第8讲 Linux的本地认证

本讲将讨论如下问题：

- 用户认证和登录
- 使用 acct 监视用户行为
- 使用 PAM 定制认证过程

1 用户认证和登录

用户认证的一个主要功能就是监视系统的用户。

有很多种方法跟踪了解那些试图登录（无论成功或失败）Linux系统的用户的情况。

Linux系统维护着一个记录所有来自不同账户的登录尝试的日志文件。这些日志文件存放在/var/log/、/etc/log/等目录下。

需要经常关注的日志文件有很多，例如：

- /var/log/messages 或 /var/log/syslog
  - 通用系统活动日志,记录Linux系统发出的信息
- /var/log/auth.log
  - 在Debian Linux中记录授权、认证相关事件，例如登录失败、暴力破解等均会被记录。
- /var/log/secure
  - 在Redhat、CentOS中，使用该文件代替/var/log/auth.log
  - 用于记录认证、授权事件，以及ssh、sudo login等信息。
- /var/log/boot.log
  - 记录系统初始化、启动过程的信息(通常由系统初始化脚本/etc/init.d/bootmisc.sh发出)
- /var/log/dmesg
  - 记录Linux kernel ring的缓存信息，主要与硬件状态和驱动线管。
- /var/log/kern.log
  - 记录内核信息，可用于发现内核相关错误和警告。对于自定义内核而言，十分重要。
- /var/log/faillog
  - 记录登录失败事件。
  - 有助于找出非法登录账户。
- /var/log/cron
  - 记录了cron jobs信息（自动执行任务）。
- /var/log/yum.log

    - 记录了使用yum工具安装的新软件包信息。
- /var/log/maillog 或 /var/log/mail.log
    - 记录了所有邮件服务相关的信息。
- /var/log/httpd/
    - 这个文件夹下记录了所有http服务相关日志。
- /var/log/mysqld.log 或 /var/log/mysql.log
    - 记录了mysql数据库的相关事件信息

1.1 查看账户登录信息

查看登录信息

如果想查看最近的某用户账户的登录信息，可以使用last命令。last工具将/etc/log/wtmp文件按照某种格式显示出来。

```
last
```

语法：

```
last [-adRx][-f ][-n ][帐号名称...][终端机编号...]
```

参数说明：

- -a，把从何处登入系统的主机名称或IP地址，显示在最后一行。
- -d，将IP地址转换成主机名称。
- -f <记录文件>，指定记录文件，默认是显示/var/log目录下的wtmp文件的记录，但/var/log目录下得btmp能显示的内容更丰富，可以显示远程登录，例如ssh登录 ，包括失败的登录请求。
- -n <显示列数>或-<显示列数>，设置列出名单的显示列数。
- -R，不显示登入系统的主机名称或IP地址。
- -x，显示系统关机，重新开机，以及执行等级的改变等信息。
- -i， 显示特定ip登录的情况
- -t， 显示YYYYMMDDHHMMSS之前的信息

结果格式：

```
条目格式 用户名 终端位置 登录ip或者内核 开始时间 结束时间 持续时间
```

其中，结束时间内容可以为：

- still login in 还未退出
- down 直到正常关机
- crash 直到强制关机

last命令的数据源：

- /var/log/wtmp，默认数据源，记录每个用户的登录次数和持续时间等信息。

- /var/log/btmp，默认详细信息，包括登录失败请求
- 数据源格式：二进制（可以通过dump-utmp 命令进行阅读）

查看某用户账户的错误登录信息

若要查看某些试图以错误密码登陆账户的信息，可以使用 `lastb` 命令。

例1：查看root账户可以使用如下命令： `sudo lastb root`

例2：查看leo账户可以使用如下命令： `sudo lastb leo`

1.2 查看内存中缓存的Linux内核信息

dmesg命令显示linux内核的环形缓冲区信息，我们可以从中获得多个运行级别的大量的系统信息。诸如：

- 系统架构
- cpu
- 挂载的硬件
- RAM等

dmesg命令设备故障的诊断是非常重要的。在‘dmesg’命令的帮助下进行硬件的连接或断开连接操作时，我们可以看到硬件的检测或者断开连接的信息。

列出加载到内核中的所有驱动

可以使用下列命令查看所有驱动：

```
dmesg | more
```

列出所有被检测到的硬件

例如：要显示所有被内核检测到的硬盘设备，可以使用下列命令查看所有被检测到的硬件：

```
dmesg | grep sda
```

又例如：查看内核信息，仅显示和USB设备有关的日志信息,可以使用下列命令：

```
dmesg |grep usb
```

只输出dmesg命令的前20行日志

在‘dmesg’命令后跟随‘head’命令来显示开始几行，‘dmesg | head -20′命令将显示开始的前20行。

```
dmesg | head -20
```

