 어디서 찾을지를 알려준다. (인증 작업을 수행하는 모듈이 있는 실제 디렉토리 경로를 지정해 준다) 대부분 구성은 로그인 구성 파일의 경우 마찬가지로 모듈의 이름만 가지고 있다. 이와 같은 경우, PAM은 기본 PAM 모듈의 디렉토리(/lib64/security 또는 /lib/security) 모듈을 찾는다.

4) module-arguments

모듈에게 전달되는 인수를 나타낸다. 각각의 모듈들은 각각의 인수를 가지고 있다.

arguments	설명	비고
debug	시스템 로그에 디버깅 정보를 남기다.	
no_warn	응용 프로그램에 경고 메시지를 제공하지 않는다.	
use_first_pass	비밀번호를 두 번 확인하지 않는다. 대신 auth 모듈에서 입력한 비밀번호를 사용자 인증 과정 시에도 재사용 해야 한다. (이 옵션은 auth 및 password 모듈에 해당하는 옵션임)	
try_first_pass	이 옵션은 use_first_pass 옵션과 비슷한데, 사용자는 두 번 비밀번호를 입력할 필요가 없기 때문이다. 하지만 기존의 비밀번호를 다시 입력하도록 되어 있다.	
use_mapped_pass	이 인자는 이전 모듈에서 입력된 텍스트 인증 토큰을 입력 받도록 하는데, 이 값으로 암호화 또는 암호화가 해제된 키 값을 생성한다. 그 이유는 모듈에 대한 인증 토큰 값을 안전하게 저장하거나 불러오기 위함이다.	
expose_account	이 값은 모듈로 하여금 계정 정보를 중요하다고 판단하지 않게 한다. 시스템 관리자에 의해 임의로 설정한 것이라 여겨진다.	
nullok	이 인자는 호출된 PAM 모듈이 null 값의 비밀번호를 입력하는 것을 허용한다.	

5.7 PAM 사용예시(일반)

```
#%PAM-1.0
auth    required pam_securetty.so
auth    required pam_unix.so nullok
auth    required pam_nologin.so
account required pam_unix.so
password required pam_cracklib.so retry=3
password required pam_unix.so shadow nullok use_authtok
session required pam_unix.so
```

첫 번째 라인은 주석으로써 인증과는 무관하게 사용하고 있는 PAM 모듈의 버전을 표기하며, auth interface를 사용하는 2~4번째 라인은 해당 서비스에 대한 인증 시 사용된다.

1) auth required pam_securetty.so

pam_securetty.so 모듈의 경우 사용자가 root로 로그인할 경우 /etc/securetty 파일을 확인하여 해당 파일에 리스팅된 터미널에서 로그인을 시도하는 경우 "성공", 기타의 경우 "실패" 값을 리턴한다.

root외 사용자가 로그인을 시도하는 경우 무조건 "성공" 값을 리턴한다. 이 모듈에 대한 테스트의 결과가 "성공"으로 끝나는 경우 다음 auth 라인으로 이동하여 테스트를 진행한다.

2) auth required pam_unix.so nullok

인증 시도 시 /etc/passwd 및 /etc/shadow 파일의 내용을 기반으로 인증 허용여부를 결정한다. 일반적인 password를 통한 인증을 하는 경우, 이 모듈을 사용한다고 보면 된다.

nullok는 password 입력 시 아무것도 입력하지 않는, null password의 입력을 허용한다는 의미로 대부분의 사용자 인증은 이 라인에서 성공/실패 여부가 결정되며 인증이 성공할 경우 다음 auth 라인으로 진행된다.

3) auth required pam_nologin.so

위 예제에서의 인증작업의 마지막 단계에 해당하며, /etc/nologin 파일의 유무를 체크하여 /etc/nologin 파일이 존재하게 되면 시스템(서비스)에서는 root를 제외한 모든 사용자의 접속을 차단하게 되므로 해당 파일이 생성된 경우 root 사용자가 아니라면 인증이 실패하게 된다.

4) account required pam_unix.so

pam_unix.so 모듈의 경우 account interface 사용 시 사용자의 계정이 활성화/비활성화 되었는지 또는 password 사용기간이 만료되지 않았는지 등을 판별한다.

auth interface 모듈에서 인증이 성공한 경우 account interface에서 계정 사용이 가능한지 여부 등을 확인하는 단계이다.

5) password required pam_cracklib.so retry=3

password 사용기간 만료 등의 이유로 password 변경을 하게 되는 경우 pam_cracklib.so 모듈을 통해 작업이 진행된다.

이 모듈에서는 password 변경 시 정해진 규칙에 따라 취약한 password를 사용하는지, 사전에 등록된 알기 쉬운 단어를 password로 사용하는지 등을 체크하여 취약한 password로의 변경 시도 시 해당 변경 작업을 차단한다.

6) password required pam_unix.so shadow nullok use_authtok

PAM을 사용하는 서비스에서 사용자 password를 변경하는 경우 해당되는 내용으로 arguments 중 shadow의 의미는 password 생성/변경 시 shadow password를 사용한다는 것이며, nullok는 null password의 사용을 허용한다는 뜻이다.

