Polythene PAM ain't what she used to be...

Martin Schwenke

IBM OzLabs Linux Technology Center

<martins@au.ibm.com> · <martin@meltin.net>

Abstract

This paper discusses the author's recent experiences with Pluggable Authentication Modules (PAM) under Linux, although most of the discussion applies to other PAM-enabled operating systems. Attempts to mix users defined in local files with users defined in an LDAP directory, and implement a defensive system administration policy, did not entirely succeed. The discussion covers the Name Service Switch (NSS) (and associated libc functionality), PAM, LDAP and user credentials, and concludes that some major changes are necessary to provide an authentication and credentials system that is reliable enough for mission critical systems.

Prelude: Polythene Pam

...
She's the kind of a girl that makes the "News of the World"
Yes, you could say she was attractively built
Yeah, yeah, yeah...
— Lennon/McCartney (© 1969 Northern Songs)

1 Introduction

The Pluggable Authentication Modules (PAM) framework was first suggested in around 1995 as a way of letting Unix system administrators configure a range of authentication technologies for a range of services, in a modular way. PAM was originally implemented on Solaris, though PAM support for Linux followed shortly afterwards. The original papers about PAM [2, 1] described an elegant system, which had the desirable feature that services could be considered orthogonal to authentication technologies.

Prior to this, with the introduction of Solaris 2.0 in the early 1990s, Sun had introduced another feature called the Name Service Switch (NSS) to support NIS+ [3]. NSS allows standard Unix user, group, host and other information to be gathered from arbitrarily specified back-end. The simplest such back-end is `files` comprising `/etc/passwd`, `/etc/shadow`, `/etc/group`, `/etc/hosts` and other standard Unix system files. Other, more elaborate back-ends, include LDAP directories, NIS+ maps and SQL databases.

For the most part PAM and NSS play together well, via the `pam_unix` module (described in more detail later), but when they don't the results can be confusing, if not totally unsatisfactory. §2 provides an example of an attempt to implement a particular defensive system administration policy. §3 outlines how this policy should be implemented, explains why it doesn't work and discusses several possible workarounds, each with their own problems. §4 discusses some underlying problems in and around PAM and some possible solutions are discussed in §5.

Note this paper is not really about security. It is about being able to implement system administration policy.

2 My Problem: Locally and Remotely Defined Users

My computer system has two groups of users:

- those defined in `/etc/{passwd,shadow,group}`; and
- those defined in an LDAP directory.

I want members of both groups to be able to login to my system. I want these logins to be as reliable as possible.

2.1 The Risk

More generally, my system has two groups of users:

- those defined in local files; and
- those defined on a remote system.

Even more generally, my system has two groups of users:

- those defined in locals file, in a standard, well-known format, that is accessed by incredibly well tested code; and
- those defined in a third-party application, in a format that I don't really understand, accessed via several layers of networking software, from a remote system that is connected to my system via cables and various pieces of networking equipment.

If any of the vaguely defined components in the second point fails, the users defined in my LDAP directory can not login. That's a risk I have to take — I'm trading some reliability for the convenience of centralised administration.

2.2 Insurance Policy

Now consider that I *need* to login as `root` on my system to resolve an issue that is making my system unusable. `root` is a locally defined user and is the only user that really gives me full control over my system. To login as `root` there is no theoretical reason why I *need* to invoke any of the software or cables that deal with my remotely defined users. I don't want to introduce that level of unnecessary complexity to something as critical as logging in to my `root` account — it's not a risk I'm willing to take.

Therefore, it is my policy that none of the configuration for remotely defined users should be considered when I login as a locally defined user. I wish to configure my system so that, when a locally defined user logs in, the system exhibits the same behaviour as when there are no remotely defined users.

2.3 Why Am I Telling You This?

The type of configuration provided as an example with the `pam_ldap` package is shown below.

```
...
auth       sufficient   pam_ldap.so
auth       required     pam_unix.so use_first_pass
account    sufficient   pam_ldap.so
account    required     pam_unix.so
session    required     pam_unix.so
...
```

This configuration attempts to authenticate users and do account and session management via LDAP before looking in `/etc/{passwd,shadow}`. This violates my policy. Many `pam_ldap` proponents argue that:

- when the LDAP server is available, the `pam_ldap` steps take almost no time; and
- timeouts on the LDAP client-side can be adjusted so that when the LDAP server is unavailable the inconvenience can be minimised.

However, a bug in `pam_ldap` or the LDAP client libraries that causes a `SIGSEGV` will not allow the calling application to continue to authenticate any users, including the `root` user. This is one of the reasons why I have my policy.