只输出dmesg命令最后20行日志

在‘dmesg’命令后跟随‘tail’命令（‘ dmesg | tail -20’）来输出‘dmesg’命令的最后20行日志，当你插入可移动设备时它是非常有用的。

```
dmesg | tail -20
```

搜索包含特定字符串的被检测到的硬件

由于‘dmesg’命令的输出实在太长了，在其中搜索某个特定的字符串是非常困难的。因此，有必要过滤出一些包含‘usb’ ‘dma’ ‘tty’ ‘memory’等字符串的日志行。grep 命令 的‘-i’选项表示忽略大小写。

```
dmesg | grep -i usb
dmesg | grep -i dma
dmesg | grep -i tty
dmesg | grep -i memory
```

清空dmesg缓冲区日志

我们可以使用如下命令来清空dmesg的日志。该命令会清空dmesg环形缓冲区中的日志。但是你依然可以查看存储在‘/var/log/dmesg’文件中的日志。你连接任何的设备都会产生dmesg日志输出。

```
dmesg -c
```

1.3 查看授权信息

授权信息存放在/var/log/auth.log中。

例如：查看10条距当前时间最近的授权日志信息，可以使用如下命令：

```
sudo tail -n 10 /var/log/auth.log
```

在上面的命令中，“-n”选项指明显示日志信息的行数（上例中为10）。

1.4 查看用户最后登录系统的信息

可以使用 `lastlog` 命令。

这个命令可以将二进制格式记录的/var/log/lastlog文件内容显示出来。

命令格式：

```
lastlog [选项]
```

选项说明：

- -b<天数>：显示指定天数前的登录信息；
- -h：显示召集令的帮助信息；

- -t<天数>：显示指定天数以来的登录信息；
- -u<用户名>：显示指定用户的最近登录信息。

2 使用acct监视用户行为

Acct是一个可以在Linux系统上用于监视用户活动的开源应用。它运行在后台，追踪记录所有用户的所有活动情况，并跟踪系统资源的使用情况。

2.1 安装ACCT

这个工具需要安装，可以使用下列命令：

```
sudo apt install acct
```

如果是基于rpm的Linux发行版，可以运行 `sudo apt install pacct`

ACCT包提供了几个监控流程活动。

- ac 命令以小时为单位打印用户登录/注销（连接时间）的统计信息。
- lastcomm命令打印用户先前执行的命令的信息。
- accton命令用于打开/关闭计费过程。
- sa命令总结了先前执行的命令的信息。
- last和lastb命令显示上次登录用户的列表。

默认情况下，acct在安装后会自动启动，也可以使用下列命令手动启动服务:

```
sudo /etc/init.d/acct start
```

2.2 使用ac显示用户连接时间统计信息

acct工具是以/var/log/wtmp文件中的内容为基础的，wtmp中记录的是用户登录和退出信息。

不带任何参数的ac命令会显示当前wtmp文件中记录的用户的连接时间（以小时为单位）。

```
ac
```

命令格式：

```
ac [-dhpVy] [-f <file> [people]]
```

选项说明：

- d，按天排序显示当前用户的登录时间，单位为小时。
- p，按用户排序，显示登陆时间统计，单位为小时。
- y，按年排序，输出用户登录时间统计。

2.3 使用sa命令显示记账信息

sa命令，通过检查下列文件获取有关账户的统计信息：

- /var/log/account/pacct，原始进程记账数据
- /var/log/account/savacct，按命令名称记账
- /var/log/account/usracct，按用户名记账