7) session required pam_unix.so

사용자가 로그인에 성공하면 사용자 이름, 서비스 타임 등을 로그파일(/var/log/secure) 등에 정상적으로 기록했는지 등을 체크한다.

5.8 PAM 사용예시(/etc/pam.d/su)

```
[root]# cat /etc/pam.d/su
##PAM-1.0
auth            sufficient    pam_rootok.so
# Uncomment the following line to implicitly trust users in the "wheel" group.
#auth          sufficient    pam_wheel.so trust use_uid
```

```
# Uncomment the following line to require a user to be in the "wheel" group.
#auth      required      pam_wheel.so use_uid
auth       include      system-auth
account    sufficient    pam_succeed_if.so uid = 0 use_uid quiet
account    include      system-auth
password   include      system-auth
session    include      system-auth
session    optional     pam_xauth.so
```

시스템 내에서 자주 사용하게 되는 su(Switching User) 명령에 대한 설정으로 su 명령의 경우 root에서 타 사용자로 전환 시 별도의 password를 확인하지 않으며, 일반 사용자에서 일반 사용자 또는 일반 사용자에서 root로의 전환 시에만 변경하고자 하는 계정의 password를 확인한다.

1) auth sufficient pam_rootok.so

사용자 전환 시 root의 경우 password 없이 인증을 통과 시켜준다는 의미로 이 모듈의 control-flag가 sufficient로 설정되어 있어 root가 su 명령을 실행할 경우 별도의 인증절차 없이 "성공" 값이 리턴되며 같은 인터페이스(auth)의 나머지 모듈들을 더이상 실행되지 않는다.

root가 아니면 "실패" 값을 리턴하나 sufficient의 경우 실패는 무시되기에 다음 라인으로 넘어 가게 된다.

2) auth include system-auth

root가 아닌 경우 다음 라인인 이 부분이 실행되며, include flag는 모듈이 아닌 다른 PAM 파일에 대한 결과를 사용한다.

system-auth 파일을 열어보면 auth 인터페이스에서 pam_unix.so 모듈이 사용되는 것을 알 수 있는데, 이 모듈이 사용자로부터 입력 받은 ID/PW를 시스템 내 DB와 비교하여 성공/실패 여부를 리턴하는 인증에 가장 중요한 모듈이다.

3) 기타

su 명령과 관련해서는 기타 인터페이스(account, password, session 등)는 거의 사용되지 않는다.

5.9 PAM 사용예시(/etc/pam.d/system-auth)

```
[root] /etc/pam.d > cat system-auth
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth      required      pam_env.so
auth      required      pam_tally2.so deny=4 unlock_time=1800 no_magic_root reset
auth      sufficient    pam_unix.so nullok try_first_pass
auth      requisite     pam_succeed_if.so uid >= 500 quiet
auth      required      pam_deny.so

account   required      pam_unix.so
account   sufficient    pam_succeed_if.so uid < 500 quiet
account   required      pam_permit.so
```

password	requisite	pam_cracklib.so	try_first_pass	retry=3	type=	minlen=8	lcredit=-1	dcredit=-1	ocredit=-1
password	sufficient	pam_unix.so	md5 shadow nullok	try_first_pass	use_authtok				
password	required	pam_deny.so							
session	optional	pam_keyinit.so	revoke						
session	required	pam_limits.so							
session	[success=1 default=ignore]	pam_succeed_if.so	service in crond quiet	use_uid					
session	required	pam_unix.so							

다양한 PAM 관련 설정파일 중 가장 핵심이라 할 수 있는 파일로 Redhat 기준으로 6.0 이상 버전의 경우 system-auth와 password-auth 두 파일이 존재하는데, 둘 중 하나만 사용되는 것이 아닌, 타 PAM 관련 파일 들마다 참조하는 파일이 다르므로 그때 그때 확인을 해야 한다.

예를 들어, /etc/pam.d/su에서는 인증에 대한 모듈로 system-auth를 참조하며, /etc/pam.d/sshd에서는 인증에 대한 모듈로 password-auth를 참조한다.

1) auth required pam_env.so

pam_env.so 모듈을 사용하여 인증을 확인한다. 성공 시 남은 모듈 확인, 실패 시 남은 모듈을 확인 하지만 접근은 실패(pam_env.so는 설정 파일에서 환경 변수를 불러온다. /etc/environment, /etc/security/pam_env.conf)한다.

2) auth required pam_tally2.so deny=4 unlock_time=1800 no_magic_root reset

pam_tally2.so는 로그인 시도 횟수를 세는 모듈로 일정 횟수 이상 실패 시에는 접근을 차단 및 관리해 주는 역할을 담당한다.

deny=4 : 로그인 시도가 4번 실패하면 추가 시도를 차단함.

unlock_time=1800 : 미리 지정한 일정 횟수 이상 로그인에 실패했을 경우 1800초(30분) 동안 계정이 잠김.

no_magic_root reset : 로그인이 성공한 경우 기존 실패 횟수를 0으로 초기화.

