Table of Contents

PAM Cross-Platform Reference

Pluggable Authentication Modules for Linux and UNIX Systems

Includes Bonus Content for Users of Novell Identity Manager

A publication from OmniBond Systems, trusted developers of Novell® software, including a comprehensive range of Identity Manager drivers for most distributions of Linux® and UNIX®, z/OS® and IBM® Power Systems running i, AIX and Linux.

PAM Cross Platform Reference

Table of Contents

(Click any line)

1.0 OVERVIEW

1.1 *IMPORTANT* – How to use this manual

Although this document provides a comprehensive view of PAM and how it relates to Novell Identity Manager, it does not need to be read from start to finish. It is organized for quick reference to the specific issues you want to read about.

For optimal navigation, use Search (**Ctrl-F** on your keyboard) or the hyperlinked Table of Contents.

Following are descriptions of the major sections in the manual:

Section	Title	Description
2	Before PAM	How limitations in the original design of UNIX authentication set the stage for the creation of PAM.
3	Understanding PAM	An overview of the PAM framework, with conceptual illustrations and a breakdown of the syntax that will enable you to begin tweaking your PAM configuration file(s).
4	PAM and the Identity Manager Bidirectional Driver	How the bidirectional driver works passively to monitor and communicate password changes made in PAM. Also provides guidelines for customizing PAM beyond the bidirectional driver's default configuration.
5	PAM and the Identity Manager Fan-Out Driver	How the Fan-Out Driver uses local components (Platform Services) on each system it integrates with Identity Manager. When a system is running Linux or UNIX, the local components include a custom PAM module.
6	Troubleshooting	Steps to take for resolving common PAM issues related to Identity Manager.
7	Past Support Solutions	A "cookbook" presentation of issues and their solutions garnered from past technical support calls with Identity Manager customers.
8	Additional Information	Further background and history on PAM, as well as a list of other resources to consult about PAM and Identity Manager.
9	Glossary	Definitions of key terms used throughout this manual.

1.2 PAM in a nutshell

PAM, which stands for Pluggable Authentication Modules, is the defacto standard for system-entry control in Linux and UNIX.

Originally, user authentication requirements for UNIX-based systems were programmed into their actual system-entry applications, such as `login`, `su`, `ftp` and `telnet`. As a result, these applications were subject to ongoing changes to meet the ever-growing needs of system security.

PAM put an end to this by providing a one-off interface that all system-entry applications could defer to for precise decision-making. While those applications did require extra iterations to work with this new interface, once they were "PAM-enabled," all ongoing changes and enhancements could take place in PAM.

By offering modules that plug in to the PAM framework, any developer now can add customized levels of security in compiled programming code that executes in tandem with any of the standard system-entry applications.

Today, PAM is built in to most all UNIX and Linux server distributions. PAM has always been supported by Novell Identity Manager.

1.3 Novell Identity Manager

One of Identity Manager's well known strengths is its robust support of Linux and UNIX platforms. Novell offers two driver designs, bidirectional and Fan-Out, enabling Linux and UNIX customers to choose the most optimal method for connecting their systems to an Identity Manager solution.

From the beginning, Identity Manager developers recognized the need for their Linux and UNIX drivers to include support for PAM. As the Identity Manager customer base has grown, a base of knowledge on the effective use of PAM with Identity Manager has also formed.

The goal of this manual is to share that knowledge base with you.

2.0 BEFORE PAM

The original design of the UNIX operating system used a simplistic approach to security. This eventually made it difficult to keep pace with the ever-expanding scope of security and network computing. Necessity led to the invention of PAM.

2.1 The significance of /etc/passwd

On traditional Linux and UNIX systems, authentication information is stored in `/etc/passwd`, a text file that contains a user's login, encrypted password, unique numerical user id (uid), numerical group id (gid), optional comment field, home directory, and preferred shell. Each field is separated by a colon.

```
      Parts:  login  :encyr_password :uid   :gid  :comment,,,             :home_dir    :pref_shell
   Examples:  bill   :Q3ycO1Sjx8LFM  :1000  :1000 :Bill Smith,,,555-1212  :/home/bill :/bin/bash
How entered:  bill:Q3ycO1Sjx8LFM:1000:1000:Bill Smith,,,555-1212:/home/bill:/bin/bash
```

Since its early beginnings, `/etc/passwd` had a flaw: it's not secure; any user on the system can read it. In the early years of UNIX, encrypted passwords were considered plenty tough, but it didn't take long for the proliferation of password-cracking programs to change that.

So when you look at the `/etc/passwd` file on your system today, a sample line looks more like this:

```
bill:x:1000:1000:Bill Smith,,,555-1212:/home/bill:/bin/bash
```

Instead of an encrypted password, you now have an `x`.

2.2 Adding /etc/shadow to the security equation

To address the security issues caused by an open-to-the-public `/etc/passwd` file, developers created the option to use *shadow* passwords.

When a system has shadow passwords enabled, the password field in `/etc/passwd` is replaced by an `x` and the user's real encrypted password is stored in `/etc/shadow`, a more secure file that only the root user can read. Each entry in `/etc/shadow` contains a user's login, encrypted password, and a number of fields relating to password expiration. Here's an example:

```
bill:/4HKllb2o4152:12009:0:99999:7:::
```

2.3 MD5 encrypted passwords

Traditionally, Linux and UNIX passwords have been encrypted by a standard function called `crypt()`. Over time, `crypt()` has become easier to crack, leading to newer encryption alternatives. Many distributions include the option to encrypt passwords with the MD5 hash algorithm (RFC 1321). While even MD5 passwords are not invincible, they are much more difficult to break.