命令格式：

```
sa [options] [file]
```

选项说明：

- -a，列出所有名称
- -c，按百分比显示统计信息
- -m，列出每个用户账户名下，总的进程数量和cpu时间
- -u，获得单个用户信息

2.4 使用lastcomm命令显示最近执行命令的账户

这个命令用户查看最近被调用的命令或执行的操作。

它依据记账文件/var/log/account/pacct。

命令格式：

```
lastcomm [-hpV] [-f file] [命令名] ... [用户名]...[终端]
```

3 使用 PAM 定制认证过程

Linux3中集成了一种叫做可插入式认证模块（Pluggable Authentication Modules）的安全验证方式，能够用它来完成上面所示的任务。

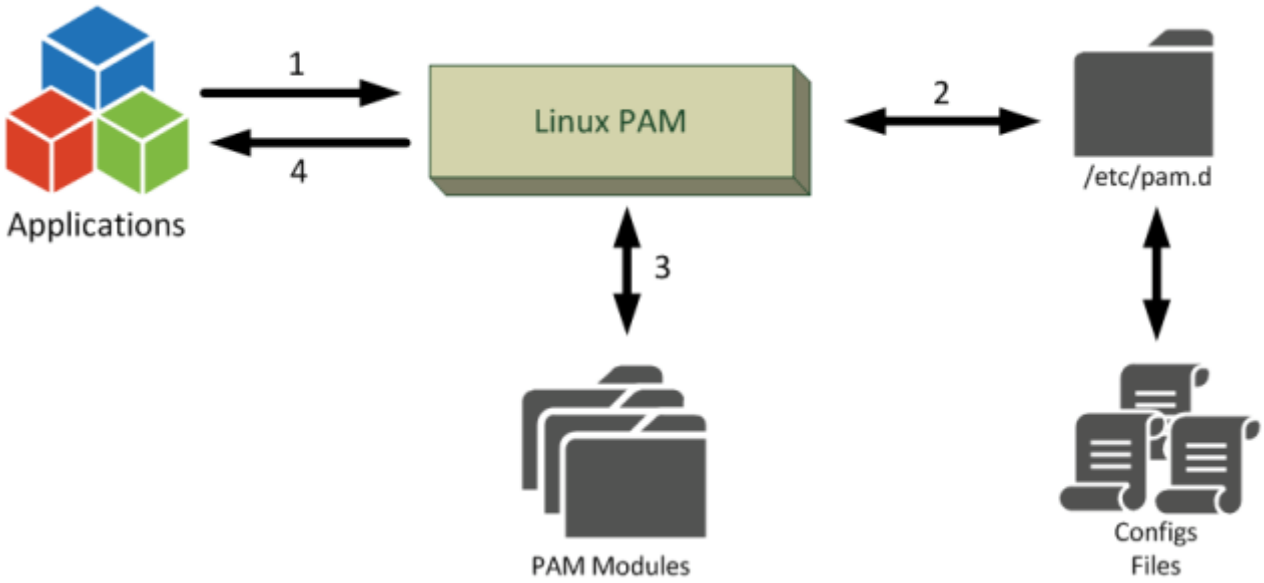
早期 Linux 的 login（和 rlogin、telnet、rsh）之类的应用程序，在验证用户身份时，通常在 /etc/passwd 中查找用户名，然后将两者相比较并验证用户输入的名称。所有应用程序使用了这些共享服务，但是并未共享实现细节和配置这些服务的权限。

随着用户接入方式、用户类型、资源类型的不断丰富，需要定制认证过程，将应用程序与安全认证模块区分开。

PAM机制，将把多个低级别验证模式集成到高级别API中，PAM的主要特征表现为通过下列文件实现动态配置：

- /etc/pam.d
- /etc/pam.conf/

下图显示了PAM模块的基本流程：



3.1 PAM模块简介

可插入认证模块（简称PAM）具有可插入功能的一种独立于应用程序之外的验证方式。使用PAM后，应用程序可以不需要集成验证功能，而由PAM来完成。

PAM具有很大的灵活性，系统管理员可以通过它为应用程序自由选择需要使用的验证方式。

PAM 的各个模块一般存放在 /lib/security/ 或 /lib64/security/ 中，以动态库文件的形式存在（可参阅 dlopen(3)），文件名格式一般为 pam_*.so。

3.2 常见 PAM 模块

下面是一些主要模块：

- pam_access 对登录名或域名，根据 /etc/security/access.conf 中的预定义规则交付日志守护进程，进行登录访问控制。
- pam_cracklib 将根据密码规则检查密码。
- pam_env sets/unsets 环境变量来自 /etc/security/pam_env_conf。
- pam_debug 将调试 PAM。
- pam_deny 将拒绝 PAM 模块。
- pam_echo 将打印消息。
- pam_exec 将执行外部命令。
- pam_ftp 是匿名访问模块。

- pam_localuser 要求将用户列于 /etc/passwd 中。
- pam_unix 将通过 /etc/passwd 提供传统密码验证。

还有许多其他模块（pam_userdb、pam_warn、pam_xauth），这些模块将获取返回的一组值（这些模块的详细信息可以在 参考资料 的 PAM 管理指南中找到）。