3) auth sufficient pam_unix.so nullok try_first_pass

pam_unix.so는 입력 받은 password 등을 시스템 내에 DB와 비교하여 결과를 판별하는 역할을 수행하며, nullok는 password를 공백으로 입력하는 것을 허용한다는 뜻이다.

try_first_pass는 이전 단계에서 입력 받은 password가 있으면 그걸 사용해서 테스트를 수행하며, 입력 받은 password가 없을 경우 새롭게 password를 입력 받아 사용한다는 의미다.

이 모듈의 경우 sufficient로 flag가 설정되어 있어 테스트 결과가 "성공"으로 확인되면 다음에 위치한 auth requisite pam_succeed_if.so, auth required pam_deny.so에 대해서는 더이상 테스트를 수행하지 않는다.

4) auth required pam_deny.so

pam_deny.so는 테스트 시도에 대해 무조건 "실패" 값을 리턴한다. 즉, auth로 설정된 위 4개 라인에 대해 테스트가 실패할 경우 마지막으로 pam_deny.so에서 인증을 거부하게 된다.

5) auth requisite pam_succeed_if.so uid >= 500 quiet

uid가 500 보다 크거나 같으면 인증을 수행하며, quiet는 로그를 남기지 않는다.

6) password requisite pam_cracklib.so try_first_pass retry=3 type= minlen=8 lcredit=-1 dcredit=-1

ocredit=-1

pam_cracklib.so는 Dictionary에 등록된 단어를 이용한 password 설정 등을 방지하기 위한 비교 및 검사를 수행할 때 사용(password-auth, system-auth 등에 필수적으로 포함됨)한다.

retry=3 : 새로운 password를 설정할 때 물어 보는 횟수를 지정, 변경하고자 하는 password가 기준에 미달 (복잡도, 최소길이 등)할 경우 몇 번의 입력을 추가로 허용할지를 설정.

minlen=8 : password 최소길이를 지정, 글자수와 크레딧(credit) 값이 조합되어 계산됨.

type= : 새로운 password를 입력할 때 화면에 표기되는 문구를 지정, 없으면 기본값.

dcredit=-1 : 숫자가 갖는 기본 크레딧 값을 지정, 기본값은 1이며 -1로 설정된 경우 password에 반드시 숫자가 하나 이상 있어야 한다는 의미.

(ucredit: 대문자, lcredit: 소문자, ocredit: 기타 특수문자)

만약, 크레딧(예, ucredit) 값이 -1이 아닌 2로 지정되었다면 password 최소길이가 8글자로 설정되었을 때 대문자 하나는 2의 값을 갖게됨. 예를 들면 Abcdefg라는 password는 얼핏 보기에 총 길이가 7문자이지만 대문자 A는 2크리딧 값을 갖게 되어, 이 문자열은 8크리딧이 되고 이에 password 최소길이 8글자라는 조건을 충족하게 된다.

5.10 PAM 사용예시(/etc/pam.d/sshd)

SSH 접속을 제한하기 위해 다음과 같이 vi 명령어를 사용하여 다음과 같은 내용을 추가한다.

```
$ vi /etc/pam.d/sshd
auth required pam_listfile.so item=user sense=deny file=/etc/ssh/sshusers onerr=succeed
```

추가 후 /etc/ssh/sshusers 파일을 생성한 후 SSH 접속을 제한할 사용자를 다음과 같이 입력한다. (파일 경로와 파일명은 임의로 설정)

```
$ vi /etc/ssh/sshusers
nuriapp
barokey
baropam
barofds
```

저장 후 해당 계정으로 SSH 접속을 시도할 경우 "Access denied." 메시지를 출력하며 접속이 불가능하다.

5.11 SSH(Secure Shell)

Secure Shell은 개방형 소스 SSH 제품인 OpenSSH 제품을 기반으로 한다. (<http://www.openssh.org>)

Secure Shell은 비보안 네트워크에서 클라이언트와 원격 호스트 간의 보안 연결을 가능하게 한다.

이 보안 연결의 주요 속성은 다음과 같다.

- 클라이언트와 원격 호스트 둘 다에 대한 강력한 인증
- 클라이언트와 원격 호스트 간의 통신에 대한 강력한 암호화 및 공개 키 암호화
- 클라이언트와 원격 호스트에서 명령을 실행하는데 사용할 보안 연결

Secure Shell은 telnet, remsh, rlogin, ftp, 및 rcp와 같은 자주 사용하는 함수와 명령의 보안 대체 항목을 제공한다.

1) Secure Shell의 주요 보안 기능

Secure Shell의 주요 보안 기능은 다음과 같다.

① 강력한 암호화

클라이언트와 원격 호스트 간의 모든 통신은 Blowfish, 3DES, AES 및 arcfour와 같은 특허 없는 암호화 알고리즘을 사용하여 암호화된다. 암호 등의 인증 정보는 네트워크에서 일반 텍스트로 전송되지 않는다. 또한 암호화는 강력한 공개 키 기반 암호화와 더불어 잠재적 보안 공격을 차단한다.

② 강력한 인증

Secure shell은 클라이언트와 서버 간의 강력한 인증 방법 집합을 지원한다. 인증은 양방향일 수 있다. 서버는 클라이언트를 인증하고 클라이언트도 서버를 인증한다. 이렇게 하면 다양한 보안 문제로부터 세션을 보호할 수 있다.

