一、什么是Linux-PAM

为安全起见，计算机系统只有经过授权的合法用户才能访问，在这里如何正确鉴别用户的真实身份是一个关键的问题。所谓用户鉴别，就是用户向系统以一种安全的方式提交自己的\*\*\*明，然后由系统确认用户的身份是否属实的过程。换句话说，用户鉴别是系统的门户，每个用户进入到系统中都必须经过鉴别这一道关。

最初，Linux系统的用户鉴别过程就像各种Unix系统 的一样：系统管理员为用户建立一个帐号并为其指定一个口令，用户用此指定的口令登录后重新设置自己的口令，这样用户就具有了一个只有他自己知道的秘密口 令。一般情况下，用户的口令经过加密处理后存放于 /etc/passwd 文件中。用户登录时，登录服务程序提示用户输入其用户名和口令，然后将口令加密并与 /etc/passwd 文件中对应帐号的加密口令进行比较，如果口令相匹配，说明用户的身份属实并允许此用户访问系统。这种思想基于只有用户自己知道他的口令，所以输入的口令是 正确的话，那么系统就认定他是所声称的那个人。

后来，还采用了许多其他的鉴别用户的方法，如用于网络环境的 Kerberos 以及基于智能卡的鉴别系统等。但是这些鉴别方案有一个通病：实现鉴别功能的代码通常作为应用程序的一部分而一起编译,这样问题就来了------如果发现 所用算法存在某些缺陷或想采用另一种鉴别方法时，用户不得不重写（修改或替换）然后重新编译原程序。很明显，我们原先的鉴别方案缺乏灵活性，这里的牵一发 而动全身的情形很是让人恼火。

鉴于以上原因，人们开始寻找一种更佳的替代方案：一方面，将鉴别功能从应用中独立出来，单独 进行模块化设计，实现和维护；另一方面，为这些鉴别模块建立标准 API，以便各应用程序能方便的使用它们提供的各种功能；同时，鉴别机制对其上层用户（包括应用程序和最终用户）是透明的。直到 1995 年，SUN 的研究人员提出了一种满足以上需求的方案--插件式鉴别模块（PAM）机制并首次在其操作系统 Solaris 2.3 上部分实现。插件式鉴别模块（PAM）机制采用模块化设计和插件功能，使得我们可以轻易地在应用程序中插入新的鉴别模块或替换原先的组件,而不必对应用程 序做任何修改，从而使软件的定制、维持和升级更加轻松--因为鉴别机制与应用程序之间相对独立。应用程序可以通过 PAM API 方便的使用 PAM 提供的各种鉴别功能，而不必了解太多的底层细节。此外，PAM的易用性也较强，主要表现在它对上层屏蔽了鉴别的具体细节，所以用户不必被迫学习各种各样的 鉴别方式，也不必记住多个口令；又由于它实现了多鉴别机制的集成问题，所以单个程序可以轻易集成多种鉴别机制如 Kerberos 鉴别机制和 Diffie - Hellman 鉴别机制等，但用户仍可以用同一个口令登录而感觉不到采取了各种不同鉴别方法。

在广大 开发人员的努力下，各版本的 UNIX 系统陆续提供对 PAM 的支持。其中，Linux-PAM 是专门为 Linux 机器实现的，包括 Caldera 1.3、2.2、Debian 2.2、Turbo Linux 3.6、Red Hat 5.0 以及 SuSE 6.2 及它们的后续版本都提供对 PAM 的支持。FreeBSD 从 3.1 版开始支持 PAM。需要注意的是，除了具体实现不同外，各种版本 Unix 系统上的 PAM 的框架是相同的，所以我们在这里介绍的 Linux-PAM 框架知识具有普遍性。因此在下文介绍其框架的过程中可以看到，我们并没有刻意区分 PAM 与 Linux－PAM 这两个术语。