3 The Obvious Solution

The obvious solution to the above problem is to put `pam_unix` before `pam_ldap` and make it `sufficient`, as follows:

```
...
auth       sufficient   pam_unix.so
auth       required     pam_ldap.so use_first_pass
account    sufficient   pam_unix.so
account    required     pam_ldap.so
session    required     pam_unix.so
...
```

This solution does not work. Moreover, with this configuration (and in fact *any* PAM configuration!) the policy set down in §2.2 is still violated in an extremely subtle way.

3.1 It Doesn't Work?

One piece of the above configuration that doesn't work is the account management. There are two chunks of account management that need to be done:

- Password expiry checks, by looking at the various `shadow` fields.
- A host access check, done by comparing the hostname of the machine to the values stored in the `host` attribute of a user's LDAP `account` object.

The first of these is done by both `pam_unix` and `pam_ldap`, which seems to be an unnecessary duplication of work. This duplication probably stems from the recommendation that `pam_ldap` be listed first and be marked as `sufficient`. The second is done only by `pam_ldap`, which makes sense because `pam_unix` has no way of knowing about the `host` attribute.

If `pam_unix`'s account management decides that the given user's password has not expired, it returns `PAM_SUCCESS`. Since this step is marked `sufficient`, `pam_ldap`'s account management is never run. Therefore, the host access check is never done.

So, why does `pam_unix`'s account management succeed for LDAP users and what can be done about it?

3.2 What is a Unix User?

The search for enlightenment begins by asking a very simple question:

> What is a Unix user?

The answer to this is quite simple: a Unix user is one that is defined in a back-end that is available to NSS. For example, to use local files and then LDAP to lookup user and group information, the relevant entries in `/etc/nsswitch.conf` look like this:

```
passwd: files ldap
group:  files ldap
shadow: files ldap
```

For now, there are two things to note:

- NSS uses system-wide configuration. It can not be configured on a per-application basis. However, it is possible to customise the behaviour of certain back-ends.
- When searching for an entry given a user or group name, if the entry is found in `files` it is returned, and `ldap` is never consulted.

3.2.1 How Does PAM Handle Unix Users?

The deceptively simple answer here is: PAM handles Unix users via the `pam_unix` module, of course! This is true to an extent. `pam_unix` can handle any type of back-end from which NSS can be configured to *retrieve user information that includes the password field*. What's more, in certain contexts, `pam_unix` thinks that it can handle any type of back-end from which NSS can be configured

to *retrieve user information*. That's quite a subtle distinction, especially since `pam_unix` doesn't check!

The first category includes standard Unix password systems, shadow password systems and NIS+. It does not include systems that can't or won't provide the password field, perhaps because they don't want to be quite so trusting. LDAP is one such system. To check if a username/password pair can be used to authenticate a user, the client must attempt to *bind* to an LDAP directory using the username/password pair. This operation can be performed without having to trust the client. However, in certain contexts, `pam_unix` thinks it can handle users in LDAP because passwords aren't relevant in those contexts.

So, this is why `pam_unix`'s account management succeeds for users defined in LDAP. `pam_unix` can see all of the relevant information via NSS, so it simply goes through the motions.

3.3 Workarounds

One possible workaround is to make `pam_unix` be `required`:

```
...
account    required     pam_unix.so
account    required     pam_ldap.so
...
```

This means that the `pam_ldap` account management is done for users defined in LDAP. However, it is also attempted for locally defined users, since PAM has no reason to stop after `pam_unix` succeeds. There are two problems with this:

- it violates the policy set down in §2.2; and
- locally defined users will be unable to login since `pam_ldap`'s account management will fail.

3.4 Working Around the Workaround

Luckily, someone thought about this and organised a workaround in `pam_ldap`. It looks like this:

```
...
account    required     pam_unix.so
account    required     pam_ldap.so   ignore_unknown_user
...
```

This makes `pam_ldap`'s account management return `PAM_IGNORE` if the user is not defined in LDAP, which keeps PAM happy. What's more, this can be done without the module in question having to workaround the 'unknown user' situation. Welcome to PAM's 'pretty baroque stuff in square brackets':[1]

```
...
account  required     pam_unix.so
account  [default=die success=ok authinfo_unavail=ignore \
          user_unknown=ignore] pam_ldap.so
...
```

[1] This is how Andrew Morgan, the Linux PAM maintainer, described this notation on the PAM mailing list [4] on 1997-08-04, nearly six months before it was actually introduced.

OK, that's much clearer! What? There's still a problem? Oh yeah! It still violates the policy set down in §2.2...

3.5 A Working Workaround

Here is a workaround[2] that doesn't violate the policy set down in §2.2 (in any obvious way):

```
account    requisite    pam_unix.so
account    sufficient   pam_localuser.so
account    required     pam_ldap.so
```

`pam_localuser` is a non-standard module[3] that doesn't do any account management as such, but simply checks if the given user is locally defined via `/etc/passwd`. This allows PAM's account management processing to be short-circuited before `pam_ldap` is even considered. This appears to meet the desired policy.