③ 포트 전달

클라이언트와 원격 호스트 간의 TCP/IP 연결 리디렉션(및 그 반대)을 포트 전달 또는 SSH 터널링이라고 한다. Secure Shell은 포트 전달을 지원한다. 예를 들어, 포트 전달을 사용하여 클라이언트와 서버 간의 ftp 트래픽(또는 전자 메일 클라이언트와 POP/IMAP 서버 간의 전자 메일 트래픽)을 리디렉션할 수 있다. 클라이언트가 직접 서버와 통신하는 대신 보안 채널을 통해 트래픽을 sshd 서버로 리디렉션할 수 있으며, 그런 다음 sshd 서버에서 트래픽을 실제 서버 시스템의 지정된 포트로 전달할 수 있다.

2) Secure Shell의 소프트웨어 구성 요소

Secure Shell 소프트웨어는 클라이언트 및 서버 구성 요소로 이루어져 있다.

구성요소	설명	위치	대응하는 비보안 구성요소
ssh	Secure Shell을 클라이언트는 telnet 및 remsh의 보안 대체 항목이며 보안 기능이 있는 remsh와 가장 유사함.	클라이언트	remsh, telnet, rlogin
slogin	ssh에 대한 심볼릭 링크임.	클라이언트	remsh, telnet, rlogin
scp	클라이언트 보안 복사 및 서버 보안 복사를 수행함,	클라이언트 및 서버	rcp
sftp	보안 ftp 클라이언트임	클라이언트	ftp
sshd	보안 Shell 데몬임.	서버	remshd, telnetd, rlogind
sftp-server	보안 ftp 데몬임.	서버	ftpd
ssh-rand-helper	sshd가 서버에서 /dev/random 또는 /dev/urandom을 찾을 수 없을 때는 사용되는 Random Number Generator. OS 커널에 상주하는 Random Number Generator인 rng가 포함되어 있다. rng가 구성 해제되어 있으면 sshd는 prngd를 사용함.	서버	해당 사항 없음
ssh-agent	클라이언트에서 서버로의 “자동” 키 기반 로그인 도구임.	클라이언트 및 서버	Rhosts 파일 메커니즘
ssh-add	클라이언트의 키 쌍을 ssh-agent에 알리는 도구임.	클라이언트	해당 사항 없음
ssh-keygen	공개 키 인증을 위해 키 쌍을 생성하는 도구임.	클라이언트	해당 사항 없음
ssh-keyscan	Secure Shell 데몬(sshd)을 실행하는 호스트 집합에 대한 공개 키를 수집하는 클라이언트 도구임.	클라이언트	해당 사항 없음

ssh-keysign	호스트 기반 인증 중에 필요한 디지털 서명을 생성하는 도구임. ssh()에서 로컬 호스트 키 호스트 기반 인증에 액세스하는데 사용됨.	클라이언트	해당 사항 없음
-------------	--	-------	----------

3) Secure Shell 실행

위 표에서 나열된 Secure Shell 클라이언트를 실행하기 전에 먼저 Secure Shell 서버 데몬 sshd를 시작한다.

sshd 데몬은 서버 시스템의 /etc/ssh 디렉토리에 있는 sshd_config 파일에서 초기 값을 가져온다.

sshd_config에 있는 가장 중요한 구성 지시어 중 하나는 sshd 데몬에서 지원하는 인증 방법 집합이다.

- ssh 클라이언트 실행

ssh 클라이언트 응용 프로그램은 sshd 서버와의 소켓 연결을 설정한다.

sshd 서버는 자식 sshd 프로세스를 시작한다.

이 자식은 연결 소켓을 상속 받고 선택된 인증 방법을 기반으로 클라이언트를 인증한다.

인증에 성공한 경우에만 성공적으로 보안 클라이언트 세션이 설정된다.

세션이 만들어진 후 모든 후속 통신은 클라이언트와 이 자식 sshd 프로세스 간에 직접 이루어진다.

이제 클라이언트는 서버에서 원격 명령을 실행할 수 있다.

ssh 클라이언트의 명령 요청이 있을 때마다 자식 sshd 프로세스에서 해당 명령을 실행할 Shell 프로세스를 시작한다.

간단히 말해서 실행 중인 ssh 클라이언트-서버 세션은 다음 프로세스로 구성된다.

- ① sshd 서버에 연결된 모든 클라이언트 시스템에는 현재 이 클라이언트 시스템에서 설정된 각 ssh 연결에 대한 하나의 ssh 클라이언트 프로세스가 있다.
- ② 서버 시스템에는 하나의 부모 sshd 프로세스와 서버에 연결된 동시 ssh 클라이언트 개수 만큼의 자식 sshd 프로세스가 있다. 서버에 관한 분리가 활성화되어 있으면 서버에서 실행되는 자식 sshd 프로세스의 수가 두 배로 증가한다.
- ③ ssh 클라이언트에서 명령 실행 요청이 있을 때마다 서버 시스템의 해당 자식 sshd 프로세스는 Shell 프로세스를 시작하고 Linux 파이프를 사용하여 명령 요청을 이 Shell 프로세스로 전달한다. 이 Shell 프로세스는 Linux 파이프를 사용하여 명령 실행 결과를 자식 sshd 프로세스로 반환하고 명령 실행이 완료되면 종료된다.