2.4 Along came PAM

The sum-total of these pieced-together components still did not add up to a central, secure authentication solution for Linux and UNIX systems. System-entry applications, such as `login` and `su`, had to be hard-coded to work with each of these options, none of which was a completely satisfactory solution. Perhaps more significantly, the growth of networking demanded the ability for systems to refer to a central repository of user account information rather than scores of local, individual, `/etc/passwd` files.

A new standard was needed to pull it all together and provide more granular control for authentication. Her name was PAM.

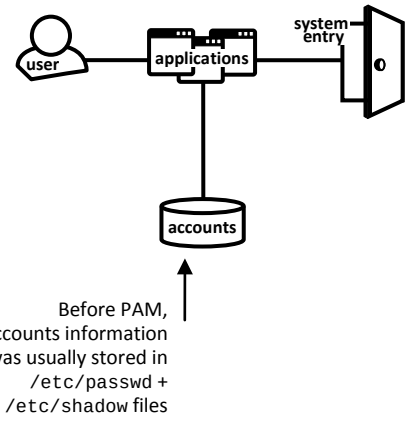
3.0 UNDERSTANDING PAM

3.1 Review

As discussed in the previous section, users who want access to a Linux or UNIX host must offer their credentials through a system-entry application, such as `login`, `ssh`, `ftp`, or `telnet`. Before there was PAM, each of these applications was designed to look for the appropriate authentication information in `/etc/passwd` and `etc/shadow/`.

While more sophisticated alternatives to `/etc/passwd` and `/etc/shadow/` are available today, including open specifications for databases and directories, they were difficult to implement before PAM.

Figure 1 System-entry without PAM.



applications
Any application that enables a user to request entry to the Linux or UNIX host system.

Examples: `passwd`, `ssh`, `ftp`, `telnet`

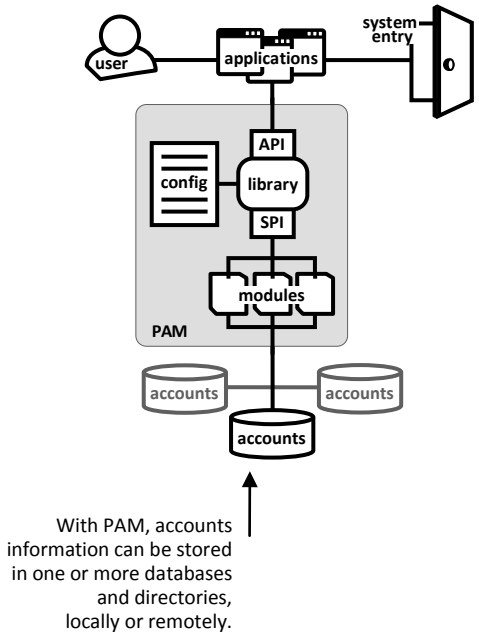
accounts
Repository of all details about the host system's accounts. Before PAM, this was usually simple text files: `/etc/passwd` and `/etc/shadow`

Today it also can be a database or directory.
Examples: eDirectory, Network Information System (NIS), NIS+, LDAP

system-entry (door)
Conceptual point at which authentication and authorization is successful, allowing entry to the system.

The solution shown in Figure 1 causes problems. It requires each system-entry application to be "retooled" whenever changes are made to the design of the host system, the accounts repository, or the functions being requested. PAM eliminates these hurdles by requiring system-entry applications to only interface with the PAM framework. PAM handles the rest with a flexible design that allows all parties to add their own requirements for system-entry via "pluggable" software modules (see Figure 2).

Figure 2 Adding the PAM Framework.



API
(Application Programming Interface) A group of PAM library functions that enable the system-entry application to communicate with the library.

library
All the software functions and subroutines necessary for the PAM framework to operate.

SPI
(Service Provider Interface) A group of PAM library functions that enable the library to communicate with the PAM modules.

modules
Software instructions stored in shared library files. Up to four modules can exist in each shared library—one for each of the four areas PAM can manage (authentication, account access, password change, and session management).

config
A single file or group of files containing simple rules about which modules should be invoked for each system-entry application. The system administrator uses this file to set system-entry requirements.

3.2 The PAM framework

The PAM framework includes five main parts:

- A **library** of programming functions and subroutines to carry out all framework tasks, including:
  - A group of functions that make up the **API** (applicaton programming interface), which enables system-entry applications to communicate with the library
  - A group of functions that make up the **SPI** (service provider interface), which enables the library to communicate with modules
- A **configuration file** or group of **configuration files** containing rules about which module(s) should be invoked for each system-entry application
- **Modules**, or blocks of programming instructions, stored in shared object (.so) files—also known as shared library files

The framework provides a uniform environment for system-entry information to be exchanged, evaluated and acted upon. It enables adjustments to be made to the system-entry process independent of the system-entry applications. Those adjustments can happen in two ways:

- Programmers can create new modules (containing new instructions) to be "plugged in" for one or more system-entry applications.
- System administrators can modify the PAM configuration file(s) to customize how modules are invoked for each system-entry application.

System-entry applications communicate with the PAM library through the API. Modules communicate with the library through the SPI.

3.3 The PAM library

The PAM library (`libpam`) is the central element in the PAM architecture:

- It receives calls from system-entry applications through the API.
- It reads the configuration file (`/etc/pam.conf` or individual files in `/etc/pam.d/`) for line-entries specific to the system-entry application.
- Based on these configuration line-entries, it makes calls to the modules through the SPI.
- It uses the control flag in each line-entry plus the results of each call to a module to determine whether the system-entry application will be granted access.