One downside of this approach is that it may require two complete traversals of `/etc/passwd` (one by `pam_unix` and one by `pam_localuser`). If `/etc/passwd` is large, this might take a long time. Another downside is that it is illogical and counterintuitive — in the six months between 2001-11 and 2002-04 about half a dozen people asked[4] why their host access check wasn't working and needed to be told about the workaround. This doesn't include those who:

- still don't know they have a problem;
- didn't know they had a problem, but implemented the workaround when they saw it posted;
- knew they had a problem and saw the answer posted before they asked; or
- trawled through the mailing list archives to find the answer.

4 Underlying Problems

The previous section described an example problem and some workarounds. This section points out some underlying problems in PAM (or outside of PAM, as will soon be explained).

4.1 `pam_unix`

NSS is used to make user information available to the operating system and, therefore, defines the idea of a Unix user. `pam_unix` attempts to build on top

[2] This workaround appears to have been initially suggested by Nalin Dahyabhai <`nalin@redhat.com`> on the `pamldap` mailing list [5] on 2001-11-15. His version had `pam_unix` listed as `required`. Paul Hilchey <`hilchey@ucs.ubc.ca`> posted an updated version of the workaround on 2002-04-05, with `pam_unix` listed as `requisite`. This is a good improvement, since you really do expect that step to succeed.

[3] `pam_localuser` is distributed with Red Hat Linux, but is not yet part of the Linux PAM distribution.

[4] On `pamldap` [5] and `pam-list` [4].

of NSS and tries to cope with all of the users that NSS makes visible. However, §3 has shown a case where it is necessary to know when a user is locally defined because the apparent generality provided by `pam_unix` is confusing. It is interesting that `pam_unix` tries to be all things for all users in a *pluggable* authentication system.

One service where `pam_unix` can not be all things to all users is the `password` service. `pam_unix` has built-in knowledge that allows it to change standard Unix passwords, shadow passwords and NIS+ passwords. `pam_unix` was obviously built with these three back-ends in mind and shows its limitations when mixed with other back-ends.

4.2 `initgroups(3)`

Even with careful configuration and judicious use of `pam_localuser`, a system that uses LDAP in NSS, will still need to timeout if the LDAP server is unavailable, and will crash if the client software contains a serious bug. This is because, after authenticating, an application that wishes to assume the full Unix credentials of the authenticated user must call `initgroups(3)` to initialise the supplementary group access list. `initgroups(3)` queries each back-end configured in NSS for a list of supplementary groups that contain the user.

This violates the policy stated in §2.2. However, this particular violation is very close to unavoidable.

4.3 Pretty Baroque Stuff in Square Brackets

The square brackets configuration notation, while perhaps a necessary evil in some circumstances, seems to promote configurations that are convenient but not necessarily good.

5 Workarounds and Solutions

5.1 `pam_unix`

There are two ways of making `pam_unix` more usable.

5.1.1 Split `pam_unix`

Recognise that at the end of the day `pam_unix` really knows about two different types of users: locally defined (`/etc/{passwd,shadow}`) and remotely defined (NIS+). Split `pam_unix` into `pam_files` and `pam_nisplus`, leaving two much simpler modules. This would help to make PAM more *pluggable*.

5.1.2 Further Complicate `pam_unix`

There are two options here:

- Add a `localfiles` option to `pam_unix`. When activated, this would make `pam_unix` check for the desired user in `/etc/passwd` at key points in its logic and return `PAM_USER_UNKNOWN` if the user can't be found.

- Make `pam_unix` able to cope with additional types of back-ends that may be configured in NSS. SuSE Linux's `pam_unix2` takes this approach by using other modules like `pam_ldap`.

Further complicating `pam_unix` seems like a mistake, since some of its current problems stem from its complexity.

5.2 `initgroups(3)`

As noted in §4.2, PAM isn't responsible for setting up the supplementary group access list. PAM also isn't responsible for setting up the the userid or primary primary group membership. In general, PAM is not responsible for establishing a user's operating system credentials on behalf of an application. This is left to the application, using whatever means the operating system provides. This means that any application that wishes to establish Unix-like credentials must call functions like `setgid(2)`, `initgroups(3)` and `setuid(2)` using information retrieved via NSS. NetBSD at least makes all of this easy by bundling all of these calls into a function called `setusercontext(3)` although, since this function is still called by the application, it still doesn't help to make credentials establishment any more pluggable.

The `pam_group` module allows an application to set extra groups via the `pam_sm_setcred` function. Other modules such as `pam_krb` use this function to set other back-end-specific credentials. However, `pam_sm_setcred` should not be used to set the basic operating system credentials.