二、PAM 的分层体系结构

PAM 为了实现其插件功能和易用性，它采取了分层设计思想：让各鉴别模块从应用程序中独立出来，然后通过PAM API作为两者联系的纽带，这样应用程序就可以根据需要灵活地在其中"插入"所需鉴别功能模块，从而真正实现了"鉴别功能，随需应变"。实际上，这一思路 非常符合软件设计中的"高内聚，低耦合"这一重要思想，PAM 的体系如下简图所示：

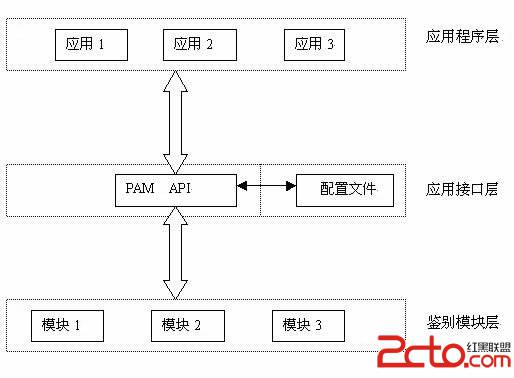


图1 PAM体系结构

从 上图可以看出，PAM API 起着承上启下的作用，它是应用程序和鉴别模块之间联系的纽带：当应用程序调用 PAM API 时,应用接口层按照配置文件 pam.conf 的规定，加载相应的鉴别模块。然后把请求（即从应用程序那里得到的参数）传递给底层的鉴别模块,这时鉴别模块就可以根据要求执行具体的鉴别操作了。当鉴别 模块执行完相应操作后，将结果返回给应用接口层，然后由接口层根据配置的具体情况将来自鉴别模块的应答返回给应用程序。

上面描述了 PAM 的各个组成部分，以及它们作为整体的运作机理。下面将对 PAM 的关键的低二层分别加以介绍。

三、第一层：模块层

模 块层处于整个结构的最底层，它向上为接口层提供用户鉴别等服务，也就是说所有具体的鉴别工作都是由该层的模块来完成的。对于应用程序，有些不但需要验证用 户的口令，还可能要求验证用户的帐户是否已经过期。此外,有些应用程序也许还会要求记录当前会话的有关信息或改变口令等，所以 PAM 在模块层除了提供鉴别模块外，同时提供了支持帐户管理、会话管理以及口令管理功能的模块。当然,这四种模块并不是所有应用程序所必需的,而是根据需要灵活 取舍,比如虽然 login 可能要求访问所有四种模块,但是 su 可能仅仅需要使用鉴别组件即可。至于如何取舍则涉及到接口层的 PAM API和配置文件,这部分内容将在下文中加以介绍。

四、第二层：应用接口层

应用接口层位于 PAM 结构的中间部分，它向上为应用程序屏蔽了用户鉴别等过程的具体细节，向下调用模块层中的具体模块所提供的特定服务。由图1可以看出，它主要由 PAM API 和配置文件两部分组成，下面将逐一介绍。

PAM API 可以分为两类，一类是用于调用下层特定模块的接口，这类接口与底层的模块相对应：

1. 鉴别类接口：pam\_authenticate（）用于鉴别用户，pam\_setcred（）用于修改用户的秘密信息。

2. 帐号类接口：pam\_acct\_mgmt（）检查受鉴别的用户所持帐户是否有权登陆系统，以及该帐户是否已过期等。

3. 会话类接口：包括用于会话管理和记帐的 pam\_open\_session（）和 pam\_close\_session（）函数。

4. 口令类接口：包括用于修改用户口令的 pam\_chauthtok（）。

第二类接口通常并不与底层模块一一对应，它们的作用是对底层模块提供支持以及实现应用程序与模块之间的通信等。具体如下：

1. 管理性接口

每组 PAM 事务从 pam\_start（）开始，结束于 pam\_end（）函数。接口 pam\_get\_item（）和 pam\_set\_item（）用来读写与 PAM 事务有关的状态信息。同时，能够用 pam\_str()输出 PAM 接口的出错信息。