The PAM library is so named because it is a *shared library* file. A shared library, within the Linux and UNIX arena for programming design, is a compiled file that an executable program (in this case, a system-entry program, such as `telnet`, `ftpd`, or `ssh`) can link to for additional functions and subroutines. Shared libraries are *dynamically linked*, which means the decision for a given executable to link to them can be made at run time.

3.4 PAM process flow

Following is a basic process flow for PAM authentication:

1. When a PAM-enabled system-entry application runs, it opens a PAM session by calling `pam_start()`.
2. `pam_start()` initializes an instance of the PAM library. It also creates a unique handle (ID) that the library can associate with all calls for this session.
3. The application calls `pam_authenticate()`, which is part of the PAM API, and passes the name of the PAM-enabled application to the PAM library.
4. The PAM library (`pamlib`) searches for all line-entries for the PAM-enabled application in the configuration file.
5. For each line-entry it finds, the PAM library calls `pam_sm_authenticate()`. This function is part of the PAM SPI.
6. `pam_sm_authenticate()` looks in the shared library for the module referenced in the configuration file. Once found the module instructions are carried out.
7. The results of each call to a module plus its control flag in the configuration file determine whether the user is allowed access to the system.

In this way, the PAM library connects PAM applications with the PAM modules.

3.5 The four PAM services

PAM's name, which stands for pluggable authentication modules, can be misleading since *authentication* is only one of the four services that PAM can provide for a system-entry application:

PAM service	What the service can do for the system-entry application
Authentication	Validate/reject user identity via two separate functions: (1) authenticate the user and (2) enable user credentials to be set, refreshed, or destroyed.
Account access	Allow an authenticated user to have account access: Check for password aging, account expiration and access hour restrictions.
Password change	Change a user's password.
Session management	Conduct additional tasks before, during or after the session in which PAM interfaces with the system-entry application. *Examples:* Open a database, log events.

All work conducted in the PAM framework is designed around these four services. For example:

- Information in the configuration file(s) is usually organized into four sections (referred to as stacks) that correspond with the four services.
- Each individual module can only do work for one of the four services. This association is called the module type (one module type for each PAM service).
- The PAM library's programming functions for the API and SPI are tied to requests for each of the four services.

The following table demonstrates these associations to the four PAM services:

PAM Service name & name of section (stack) in PAM config file(s)	Module-type token for line-entries in PAM config file(s)	PAM API Functions	PAM SPI Functions
Authentication	`auth`	`pam_authenticate()` `pam_setcred()`	`pam_sm_authenticate()` `pam_sm_setcred()`
Account access	`account`	`pam_acct_mgt()`	`pam_sm_acct_mgt()`
Password change	`password`	`pam_chauthtok()`	`pam_sm_chauthtok()`
Session management	`session`	`pam_open_session()` `pam_close_session()`	`pam_sm_open_session()` `pam_sm_close_session()`

3.6 PAM modules

The container for a PAM module is a shared object (`.so`) file—also commonly known as a shared library file. Shared library files, as discussed earlier in section 3.3, are compiled files that executable programs (such as `telnet`, `ftpd` and `passwd`) can dynamically link to at run time to follow additional functions and subroutines.

Each PAM-module shared library can actually contain up to four separate modules—one for each PAM service (authentication, account access, password change, session management).

So when a system-entry application links first to the PAM library (also a shared library), instructions in the configuration file then determine which PAM modules should also be linked to.

Generally, a set of commonly used modules is available for each implementation of UNIX and Linux. However, you are in no way limited to these publicly available modules. The PAM Framework includes specifications that enable anyone with programming knowledge to design and implement their own modules for custom security requirements.

Note You may find common module names among different PAM implementations, but don't be fooled: these modules are not standard implementations. For example, many of the earlier PAM installations came preconfigured with a module named `pam_unix`. Although this module's general purpose was consistent—to provide basic system-entry to the UNIX or Linux host system—the compiled code it contained varied according to the flavor of UNIX or Linux it came with.

3.7 PAM items

Developers of PAM modules move data between the system-entry application and PAM modules through **items,** standard place-holders to which values can be assigned for exchange. For example, the following items are useful during the start of a PAM session:

Item	Description
PAM_USER	Currently authenticated user
PAM_AUTHTOK	Password
PAM_USER_PROMPT	User name prompt
PAM_TTY	Terminal through which the user communication takes place
PAM_RHOST	Remote host through which user enters the system
PAM_REPOSITORY	Any restrictions on the user account repository
PAM_RESOURCE	Any controls on resources

Items can be set by the system-entry application through the `pam_set_item` function. Values that have been set by the modules can be retrieved by the application through the `pam_get_item` function.

3.8 PAM return codes

After a PAM module executes, it communicates success or failure via a return code. There are numerous possible return codes and some variations among different implementations of PAM. One simple way to categorize all return codes is by basic logic:

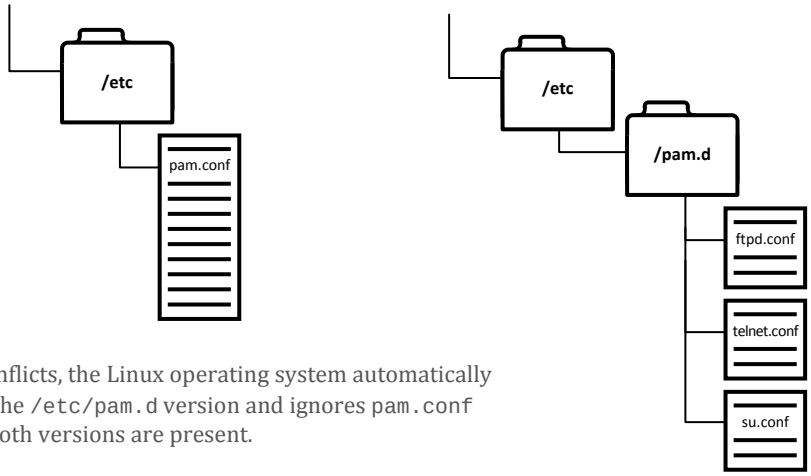
Category	Return code	Description
Module was successful	PAM_SUCCESS	The module successfully completed its intended policy.
Module did not matter	PAM_IGNORE	The module determined that PAM does not need to consider its intended policy.
Module failed	All other return codes. *Examples*: PAM_ABORT, PAM_SYSTEM_ERR, PAM_CRED_ERR, PAM_USER_UNKNOWN, PAM_PERM_DENIED	The module failed to complete its intended policy. Specific cause is detailed by individual return code.

3.9 Single vs. multiple PAM configuration files

The PAM library gets its configuration information from one or more simple text files. These files contain the rules by which the library should link to modules to execute their compiled code. PAM configuration files come in two versions (see Figure 3):

- `/etc/pam.conf`
  All UNIX and some Linux distributions employ this single file to contain configuration information for modules in the PAM framework.
- `/etc/pam.d/`
  In most Linux implementations, this directory exists to hold PAM configuration information, which is separated into individual files—one configuration file for each system-entry application.

Figure 3 PAM configuration information is organized either in a single file or in multiple files.



Note To avoid conflicts, the Linux operating system automatically defaults to the `/etc/pam.d` version and ignores `pam.conf` whenever both versions are present.

3.10 Stacking

Stacking and *stacks* are related terms used to help describe the sequence in which line-entries in a PAM configuration file are processed.

The configuration file is a series of line-entries, which essentially are commands about how to invoke the modules that need to run for any given system-entry application.

A stack represents all line-entries of a particular module type. As detailed in later sections, there are four possible module types, one for each of the four PAM services (authentication, account access, password change, session management). It follows, then, that every configuration file can have up to four stacks.

To reinforce the concept of stacking, convention calls for organizing the line-entries in a PAM configuration file by their stack (module type):

```
#
#authentication stack
telnet line-entry 1 for authentication
telnet line-entry 2 for authentication
passwd line-entry 1 for authentication
OTHER  line-entry 1 for authentication
OTHER  line-entry 2 for authentication
#
#account access stack
telnet line-entry 1 for account access
telnet line-entry 2 for account access
telnet line-entry 3 for account access
OTHER  line-entry 1 for account access
#
#session management stack
OTHER line-entry 1 for session management
#
#password change stack
telnet line-entry 1 for password change
telnet line-entry 2 for password change
passwd line-entry 1 for password change
OTHER  line-entry 1 for password change
OTHER  line-entry 2 for password change
```

So the first line-entry to be read in a stack is said to be at the top of the stack. The last is at the bottom of the stack.

Note While the above organization is helpful in visualizing how PAM configuration files are processed, this is not a requirement. If, for example, line-entries of different module types are interspersed throughout the file, the PAM library will still read through the entire file (from top to bottom) to execute them as stacks.

3.11 Line-entry format for PAM configuration file(s)

A PAM configuration file is basically a collection of line-entries. The line-entries are organized into stacks (see section 3.10) and the order in which the line-entries are written can affect system-entry. The PAM library uses the line-entries as instructions about which modules to invoke for each system-entry application.

These line-entries use a format of tokens separated by one or more spaces or tabs. The format varies slightly between systems using a single `/etc/pam.conf` configuration file and those that use multiple configuration files in the `/etc/pam.d/` directory.

Here's the format for line-entries in `/etc/pam.conf`:

```
service-name  module-type  control-flag  file-path  option
```

And here's the format for line-entries in `/etc/pam.d/`:

```
              module-type  control-flag  file-path  option
```

As you can see, the only difference is the inclusion of the `service-name` token, which is actually the name of the system-entry application:

- With `/etc/pam.conf`, all system-entry applications are covered in the single file.
- With `/etc/pam.d/` an individual configuration file exists for each system-entry application.

Here is an example of an entry line in `/etc/pam.conf`:

```
telnet  auth  required  /usr/lib/security/pam_unix.so.1 debug
```

This entry line tells the PAM library to invoke the authentication module in the `pam_unix.so.1` shared library whenever `telnet` (the system-entry application) runs. With the debug option, it calls for all events to be logged. See the next section for detailed descriptions of each token.

3.12 Line-entry token descriptions

This section describes each of the tokens in a PAM configuration line-entry:

```
service-name  module-type  control-flag  file-path  option
```

With the exception of `file-path`, none of these tokens are case-sensitive, although they usually are written in lowercase for readability. Case sensitivity of the `file-path` token is unique in that it should match file and directory case conventions of the local operating system.

`service-name`
This token indicates the name of the system-entry application, such as `telnet`, `ftpd` and `su`.

When you want an entry to apply to all applications that are not specifically named, you use the keyword `OTHER` (typically capitalized for emphasis) in place of an application name. It is often used at the bottom of each stack for invoking a variety of modules that essentially exist to deny system-entry whenever no other modules can be satisfied.

Note As mentioned earlier, `service-name` is not included when multiple configuration files are used in `/etc/pam.d/`.

`module-type`
This token indicates which module to use in a given shared library, based on the PAM service that the module supports (for more information see Section 3.5, "The four PAM services") . The possible values for module-type are `auth`, `account`, `password` and `session`.