- sftp 클라이언트 실행

sftp 클라이언트 응용 프로그램은 sftp 클라이언트 응용 프로그램이 ssh 클라이언트를 시작 하도록 하고 Linux 파이프를 사용하여 이 클라이언트와 통신한다. 그런 다음 ssh 클라이언트는 sshd 서버와의 소켓 연결을 설정한다.

서버 상호 작용의 나머지 부분은 ssh 클라이언트의 경우와 유사하다. 차이점은 원격 명령을 실행한 Shell 을 시작하는 대신 자식 sshd 프로세스가 sftp-server 프로세스를 시작한다는 것이다. 이 sftp 세션 중의

모든 후속 통신은 다음 프로세스 간에 발생한다.

- ① Linux 파이프를 사용하여 클라이언트 시스템에서 sftp 클라이언트와 ssh 클라이언트 간에
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Linux 파이프를 사용하여 자식 sshd 프로세스와 sftp 서버 프로세스 간에

- scp 클라이언트 실행

scp 클라이언트의 경우는 sftp 클라이언트 실행과 거의 동일하다. 차이점은 sftp-server 프로세스를 시작하는 대신 자식 sshd 프로세스가 scp 프로세스를 시작한다는 것이다. scp 세션 중의 모든 후속 통신은 다음 프로세스간에 발생한다.

- ① 클라이언트 시스템에서 scp 클라이언트와 ssh 클라이언트 간에, Linux 파이프 사용
- ② 설정된 연결 소켓을 통해 ssh 클라이언트와 자식 sshd 프로세스 간에
- ③ Linux 파이프를 사용하여 자식 sshd 프로세스와 scp 서버 프로세스 간에

4) Secure Shell 권한 분리

Secure Shell은 권한 분리 기능을 통해 보다 향상된 수준의 보안을 제공한다. 부모 sshd 및 자식 sshd 프로세스는 권한이 부여된 사용자로 실행된다. 권한 분리를 활성화하면 사용자 연결당 하나의 추가 프로세스가 시작된다.

ssh 클라이언트가 권한 분리에 대해 구성된 sshd 서버에 연결하면 부모 sshd 프로세스가 권한이 부여된 자식 sshd 프로세스를 시작한다. 권한 분리를 활성화하면 자식 sshd 프로세스는 권한이 없는 자식 sshd 프로세스를 추가로 시작한다. 권한이 없는 자식 sshd 프로세스는 연결 소켓을 상속 받는다. 클라이언트와 서버 간의 모든 후속 통신은 권한이 없는 이 자식 sshd 프로세스를 사용하여 이루어진다.

클라이언트의 원격 명령 실행 요청은 대부분 비권한이며 권한이 없는 이 자식 sshd 프로세스에서 시작된 Shell에 의해 처리된다. 권한이 없는 자식 sshd 프로세스에서 권한이 부여된 기능을 실행해야 하는 경우 Linux 파이프를 사용하여 권한이 부여된 부모 sshd 프로세스와 통신한다.

권한 분리는 침입자에 의한 잠재적 손상을 제한하는데 도움이 된다. 예를 들어, Shell 명령을 실행하는 동안 버퍼 오버플로 공격이 발생할 경우 권한이 없는 프로세스 내에서 제어되므로 잠재적 보안 위험이 제한된다.

권한 분리는 Secure Shell의 기본 구성이다. sshd_config 파일에서 "UsePrivilegeSeparation NO" 를 설정하여 권한 분리를 해제할 수 있다. 잠재적 보안 위험이 있으므로 권한 분리를 해제할 경우 신중하게 고려해야 한다.

5) Secure shell 인증

Secure Shell은 다음 인증 방법을 지원한다.

- GSS-API(Kerberos 기반 클라이언트 인증)
- 공개 키 인증
- 호스트 기반 인증
- 암호 인증

클라이언트는 원격 sshd 데몬과 연결될 때 원하는 인증 방법(앞에 나열된 방법 중 하나)을 선택하고 연결

요청의 일부로 적절한 자격 증명을 제공하거나 서버에서 보낸 프롬프트에 응답한다. 모든 인증 방법이 이런 방식으로 작동한다.

서버가 성공적으로 연결을 설정하려면 클라이언트로부터 적절한 키, 암호구, 암호 또는 자격 증명을 받아야 한다.

sshd 인스턴스에서 보안 요구 사항을 기반으로 지원되는 인증 방법 중 일부만 지원하도록 선택할 수 있다.

Secure Shell은 앞에 나열된 인증 방법을 지원하지만 시스템 관리자가 환경의 특정 보안 요구 사항을 기반으로 sshd 인스턴스에서 제공되는 인증 방법을 제한할 수 있다. 예를 들어, Secure Shell 환경에서 모든 클라이언트가 공개 키 또는 Kerberos 방법을 사용하여 인증해야 한다고 지정할 수 있으며, 이로 인해 나머지 방법은 비활성화될 수 있다. 지원되는 인증 방법 활성화 및 비활성화는 sshd_config 파일에 지정된 구성 지시어를 통해 이루어진다.

ssh 클라이언트 연결 요청이 있으면 서버는 먼저 지원되는 인증 방법 목록으로 응답한다. 이 목록은 sshd 서버에서 지원하는 인증 방법과 이러한 방법이 시도되는 시퀀스를 나타낸다. 클라이언트는 이러한 인증 방법을 하나 이상 생략할 수 있다. 클라이언트에서 방법이 시도되는 시퀀스를 변경할 수도 있다. 이렇게 하려면 클라이언트 구성 파일 /etc/ssh/ssh_config의 구성 지시어를 사용한다.