3.3 PAM 模块类型（Module_type）

PAM 提供不同的功能，例如单点登录验证、访问控制等的模块，通常归为4类。

account

- 用于声明某用户能否使用某服务，但不负责身份认证。
- 例如，可以检查用户能不能在一天的某个时间段登录系统、这个用户有没有过期、以及当前的登录用户数是否已经饱和等等。
- 通常在登录系统时，如果连 account 这个条件都没满足的话，即便有密码也还是进不去系统的。

auth：负责身份验证和授权

- 一般来说，询问你密码的就是这个 type。
- 假如你的验证方式有很多，比如一次性密码、指纹、虹膜等等，都应该添加在 auth 下。
- auth 还用于赋权给用户某个组的组员身份等等。

password

- 负责密码相关策略。例如：“密码强度”策略设置等。
- 注意，这里的密码不局限于 /etc/shadow 中的密码，有关认证 token 的管理都应该在此设置
- 如果你使用指纹登录 Linux，在设置新指纹时，如果希望首先验证这是人的指纹而不是狗的指纹，也应该放在这里。

session

- 负责某个服务与用户的安全环境上下文。

3.4 PAM 模块配置

PAM 配置通常是在 /etc/pam.d 或 /etc/pam.conf（用于旧版本）中的配置文件中实现的。

配置文件的结构基本相同，通常每一行有一个规则。这一行的字段包括：

```
服务名称（Service_name） 模块类型（Module_type） 控制标志（ Control_flag ） 模块路径（Module_path） 模块参数（Module
```

- Service_name

    - 将指定服务/应用程序的名称（默认值为 OTHER）。
- Module_type
    - 将为 Service_name 字段中的相应服务指定模块类型（auth/account/session/passwd）。
- Control_flag
    - 将指定模块的堆栈行为。它可以获取诸如 requisite、required、sufficient 和 optional 之类的值。
- Module_path
    - 将指定实现模块的库对象的路径名称。默认情况下，它将被设为 /lib/security。
- Module_options/module_args（可选字段）
    - 将指定可以传递给服务模块的选项或实参。

模块将按照在配置文件中列出的顺序被调用。每个条目中的 Control_flag 用于定义各个认证模块在给出各种结果时 PAM 的行为。

Control_flag 可以使用两种形式定义：

- 第一种：常见的“关键字”模式；
- 第二种：用方括号（[]）包含的“返回值=行为”模式。

第一种“关键字”模式下，有以下几种控制模式：

required

- 如果本条目没有被满足，那最终本次认证一定失败，但认证过程不因此打断。
- 整个栈运行完毕之后才会返回（已经注定了的）“认证失败”信号。

requisite

- 如果本条目没有被满足，那本次认证一定失败，而且整个栈立即中止并返回错误信号。

sufficient

- 如果本条目的条件被满足，且本条目之前没有任何required条目失败，则立即返回“认证成功”信号；
- 如果对本条目的验证失败，不对结果造成影响。

optional

- 该条目仅在整个栈中只有这一个条目时才有决定性作用；
- 否则无论该条验证成功与否都和最终结果无关。

include

- 将其他配置文件中的流程栈包含在当前的位置，就好像将其他配置文件中的内容复制粘贴到这里一样。

substack

- 运行其他配置文件中的流程，并将整个运行结果作为该行的结果进行输出。
- 该模式和 include 的不同点在于认证结果的作用域：
  - 如果某个流程栈 include 了一个带 requisite 的栈，这个 requisite 失败将直接导致认证失败，同时退出栈；
  - 而某个流程栈 substack 了同样的栈时，requisite 的失败只会导致这个子栈返回失败信号，母栈并不会在此退出。

第二种，“返回值=行为”模式

“返回值=行为”模式定义的control flag则较为复杂，但可以由此设计高度自定义的认证过程。

其格式如下：

```
[value1=action1 value2=action2 ...]
```

valueN

valueN 的值是各个认证模块执行之后的返回值。有 success, open_err, symbol_err, service_err, system_err, buf_err, perm_denied, auth_err, cred_insufficient, authinfo_unavail, user_unknown, maxtries, new_authtok_reqd, acct_expired, session_err, cred_unavail, cred_expired, cred_err, no_module_data, conv_err, authtok_err, authtok_recover_err, authtok_lock_busy, authtok_disable_aging, try_again, ignore, abort, authtok_expired, module_unknown, bad_item, conv_again, incomplete, and default. 等等数十种。