`control-flag`
This token indicates how each invoked module affects the success or failure of the stack. Remember the configuration file includes up to four stacks—lists of rules for each PAM service (authentication, account access, password change, session management). Modules in a stack are invoked in the order they are listed:

control-flag	Action
`required`	The module named in this line-entry must return a successful return code for the stack to succeed. If the named module fails, the stack fails but the next line-entry in the stack is executed.
`requisite`	Same as `required`, except stack processing ceases immediately if the module named in this line-entry fails. `requisite` is not used often.
`sufficient`	If the module named in this line-entry returns a successful return code, and any previous `required` modules have also succeeded, the stack succeeds and the remaining line-entries in the stack are skipped. If the named module fails, it does not affect the success of the stack and the next line-entry in the stack is executed.
`optional`	If the module named in this line-entry returns a successful return code, and no other line-entries in the stack are `required`, the stack succeeds. If the named module fails, the stack can still succeed if any other line-entry in the stack succeeds. In other words, a successful return code from this module is not necessary for the stack to succeed.
`include`	A line-entry with this `control-flag` does not name a module to invoke. Instead it names another text file (in its `file-path` token) that contains additional line-entries to be read and processed as if they were part of this stack. Maximum number for an include stack is 32.
`binding`	If the module named in this line-entry returns a successful return code, and any previous `required` line-entries have also succeeded, the stack succeeds and the remaining line-entries in the stack are skipped. If the named module fails, the stack fails and the next line-entry in the stack is processed.

Note An alternative method and syntax is also available for the `control-flag` token. It gives more control in configuration, but it is more complex, requiring you to determine actions for up to 30 return variables. This method, since it is rarely used in practical PAM implementations, is beyond the scope of this document.

`file-path`
For line-entries with any control-flag other than `include`, this token indicates the path and name of the shared library file containing the module to be invoked. For line-entries with the control-flag `include`, this token indicates the path and name of the text file to be included.

If the first character of the `file-path` is "`/`" it is interpreted as a complete path. Otherwise, the given path is appended to the default module path.

option
This token indicates an optional parameter to be passed to the module. Availability of these options is determined by the module developer. While the developer has full discretion over naming and functionality for each option, many PAM modules support the following option conventions:

option	Description
debug	An option that generally instructs a module to write debug entries to the system log.
no_warn	An option that generally instructs a module not to pass any warnings to a system-entry application that is requesting authentication.
use_first_pass	An option that generally instructs a module to authenticate a user with a password that has already been accepted by a previous module. A user may be denied authentication if the password is not properly passed between the two modules. If the previous module does not validate the password, the current module will deny authentication and quit. This option could be used in a security philosophy that limits user input to a single module.
try_first_pass	An option that is similar to use_first_pass, except that the password accepted by the previous module isn't final. If it fails after being "tried," the user can be reprompted for to attempt another password input.

3.13 PAM configuration file examples

These examples assume the shared library pam_unix.so.1 contains PAM modules with all the subroutines needed by the PAM framework.

A pam.conf file could be as simple as this:

```
OTHER  auth     required  pam_unix.so.1
OTHER  account  required  pam_unix.so.1
OTHER  password required  pam_unix.so.1
OTHER  session  required  pam_unix.so.1
```

In this example, there is one line-entry for each stack. Remember a stack is all the line-entries for a particular module type. Since the above example has no line-entries for specific system-entry applications, all applications using PAM would resolve to the OTHER line-entry for each module type.

Typically, pam.conf files are not this simple. Often there are many system-entry applications that need application specific modules. A more typical pam.conf might look like this:

```
#
#authentication stack
rsh     auth sufficient       pam_rhosts_auth.so.1
rsh     auth required         pam_unix.so.1
su      auth required         pam_rootok.so.1
su      auth required         pam_unix.so.1
ssh     auth required         pam_unix.so.1
passwd  auth required         pam_unix.so.1
OTHER   auth required         pam_deny.so.1
#
#account access stack
rsh     account required      pam_unix.so.1
su      account required      pam_unix.so.1
ssh     account required      pam_unix.so.1
passwd  account required      pam_unix.so.1
OTHER   account required      pam_deny.so.1
#
#password change stack
rsh     password required     pam_unix.so.1
su      password required     pam_unix.so.1
ssh     password required     pam_unix.so.1
passwd  password required     pam_unix.so.1
OTHER   password required     pam_deny.so.1
#
#session management stack
rsh     session required      pam_unix.so.1
su      session required      pam_unix.so.1
ssh     session required      pam_unix.so.1
passwd  session required      pam_unix.so.1
OTHER   session required      pam_deny.so.1
```

In this example, numerous applications are specifically configured for their own needs. Notice how the `rsh` application needs a special module to deal with `.rhosts` files. The `su` application also has a special module to verify the UID of the calling process. Any application not specifically configured resolves to `OTHER`, which denies access through a module named `pam_deny`.