2. 应用程序与模块间的通讯接口

在 应用程序初始化期间,某些诸如用户名之类的数据可以通过 pam\_start（）将其存放在PAM接口层中，以备将来底层模块使用。另外，底层模块还可以使用 pam\_putenv()向应用程序传递特定的环境变量，然后应用程序利用 pam\_getenv() 和 pam\_getenvlist() 读取这些变量。

3. 用户与模块间的通讯接口

pam\_start（）函数可以通过会话式的回调函数，让底层模块通过它们读写模块相关的鉴别信息，比如以应用程序所规定的方式提示用户输入口令。

4. 模块间通讯接口

尽管各模块是独立的,但是他们仍然能够通过 pam\_get\_item（）和 pam\_set\_item（）接口共享某些与鉴别会话有关的公用信息，诸如用户名、服务名、口令等。此外，这些API还可以用于在调用 pam\_start（）之后，让应用程序修改状态信息。

5. 读写模块状态信息的接口

接口 pam\_get\_data（）和 pam\_set\_data（）用以按照PAM句柄要求访问和更新特定模块的信息。此外，还可以在这些模块后附加一个清除数据函数，以便当调用 pam\_end（）时清除现场。

由于 PAM 模块随需加载,所以各模块始化任务在第一次调用时完成。如果某些模块的清除任务必须在鉴别会话结束时完成,则它们应该使用 pam\_set\_data（）规定清除函数，这些执行清除任务的函数将在应用程序调用 pam\_end（）接口时被调用。

五、配置文件

我们注意到，配置文件也放在了在应用接口层中，它与 PAM API 配合使用，从而达到了在应用中灵活插入所需鉴别模块的目的。它的作用主要是为应用选定具体的鉴别模块，模块间的组合以及规定模块的行为。下面是一个示例配置文件：

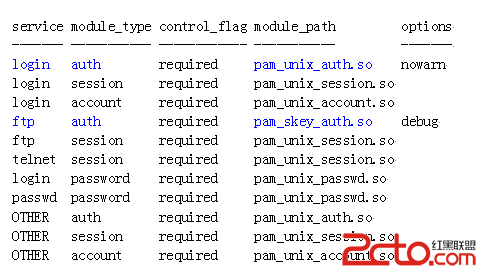


图2.示例配置文件

我们可以看到,配置文件有许多登记项(每行对应一个登记项)组成,每一行又分为五列(每列对应一栏),详细解释如下:

第 一栏，service表示使用PAM的应用程序，比如login、passwd、rlogin等。这一栏中的OTHER表示所有没在该文件中显式列出的应 用。也就是说，如果所有程序具有相同的需求，整个配置文件只需要一行即可，并且该行的第一栏为OTHER。本例中，因为所有应用程序使用相同的会话模块, 所以实际上可以用单行，即

" OTHER auth required pam\_unix\_auth.so"

来代替文件中的这些行：

"login session required pam\_unix\_session.so

ftp session required pam\_unix\_session.so

telnet session required pam\_unix\_session.so"。

第 二栏,module\_type指明程序所用PAM底层模块的类型：auth表示鉴别类模块；account表示帐户类模块；session表示会话类模 块；password表示口令类模块。注意，每行只能指定一种类型模块，如果程序需要多种模块的话，可在多行中分别规定。

第 三栏,control\_flag规定如何处理模块的成功和失败情况。单个应用程序可以调用多种底层模块，这通常称为"堆叠"，对应于某程序的按照配置文件 中出现顺序执行的所有模块成为"堆"，堆中的各模块的地位与出错时的处理由control\_flag栏的取值决定，它的五种可能的取值分别为 required、Requisite、sufficient或\_optional，现介绍如下：

required--它 表示该模块的成功是用户通过鉴别的必要条件，换句话说，只有当对应于应用程序的所有带required标记的模块全部成功后，该程序才能通过鉴别。同时， 如果任何带required标记的模块出现了错误，PAM并不立刻将错误消息返回给应用程序，而是在所有模块都调用完毕后才将错误消息返回调用它的程序。