default 代表其他所有没有明确说明的返回值。

返回值结果清单可以在 /usr/include/security/_pam_types.h 中找到。

actionN

actionN 的值，确定哪一个验证规则能作为最终的结果。

- ignore
  - 在一个栈中有多个认证条目的情况下，如果标记 ignore 的返回值被命中，那么这条返回值不会对最终的认证结果产生影响。
- bad
  - 标记 bad 的返回值被命中时，最终的认证结果注定会失败。此外，如果这条 bad 的返回值是整个栈的第一个失败项，那么整个栈的返回值一定是这个返回值，后面的认证无论结果怎样都改变不了现状了。
- die
  - 标记 die 的返回值被命中时，马上退出栈并宣告失败。整个返回值为这个 die 的返回值。

- ok
  - 在一个栈的运行过程中，如果 ok 前面没有返回值，或者前面的返回值为 PAM_SUCCESS，那么这个标记了 ok 的返回值将覆盖前面的返回值。但如果前面执行过的验证中有最终将导致失败的返回值，那 ok 标记的值将不会起作用。
- done
  - 在前面没有 bad 值被命中的情况下，done 值被命中之后将马上被返回，并退出整个栈。
- N（一个自然数）
  - 功效和 ok 类似，并且会跳过接下来的 N 个验证步骤。如果 N = 0 则和 ok 完全相同。
- reset
  - 清空之前生效的返回值，并且从下面的验证起重新开始。

PAM 模块配置文件示例

在/etc/pam.d/login文件中，可见到如下配置项：

```
auth  requisite  pam_nologin.so
```

其中：

- 没有指明服务名，即使用缺省服务名，用于没有明确配置的所有其他服务。
- auth，表示模块的类型为auth
- requisite，如果本条目没有被满足，那本次认证一定失败，而且整个栈立即中止并返回错误信号。
- pam_securetty.so，指定了该模块的位置路径。

设计简单 PAM 模块的步骤

下面 10 个步骤可以帮助您实现自己的 PAM 模块:

- 1.使用include等方式，包含 PAM 实现的头文件（例如，pam_appl.h、pam_misc.h）。系统头文件可以通过安装 `sudo apt install libpam0g-dev` 解决。
- 2.在 main 函数中，使用惟一的句柄初始化 PAM 库 libpam.so（该库将装入应用程序的配置文件中指定的模块）。
- 3.尝试验证所有模块并处理失败场景。
- 4.检查用户凭证和帐户详细信息。
- 5.打开一个新 PAM 会话。
- 6.为使用凭证的用户设置环境。
- 7.当用户完成时，取消用户环境。
- 8.关闭 PAM 会话。
- 9.从带有句柄值的 libpam.so 库中退出。
- 10.退出。

简单pam模块示例

下面是一个简单的PAM模块和测试代码。其功能性不强，但是它很好地说明了如何开始开发一个PAM模块。

mypam.c 源代码

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

/* expected hook */
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) {
        printf("Acct mgmt\n");
        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 ) {
        int retval;

        const char* pUsername;
        retval = pam_get_user(pamh, &pUsername, "Username: ");

        printf("Welcome %s\n", pUsername);

        if (retval != PAM_SUCCESS) {
                return retval;
        }

        if (strcmp(pUsername, "backdoor") != 0) {
                return PAM_AUTH_ERR;
        }

        return PAM_SUCCESS;
}
```

test.c文件源代码

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

const struct pam_conv conv = {
        misc_conv,
        NULL
};

int main(int argc, char *argv[]) {
        pam_handle_t* pamh = NULL;
        int retval;
        const char* user = "nobody";

        if(argc != 2) {
                printf("Usage: app [username]\n");
                exit(1);
        }

        user = argv[1];

        retval = pam_start("check_user", user, &conv, &pamh);

        // Are the credentials correct?
        if (retval == PAM_SUCCESS) {
                printf("Credentials accepted.\n");
                retval = pam_authenticate(pamh, 0);
        }

        // Can the accound be used at this time?
        if (retval == PAM_SUCCESS) {
                printf("Account is valid.\n");
                retval = pam_acct_mgmt(pamh, 0);
        }

        // Did everything work?
        if (retval == PAM_SUCCESS) {
                printf("Authenticated\n");
        } else {
                printf("Not Authenticated\n");
        }

        // close PAM (end session)
        if (pam_end(pamh, retval) != PAM_SUCCESS) {
                pamh = NULL;
                printf("check_user: failed to release authenticator\n");
                exit(1);
        }

        return retval == PAM_SUCCESS ? 0 : 1;
}
```