Why don't PAM modules set operating system credentials, especially given the presence of `pam_sm_setcred`? The original PAM RFC [2] doesn't make this clear, but makes vague references to GSS-API — although GSS-API is a client/server authentication mechanism and clearly doesn't perform a credentials establishment role on current platforms that use PAM. The example code in the appendices of the PAM RFC performs `setgid(2)`, `initgroups(3)` and `setuid(2)` calls.

On 1997-03-03, Andrew Morgan wrote on `pamlist` [4]:

> Credentials include things like (Kerberos) tickets. The natural extension of this is to make the setuid and initgroups calls part of this scheme, however Sun have ruled that these two things are actually in the domain of the application code.

Therefore, it looks to have been a decision made by Sun, the inventors of PAM. This decision has been regularly questioned on `pamlist` [4], but for now, it remains part of the 'standard'.

On 2002-06-27, Norbert Klasen suggested [5] the following workaround for the `initgroups(3)` problem when it involves `nss_ldap`:

> If local (system) users need not be members of groups held in ldap, one might introduce a "nss_min_uid" option [...] For example, if uid<100 then nss_ldap would skip the group lookup in ldap.

However, this was rejected by Luke Howard, author of `pam_ldap` and `nss_ldap`, who explained that `initgroups(3)` takes a username, not a uid as its argument,

so it would still need to lookup the uid in LDAP.[5]

Workarounds aside, I think Sam Hartman, the Debian GNU/Linux PAM maintainer, summed up the situation quite well in his 2002-05-14 post to the PAM mailing list [4]:

> Long term, I think having PAM evolve to handle credentials establishment would be a net good; [...]
>
> Of course when you take things to their logical conclusion, PAM would be responsible both for the setuid call *and* initgroups; I think doing one without the other would be wrong.
>
> Getting to that ideal world would be very difficult; I think the PAM upstream, libc upstream and application writers would all disagree with us. We'd also need to think carefully about the API and potentially change things and better define things such that PAM could actually be responsible for user credential management. But hey if anyone ever wants to fight that battle, I'm certainly interested in helping.

Epilogue

Well you should see Polythene Pam
She's so good-looking but she looks like a man
Well you should see her in drag dressed in her polythene bag
Yes, you should see Polythene Pam
Yeah, yeah, yeah...

Get a dose of her in jackboots and kilt
She's killer-diller when she's dressed to the hilt
She's the kind of a girl that makes the "News of the World"
Yes, you could say she was attractively built
Yeah, yeah, yeah... — Lennon/McCartney (© 1969 Northern Songs)

6 Conclusions

Despite the title of this paper, PAM hasn't changed very much, apart from that 'pretty baroque stuff in square brackets'. However, the world that PAM lives in has changed quite a bit, placing more varied demands on PAM. PAM has started to look a little left behind and dated. The biggest improvement would be to somehow integrate credentials establishment into PAM, so that too could be pluggable. It might be time for PAM to put away the polythene bag and try on the jackboots and kilt...

[5] I must respond to Luke and ask whether the initgroups(3) implementation in `nss_ldap` would really go directly to LDAP to lookup the the uid for the given username . Surely the username should be looked up using `getpwnam(3)`, but perhaps that isn't allowed inside the implementation of another NSS function?

7 Thanks...

Many thanks to:

- Sam Hartman and Steve Langasek for useful discussions on `pam-list` [4].
- Stephen Rothwell and David Gibson for useful discussions about PAM, agreeing with some of the things I said and proofreading drafts of this paper.
- Melynda McDonald for being there while I wrote yet another paper.

References

[1] Vipin Samar and Charlie Lai.
Making Login Services Independent of Authentication Technologies.
Sunsoft, Inc.. An earlier version of this paper was presented at the 3rd ACM Conference on Computer and Communications Security, March, 1996.

[2] V. Samar and R. Schemers.
Unified Login with Pluggable Authentication Modules (PAM).
Open Software Foundation, Request For Comments: 86.0, October 1995.
`http://www.opengroup.org/tech/rfc/rfc86.0.html`

[3] Sun Microsystems.
Network Information Service Plus(NIS+): An Enterprise Naming Service.
1992.
`http://wwws.sun.com/software/whitepapers/wp-nisplus/`
`http://www.nrao.edu/computing/sol2/NISPlus-Admin-WP.ps`.

[4] PAM mailing list <`pam-list@redhat.com`>. Archived at <`https://listman.redhat.com/mailman/private/pam-list/`> (list subscribers only).

[5] `pam_ldap` mailing list <`pamldap@padl.com`>. Archived at <`http://www.netsys.com/pamldap/`>.