3.14 A closer look at password change

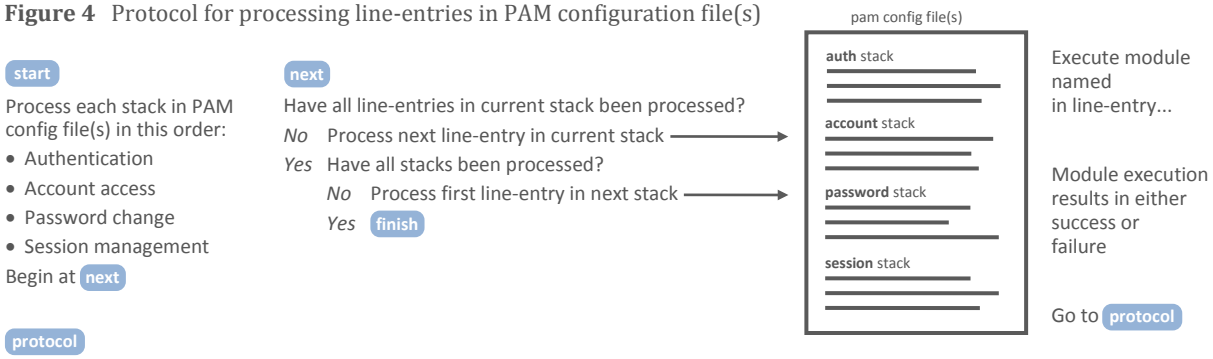
The password change module-type is unique. Modules invoked for this PAM service are designed to operate on a set of assumptions quite different from modules for the other three PAM services. These distinctions are especially helpful in understanding how PAM works with the Fan-Out Driver later in section 5.0 of this manual. If you are only working with the bidirectional driver, you can skip this section unless you are just curious to know more about PAM. The following table explains how PAM works with password change in ways different from the other three PAM services:

What happens with password change	What happens with the other 3 PAM services
PAM processes the password change stack in the PAM configuration file ***only*** if asked to change a user's password. Otherwise, it ignores it.	PAM processes the corresponding stack in the PAM configuration file ***always***.
PAM is asked to change a user's password (to process the password change stack) in two situations: • Whenever `passwd`, the system-entry application that exists for the sole purpose of manually changing a password, runs • Whenever a system-entry application other than `passwd` determines that a user's password needs to be changed. These applications will usually trigger a setting to process the password change stack while their authentication and/or account management PAM services are being executed.	Not relevant since the stacks for these 3 PAM services are always invoked.
When PAM processes the password change stack, it invokes each qualifying module twice. It uses `pam_sm_chauthtok` twice: once to perform a preliminary account validation and once to carry out the actual password change. PAM must receive two return codes before it enables a password change.	PAM invokes each qualifying module in the corresponding stack once. It uses their corresponding calling functions once, moving on to the next operation as soon as one return code is received.

3.15 Line-entry processing logic

Figure 4 illustrates the logic PAM uses to process line entries in the PAM configuration file(s). Note the subtle differences in protocol determined by the module-type (required, requisite, binding, sufficient, optional) for each line-entry.

Figure 4 Protocol for processing line-entries in PAM configuration file(s)



protocol

If module-type in line-entry was...	And result was...	Take this action...
required	success	Go to **next**
	failure	Mark the entire stack as failure and go to **next**
requisite	success	Go to **next**
	failure	Mark the entire stack as failure and **quit**
binding	success	If a required module has already failed, go to **next** If not, **finish**
	failure	Mark the entire stack as failure and go to **next**
sufficient	success	If a required module has already failed, go to **next** If not, **finish**
	failure	If no required modules exist, mark entire stack as failure and go to **next**
optional		(Result matters only if no required or requisite modules exist)
	success	Go to **next**
	failure	Mark the entire stack as failure and go to **next**

4.0 PAM AND THE IDENTITY MANAGER BIDIRECTIONAL DRIVER

Customers who want to connect a Linux/UNIX system to Novell Identity Manager can accomplish this goal by installing the Identity Manager bidirectional driver for Linux and UNIX.

Note Novell offers two driver designs, bidirectional and Fan-Out, enabling customers to choose the most optimal method for connecting their systems to an Identity Manager solution. For information about how PAM works with the Fan-Out Driver, please see section 5.0. For information not specific to PAM on installation, configuration and maintenance of drivers, refer to the Identity Manager driver documentation.

4.1 PAM works passively

The PAM module that comes with the bidirectional driver is passive. Its job is to listen for any changes caused by any other modules in the password change stack. If, and only if, it detects a change, the new password is relayed to Identity Manager.

4.2 How the bidirectional driver works with PAM

By installing the Identity Manager driver for Linux and UNIX, you update your PAM framework in two main areas:

- A PAM module is added. The base name of this module is `pam_nxdrv`, with various possible name extensions depending on the flavor of Linux or UNIX you are running. The module, if invoked, can capture any password changes made by the currently running system-entry program.
- A line-entry for the module is added to your PAM configuration file for two possible scenarios:
  - If any line-entries already exist for the `passwd` system-entry application within the password change stack, an additional line-entry is added after them.
  - If no line-entries exist for the `passwd` application within the password change stack, a line-entry that uses the `OTHER` keyword for its `service-name` token is added to the end of the stack.

As a result of this automatic configuration of PAM, the following will occur:

Anytime the `passwd` system-entry application is used to change a password, the `pam_nxdrv` module is executed. (If the line-entry that was added uses the `OTHER` keyword, then the module is actually executed for any application not listed elsewhere in the stack.) The `pam_nxdrv` module is designed to essentially grab the data about the account whose password is changed and send it to the change log of the driver. The driver, in turn, passes the information to Identity Manager.

Note This functionality, resulting from the default configuration at installation, is limited to, at least, the `passwd` system-entry application. Through customization of the PAM configuration file(s), you can expand this functionality to other system-entry applications. Later sections in this manual explain how to do this.

4.3 PAM automatic configuration

PAM automatic configuration for Identity Manager occurs through `nxdrv-config`, a program included with the Identity Manager driver for Linux and UNIX. There are two ways you invoke this program:

- The program automatically runs during installation, ensuring the option for automatic configuration of the bidirectional driver's PAM module at that time.
- You can invoke the program anytime after that by entering the program name as a command.