Secure Shell에서 지원하는 인증 방법은 다음과 같이 요약되어 있다.

① GSS-API (Kerberos 기반 클라이언트 인증)

Kerberos 기반 클라이언트 인증인 GSS-API (Generic Security Service application Programming Interface)를 사용하는 경우 클라이언트가 미리 Kerberos 자격 증명을 받아야 하며, 해당 클라이언트 디렉토리에 Kerberos 구성 파일이 있어야 한다.

클라이언트는 sshd 데몬과 연결될 때 연결 시 자격 증명을 제공한다.

서버는 이러한 자격 증명을 이 특정 사용자의 자격 증명 복사본과 일치 시킨다.

또한 선택적으로 클라이언트 호스트 환경의 타당성을 설정할 수 있다.

② 공개 키 인증

공개 키 인증을 사용하려면 Secure Shell 환경에 다음 설정이 있어야 한다..

클라이언트와 서버 둘 다에 키 쌍이 있어야 한다. 모든 ssh 클라이언트와 모든 sshd 서버가 ssh-keygen 유틸리티를 사용하여 자체적으로 키 쌍을 생성해야 한다.

클라이언트는 통신해야 하는 모든 sshd 서버에 해당 공개 키를 알려야 한다. 이렇게 하려면 각 클라이언트의 공개 키를 모든 관련 서버의 미리 지정된 디렉토리에 복사한다.

클라이언트는 통신해야 하는 모든 서버의 공개 키를 구해야 한다. 클라이언트는 ssh-keyscan 유틸리티를 사용하여 공개 키를 받는다.

이 설정이 완료 되면 sshd 서버에 연결하는 ssh 클라이언트가 공개 키와 개인 키를 사용하여 인증된다.

Secure Shell은 공개 키 인증을 단순화하는 추가 기능을 제공한다. 일부 환경에서는 항상 암호 프롬프트에 응답할 필요가 없도록 설정할 수 있다. 둘 다 클라이언트 시스템에서 실행되는 ssh-agent 및 ssh-add 프로세스를 조합해서 사용하면 암호에 응답하지 않아도 된다. 클라이언트는 ssh-add 유틸리티를 통해 ssh-agent 프로세스에 모든 키 정보를 등록한다. 그런 다음 클라이언트와 서버 간의 공개 키 인증은 sshd 데몬이 클라이언트와 상호 작용할 필요 없이 ssh-agent에 의해 수행된다.

③ 호스트 기반 및 공개 키 인증

호스트 기반 및 공개 키 인증은 보다 안전한 공개 키 인증 방법의 확장이다.

클라이언트와 서버 둘 다의 키 쌍이 있는 것 외에도 이 방법을 사용하면 클라이언트 환경에서 통신할 서버를 제한할 수 있다.

클라이언트의 홈 디렉토리에 `.rhosts` 파일을 만들어 이 제한을 구현한다.

④ 암호 인증

암호 인증 방법은 단일 사용자-ID 및 암호 기반 로그인을 사용한다. 이 로그인은 `/etc/passwd`에 지정된 사용자 로그인을 기반으로 하거나 PAM 기반일 수 있다.

Secure Shell은 서버 시스템에서 사용할 수 있는 PAM 모듈과 완전히 통합된다. 이 목적을 위해 `/etc/ssh/sshd_config` 파일에 UsePAM 구성 지시어가 있다. YES로 설정하면 클라이언트에서 암호 인증 요청이 있을 때마다 sshd는 PAM 구성 파일(`/etc/pam.conf`)을 확인한다. 그런 다음 구성된 PAM 모듈을 통해 암호 인증이 성공할 때까지 순서대로 수행한다.

PAM 인증을 무시하려면 UsePAM 지시어를 NO로 설정한다. 그러면 클라이언트에서 암호 인증 요청이 있을 때 sshd가 서버의 PAM 구성 설정을 무시한다. 대신 sshd는 `gwtppnam()` 라이브러리 호출을 직접 호출하여 사용자 암호 정보를 가져온다.

Secure Shell은 PAM_UNIX, PAM_LDAP 및 PAM_KERBEROS를 사용하여 테스트 되었다. 또한 PAM_DCE 및 PAM_NTLM과 같은 다른 PAM 모듈에서 작동한다.

6) 통신 프로토콜

Secure Shell 사용자는 SSH-1 또는 SSH-2 프로토콜을 사용하여 원격 sshd 데몬과 연결할 수 있다. SSH-2가 더 안전하므로 SSH-1 대신 권장한다.