Requisite--它与required相仿，只有带此标记的模块返回成功后，用户才能通过鉴别，不同之处在于其一旦失败就不再执行堆中后面的其它模块，并且鉴别过程到此结束。

optional--它表示即便该模块失败，用户仍能通过鉴别。在PAM体系中，带有该标记的模块失败后将继续处理下一模块。

sufficient-- 它表示该模块取得成功是用户通过鉴别的充分条件,也就是说只要标记为sufficient的模块一旦成功，那么PAM便立即向应用程序返回成功而不必尝试 任何其他模块。当标记为sufficient的模块失败时，sufficient模块当做optional对待。

第四栏,module\_path指出PAM模块的位置。

第五栏,options用于向特定模块传递相关的选项，然后由模块分析解释这些任选项。比如使用此栏打开模块调试，或向某模块传递诸如超时值之类的参数等。另外，它还用于支持下文所述的口令映射技术。

如果任一栏出现错误,或某模块没有找到,那么所在行被忽略并将其作为严重错误进行记录。

本例中，login程序使用UNIX口令模块进行鉴别，而ftp程序却使用S/Key模块进行鉴别。如我们想改变ftp程序的鉴别方法，比如也用UNIX口令模块进行鉴别，那么我们不必改动源程序，只需将配置文件中的

ftp auth required pam\_skey\_auth.so debug

改为

ftp auth required pam\_unix\_auth.so debug

这 样，当用户使用ftp时，将用传统的UNIX口令鉴别方式来验证其身份。由此可见，在PAM体制下为应用程序改变鉴别机制是一件轻松的事情。另外，PAM 体制的堆叠功能还使得应用程序能够支持多种鉴别机制，如下例中的login程序在配置文件中先后出现了三项与鉴别有关的登记项：

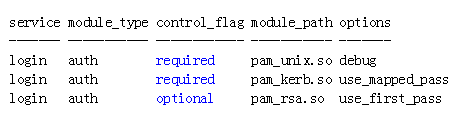


图3.示例配置文件

当 login程序执行时先用pam\_unix.so模块即传统的UNIX口令方式鉴别用户，然后再调用pam\_kerb.so模块即 Kerberos对用户进行鉴别，最后用pam\_rsa.so模块即RSA方式鉴别用户。在按上述顺序鉴别用户的过程中，如果pam\_unix.so模块 鉴别失败，它将继续调用下面的模块进行鉴别而非立刻向login程序返回错误消息；pam\_kerb.so模块也按同样方式处理，直到顺序处理完最后一个 pam\_rsa.so模块后，PAM才将前面出现的错误信息返回给login程序。对于该配置，即使pam\_rsa.so模块顺利通过，只要 pam\_unix.so模块和pam\_kerb.so模块中有一个出现错误，用户就不能通过鉴别；相反，即使pam\_rsa.so模块失败，只要 pam\_unix.so模块和pam\_kerb.so模块都通过了，用户也能通过鉴别。

六、口令映射

在 同一个机器上使用多个鉴别机制，尤其是一个应用程序集成多种鉴别机制可能导致用户需要记忆多个口令，这会让用户觉得很不舒服。虽然可以让所有机制使用相同 的口令来获取易用性,但是这将削弱系统的安全性--如果其中任何一个机制的口令泄露了，则所有机制都会受到牵累。此外,不同的鉴别机制在口令长度、容许的 字符、更新间隔、有效期等方面可能具有他们特有的要求，这些要求也是为多鉴别机制使用同一个口令必须考虑的一个问题。