If you are configuring PAM on a remote client, the `nxdrv-config` program does the following:

- Prompts you for the host name or IP address and port number of the Linux or UNIX connected system.
- Calls the command to mint a security certificate for the remote client. This command requires you to enter the Remote Loader password.
- Adds a line-entry in the PAM configuration file to execute the PAM module whenever `passwd` runs.

If you are configuring PAM on the connected system, `nxdrv-config` adds a line-entry in the PAM configuration file to execute the PAM module whenever `passwd` runs..

```
Are you configuring PAM from a remote NIS client? (Y/N) [N]
Configuring PAM...
Using PAM configuration file: [/etc/pam.conf]
Inserting line [/usr/lib/security/pam_nxdrv.so.1 mechanism=api]
original PAM file backed up to /etc/pam.conf.nxdrv.04152006151641
```

Note The `nxdrv-config` program makes a backup copy of the PAM configuration file before it is modified.

The file name and location for the PAM module installed by `nxdrv_config` depends on which version of Linux or UNIX you are running:

Operating system	PAM module name and location
AIX	`/usr/lib/security/pam_nxdrv`
HP-UX	`/usr/lib/security/libpam_nxdrv.1`
Linux	`/lib/security/pam_nxdrv.so`
Solaris	`/usr/lib/security/pam_nxdrv.so.1`

4.4 Installing PAM locally vs. remotely

As explained in the previous section, installation and configuration of the bidirectional driver and its PAM components varies according to how you store accounts information on your Linux and UNIX servers:

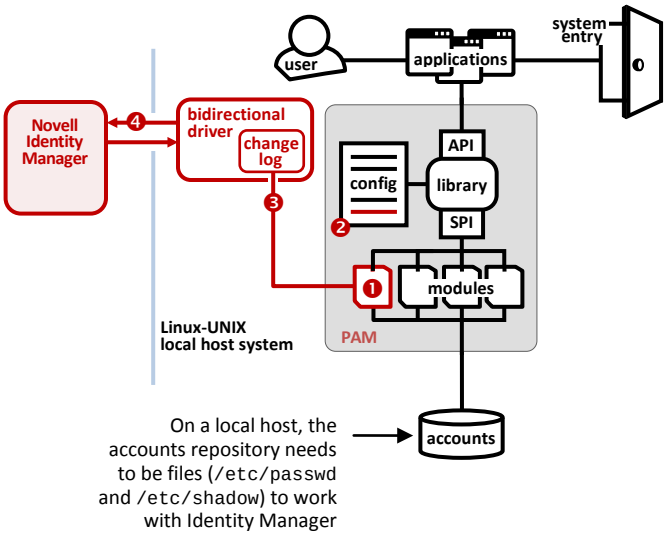
- If a system uses files (`/etc/password` and `/etc/shadow`), then that system is treated as a host and receives a complete driver installation.
- If a system uses a centralized implementation of NIS or NIS+ *that resides on that system*, then that system also is treated as a host and receives a complete driver installation.
- If a system uses a centralized implementation of NIS or NIS+ *that resides on a separate server*, then that system is treated as a remote client and receives a partial driver installation, mainly consisting of the PAM components. Those components are then configured to communicate with the driver installed on the NIS/NIS+ server.

4.4.1 Local host installation

If a system is using `/etc/password` and `/etc/shadow` or an NIS/NIS+ implementation that resides on that system to store its account information, it will employ a complete installation of the bidirectional driver to connect with Identity Manager.

Figure 5 illustrates how the addition of the bidirectional driver's PAM components enable communication with Identity Manager on a local host.

Figure 5 Adding Identity Manager to PAM on a local host.



❶ Identity Manager module (pam_nxdrv)
This module is plugged in to the local PAM framework when the bidirectional driver is installed on the platform.

❷ PAM configuration file(s)
A line-entry is added to the password-change stack when the bidirectional driver is installed. It invokes the Identity Manager module (❶) anytime the `passwd` system-entry application is used to change a password.

❸ change log
Anytime the module (❶) executes and detects a password change, it writes that event to the change log of the bidirectional driver.