7) Secure Shell에서 사용하는 기능의 일부 목록

Secure Shell은 실제로 Shell이 아니라 호스트에서 안전하게 원격 shell 세션을 실행하기 위해 클라이언트와 원격 호스트 간에 보안 연결을 만드는 매커니즘이다. 보안 연결을 설정하기 위해 Secure Shell에서 인증과 세션 생성을 대부분 수행한다. Secure Shell에서 사용하는 기능의 일부 목록은 다음과 같다.

① login 시도 기록

telnet 또는 remsh와 마찬가지로 Secure Shell은 성공한 세션과 실패한 세션을 각각 `/var/log/wtmp` 및 `/var/log/btmp` 파일에 기록한다.

② PAM 모듈

Secure Shell은 클라이언트 세션에 대해 PAM 인증을 사용할 수 있다. PAM 인증을 선택하면 Secure Shell은 `/etc/pam.conf` 파일을 사용하고 인증을 위해 해당 PAM 모듈을 호출한다.

③ `/etc/default/security` 파일 사용

로그인 동작 암호 및 기타 보안 구성을 정의하는 속성이 들어 있는 시스템 범위 구성 파일이다. 몇 가지 제한은 있지만 Secure Shell에서 이러한 속성을 사용할 수 있다.

④ Shadow passwd

Secure Shell은 Shadow passwd 기능과 통합된다.

⑤ 컨트롤 시스템 로그(syslog)

Secure Shell은 syslog를 사용하여 중요한 메시지를 남긴다.

⑥ 감사 로깅

Secure shell은 해당 코드로 감사 로깅(트러스트된 모드)을 구현했다.

8) 비밀번호 없이 SSH에 접속하는 방법

SSH 접속 시 비밀번호 없이 접속하는 방법은 다음과 같다.

① ssh key 생성 (없는 경우만)

```
$ ssh-keygen -t rsa
```

② 서버로 로컬에서 만든 공개키 복사

```
$ scp ~/.ssh/id_rsa.pub [id]@[address]:id_rsa.pub
```

③ 서버 접속 (일반접속)

```
$ ssh [id]@[address]
```

④ 로그인 후 개인폴더에 .ssh 폴더 생성 (있으면 pass)

```
$ mkdir .ssh
```

⑤ .ssh폴더 권한 변경

```
$ chmod 700 .ssh
```

⑥ 공개키 인증키 목록에 추가

```
$ cat id_rsa.pub >> .ssh/authorized_keys
```

⑦ 필요없는 공개키 제거

```
$ rm -f id_rsa.pub
```

⑧ 인증키 목록 권한 수정

```
$ chmod 644 .ssh/authorized_keys
```

⑨ 서버 접속

```
$ ssh -i ~/.ssh/id_rsa [id]@[address]
```

5.12 PAM 프로그래밍

아래 소스 코드는 PAM 모듈의 기본 형태로 “#if 0” 으로 막혀 있는 부분에 통제를 할 수 있는 코드를 넣

어 두면 원하는 동작(차단 또는 허용, 로그 기록)을 쉽게 처리할 수 있다.

```

/* pam_test.c */

#include <stdio.h>
#include <stdlib.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>

#ifdef PAM_MODULE_ENTRY
PAM_MODULE_ENTRY( "pam_test" );
#endif

#ifndef PAM_EXTERN
#define PAM_EXTERN extern
#endif

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

PAM_EXTERN int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    return PAM_SUCCESS;
}

/* expected hook, this is where custom stuff happens */
PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv) {
    #if 0
        // 아래 조건을 만족하지 못할 경우 로그인 실패로 처리함
        if (check_auth_pam(pamh) < 0)
            return PAM_PERM_DENIED;
    #endif
    return PAM_SUCCESS;
}

```

위에 있는 pam_sm...으로 된 함수들이 사용자가 임의로 추가한 함수가 아니라 PAM에서 기본 함수들로서 해당 함수 부분에 원하는 동작 코드를 넣도록 되어 있다.

PAM 모듈 자체가 기존에 동작하는 telnet이나, ssh, login 등의 바이너리에 동적 라이브러리 형태로 붙기 때문에 위 코드에 삽입한 코드들은 로그인과 동시에 해당 서버로의 접속자의 모든 상황들을 감시하고 제어할 수 있다.

5.13 PAM 컴파일 방법

PAM 인증 모듈은 Linux의 특성별 OS에서 제공하는 최적화 옵션에 따라 컴파일 하는 방법은 다음과 같다.

```
> sudo gcc --std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o baro_auth.o baro_auth.c
> sudo gcc -g -o baro_auth baro_auth.o -ldl
> sudo gcc -g -rdynamic -o baro_auth baro_auth.o -lc -ldl
> sudo gcc --std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o pam_baro_auth.o pam_baro_auth.c
> sudo gcc --std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o base64.o base64.c
> sudo gcc --std=gnu99 -Wall -O2 -g -fPIC -c -fvisibility=hidden -g -fno-strict-aliasing -
I/usr/local/ssl/include -o xxtea.o xxtea.c
> sudo gcc -shared -g -o pam_baro_auth.so pam_baro_auth.o base64.o xxtea.o -lpam -L/usr/lib64 -
lssl -lcrypto -ldl -lz
```

PAM 프로그램 컴파일할 때 반드시 sudo 명령어를 사용하여 컴파일 해야 한다. 그렇지 않으면 다음과 같은 오류가 발생한다.

```
Oct 26 05:16:23 baropam sshd[1452]: [ID 437441 auth.alert] open_module[0:/etc/pam.d/sshd]: module
/usr/baropam/pam_baro_auth.so writable by group
Oct 26 05:16:23 baropam sshd[1452]: [ID 625676 auth.error] load_modules[0:/etc/pam.d/sshd]: can
not open module /usr/baropam/pam_baro_auth.so
```

sudo 명령어는 유닉스 및 유닉스 계열 운영 체제에서 다른 사용자의 보안권한과 관련된 프로그램을 구동할 수 있게 해주는 프로그램이다.