PAM 为我们提供了这样一种解决方案,它不排除为所有鉴别机制共用一个口令,同时允许通过口令映射技术让每个机制使用不同的口令。该方案用用户的" 主口令"加密其他的"副口令",并且将这些经过加密的副口令存放在一个用户能访问的地方。主口令一旦经过验证,鉴别模块就能用它解密那些加密的副口令从而 获得相应口令,然后将所需口令传递给鉴别模块。这称为"口令映射"。如果口令映射出现错误,或如果映射不存在,那么各鉴别模块应该提示用户输入口令。为支 持口令映射,主口令应保存在PAM第二层并且在需要时由其提供给堆叠的各个鉴别模块。同时，口令要在pam\_authenticate函数返回之前清除。 为了保障口令映射的安全，主口令必须足够强壮,可以考虑使其的长度更长、组成口令字符的类型多样化并使用混合类型的字符组成口令等有效措施。

口令如何加密及其存储完全取决于具体的实现：它能够将加密的副口令（也称作"映射口令"）存储在可靠或不可靠的地方,诸如智能卡、本地文件或目录服务。当然，如果加密的口令保存在一个不可靠的允许公共访问的地方,会留下受到字典攻击的隐患。

为实现口令映射，所有鉴别模块应支持以下四个映射选项∶

1. use\_first\_pass∶它表示当该模块执行时不提示用户输入口令，而将该模块之前的提示用户输入的主口令作为它们的公共口令进行验证。如果用户没 能通过主口令的鉴别，则该模块不提示用户输入口令。此选项一般说来在系统管理员想强制用同一个口令通过多模块时使用。

2. try\_first\_pass∶除了如果主口令不正确，将提示用户输入口令之外，它的用法与use\_first\_pass相同。

3. use\_mapped\_pass∶它表示使用口令映射技术得到此模块的有效口令。也就是说，该模块执行时不提示用户输入口令，而是用映射口令即用主口令解 密得到的该模块的副口令作为本模块的口令输入进行验证。如果在此之前用户没能通过主口令的鉴别，则该模块也不会提示输入口令。

4. try\_mapped\_pass∶除了如果主口令不正确，它将提示用户输入口令之外，该项与use\_mapped\_pass用法相同。

当口令更换后,PAM会保存所有新旧口令，并且使有关模块能够访问到它们。其他模块能够使用此信息更新加密的口令而不必强制用户再次输入口令。

现以下面的配置文件为例讲解口令映射：

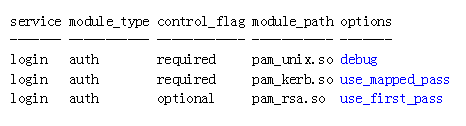


图4. 示例配置文件

 在 这里login程序集成了三种鉴别方式：传统UNIX口令鉴别、Kerberos和RSA鉴别，但通常情况下用户仅输入一次口令便能通过鉴别了。当程序调 用pam\_unix.so模块时，PAM提示用户输入他们的UNIX口令，然后由pam\_kerb.so模块对用户输入的UNIX口令进行鉴别。继而调用 pam\_kerb.so模块，由于该模块的选项为use\_mapped\_pass，它将利用口令映射机制进行认证，也就是说，如果UNIX口令鉴别通过的 话，就将其作为pam\_kerb.so模块的主口令来解密其对应的映射口令从而进行Kerberos鉴别。如果pam\_unix.so模块所需口令没能通 过验证，则无法进行口令映射，那么PAM将直接调用下一鉴别模块而不提示用户输入其Kerberos口令。最后一个模块的选项为 use\_first\_pass，所以pam\_rsa.so模块将使用前边输入的主口令来鉴别用户，如果口令错误也不提示用户输入RSA口令。所以，只要第 一次输入的口令是正确的，并且映射口令存在，则一个口令便足以通过鉴别。

七、结束语

Linux-PAM是一种使用灵活功能强大的用户鉴别机制，本文对它的组成结构以及各部分之间的关系进行了相应的分析，希望对读者理解PAM的机制有所帮助。