이것은 substitute user do (다른 사용자의 권한으로 명령을 이행하라, 는 뜻이다.) 의 줄임말이다.

기본적으로 Sudo는 사용자 비밀번호를 요구하지만 루트 비밀번호(root password)가 필요할 수 도 있고, 한 터미널에 한번만 입력하고 그 다음부터는 비밀번호가 필요 없다.

Sudo는 각 명령줄에 사용할 수 있으며 일부 상황에서는 관리자 권한을 위한 슈퍼유저 로그인(superuser login)을 완벽히 대신하며, 주로 우분투, 리눅스와 애플의 OS X 에서 볼 수 있다.

PAM을 컴파일하기 위해서는 암호화 모듈에 "openssl" 라이브러리를 사용하므로 반드시 "openssl"을 다음과 같은 명령어로 설치해야 한다.

```
> yum install openssl
```

5.14 PAM 환경설정

보통 Linux의 경우는 /etc/pam.d/ 디렉토리 안에 서비스마다 별도 설정을 해줘야 한다. 단 설정 양식은 아래와 비슷하다.

```
login auth required /usr/lib/security/pam_test.so common
ftp auth required /usr/lib/security/pam_test.so ftp
```

```
sshd auth required /usr/lib/security/pam_test.so ssh
telnet auth required /usr/lib/security/pam_test.so telnet
rsh auth required /usr/lib/security/pam_test.so rsh
rlogin auth required /usr/lib/security/pam_test.so rlogin
dtlogin auth required /usr/lib/security/pam_test.so dtlogin
```

PAM설정이 되어 있는 경우 PAM모듈 프로그램이 잘못되거나 환경설정 파일 수정에 실수가 있는 경우, 그 순간부터 신규 접속은 모두 차단된다.

따라서 반드시 작업 전에 telnet/ssh 창을 접속해 둔 다음 테스트 하시고, 테스트 중에 문제가 발생한 경우 이전에 붙여 놓았던 창을 이용하여 복원해야 한다.

5.15 PAM 디버그 사용

PAM (Pluggable Authentication Modules) 라이브러리는 실행 중에 디버그 정보를 제공 할 수 있다.

시스템이 디버그 출력을 수집하도록 설정한 후에는 수집된 정보를 사용하여 PAM API 호출을 추적하고 현재 PAM 설정에서 오류 지점을 확인할 수 있다.

PAM 디버그 출력을 사용하려면 다음 단계를 완료 해야 한다.

step 1) /etc/syslog.conf 파일을 편집하여 원하는 우선 순위 수준에서 syslog 메시지를 로깅 할 파일을 식별한다.

예를 들어, PAM 디버그 수준 메시지를 /var/log/auth.log 파일에 보내려면 다음 텍스트를 syslog.conf 파일의 새 행으로 추가해야 한다.

```
*.debug /var/log/auth.log
```

step 2) touch 명령을 사용하여 1 단계에서 참조한 출력 파일 (/var/log/auth.log)이 존재하지 않으면 작성해야 한다.

step 3) 구성 변경 사항이 인식되도록 syslogd 데몬을 다시 시작하려면 다음 단계를 완료 해야 한다.

```
> sservice syslog start | stop | restart
```

PAM 응용 프로그램이 다시 시작되면 /etc/syslog.conf 구성 파일에 정의된 출력 파일에 디버그 메시지가 수집된다.

5.16 PAM 테스트 방법

PAM 프로그램 컴파일 또는 PAM 구성 환경 변경 후 sshd를 restart하지 않고 테스트 할 수 있는 방법은 다음과 같이 진행 할 수 있다.

1) sshd 데몬 기동하기

sshd 데몬(22000 포트)을 디버깅 모드로 띄우기 위하여 다음과 같은 명령어를 수행한다.

```
> /usr/sbin/sshd -D -p22000 -d
```

2) ssh로 접속하기

sshd 데몬(22000 포트)에 ssh로 접속하기 위하여 다음과 같은 명령어를 수행한다.

```
> ssh -v -p22000 root@10.0.2.15
```

6. About BaroPAM



Version 1.0 – Official Release – 2016.12.1
Copyright © Nurit corp. All rights reserved.
<http://www.nurit.co.kr>

제 조 사 : 주식회사 누리아이티
등록번호 : 258-87-00901
대표이사 : 이종일
대표전화 : 02-2665-0119(영업문의/기술지원)
이 메 일 : mc529@nurit.co.kr
주 소 : 서울시 강서구 마곡중앙2로 15, 913호(마곡동, 마곡테크노타워2)