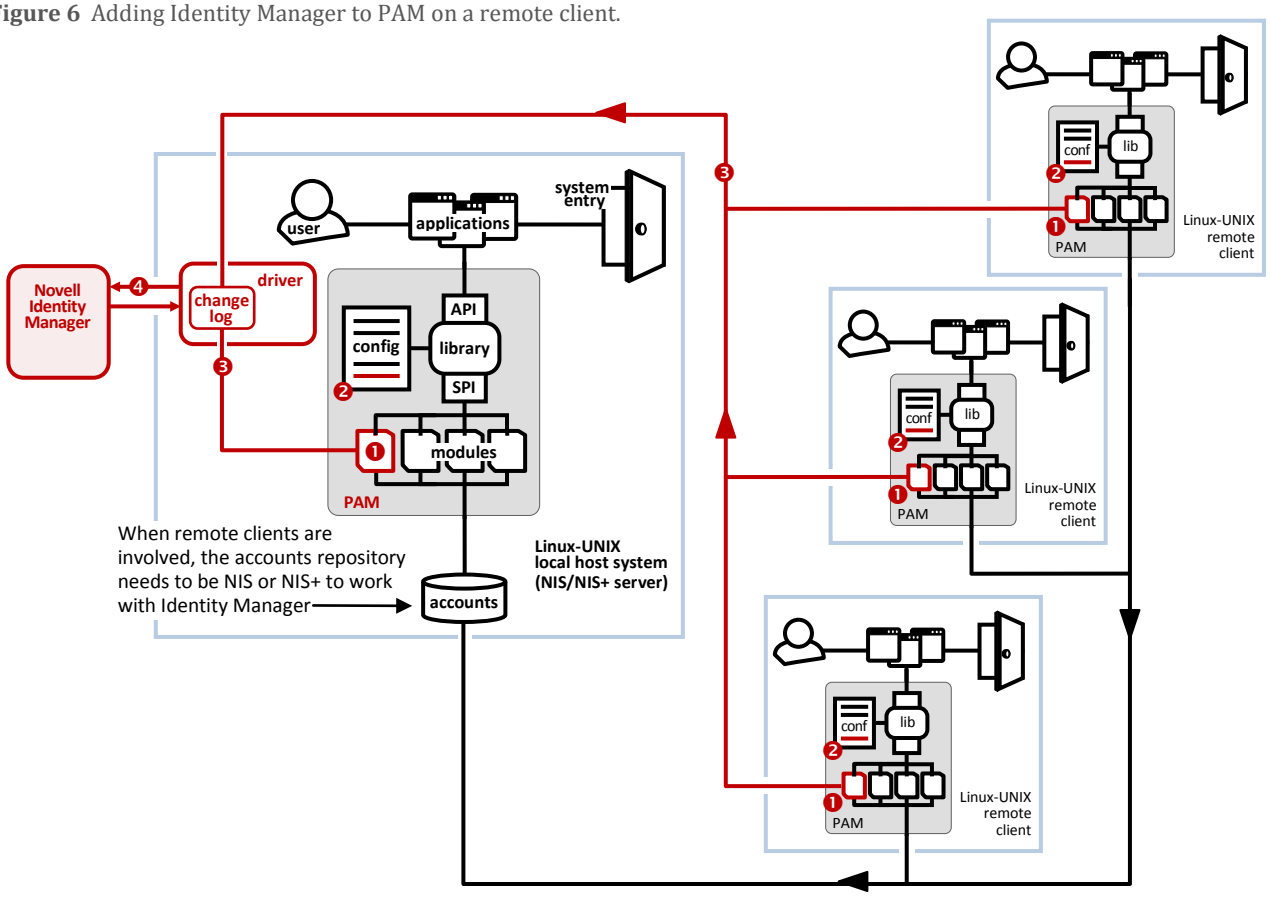
❹ Identity Manager update
The bidirectional driver communicates with Identity Manager and the password change is recorded in the Identity Vault.

4.4.2 Remote client installation

If a system is using an NIS/NIS+ implementation that resides on a different server to store its account information, it will employ a partial installation of the bidirectional driver, mainly consisting of PAM components, which notify the driver installed on the separate NIS/NIS+ server of password changes. The driver then relays this information to Identity Manager.

Figure 6 illustrates how the addition of the bidirectional driver's PAM components enable communication with Identity Manager on remote clients.

Figure 6 Adding Identity Manager to PAM on a remote client.



❶ Identity Manager module (`pam_nxdrv`)
This module is plugged in to the local PAM framework on all systems.

❷ PAM configuration file(s)
A line-entry is added to the Password Change stack during driver installation. It invokes the Identity Manager PAM module (❶) anytime the `passwd` system-entry application is used to change a password.

❸ change log
Anytime any of the PAM modules (❶) execute and detect a password change, it writes that event to the change log of the bidirectional driver on the NIS/NIS+ server.

❹ Identity Manager update
The bidirectional driver communicates with Identity Manager and the password change is recorded in the Identity Vault.

4.5 Working with the default PAM installation

To review, installing the bidirectional driver for Linux and UNIX automatically configures PAM, at a minimum, to notify Identity Manager about any password changes made through use of the `passwd` system-entry application.

Since there are many other system-entry applications (such as `telnet`, `su`, and `ftpd`) that allow you to change or create passwords on Linux and UNIX, the following should be repeated for emphasis:

> Default configuration of the PAM module that comes with the bidirectional driver is only certain to detect password changes submitted through `passwd`.

As the next section discusses, you can expand this functionality to other system-entry applications by customizing your PAM configuration file(s). However, if you expect to conduct most or all of your password changes through `passwd`, here are some guidelines to be aware of:

- Any password created or changed through a program other than `passwd` may not be known by Identity Manager.
- If you use any of the following program commands for password configuration, Identity Manager will not be notified because these commands will bypass PAM:
  - `yppasswd`
  - `passwd –r`
  - `useradd`

4.6 A closer look at default PAM configuration

So what happens to the PAM configuration file(s) during installation of the bidirectional driver?

There are two possible answers to this question, depending on what already exists in the PAM configuration file(s):

- If any line-entries exist for the `passwd` system-entry application in the password change stack, an additional line-entry is added after them to execute the driver's PAM module.
- If no line-entries exist for the `passwd` application in the password change stack, a line-entry using the `OTHER` keyword (as its `service-name` token) is added to the end of the stack to execute the driver's PAM module.

4.6.1 When passwd line-entry is present

Following is a sample `pam.conf` file before installation of the Identity Manager driver:

```
#authentication stack
telnet   auth     sufficient  /lib/security/pam_specialtelnet
telnet   auth     required    /lib/security/pam_generic
passwd   auth     required    /lib/security/pam_generic
OTHER    auth     required    /lib/security/pam_generic

#account access stack
telnet   account  required    /lib/security/pam_specialtelnet
telnet   account  required    /lib/security/pam_generic
passwd   account  required    /lib/security/pam_generic
OTHER    account  required    /lib/security/pam_generic

#session management stack
OTHER    session  required    /lib/security/pam_generic

#password change stack
telnet   password required    /lib/security/pam_generic
passwd   password required    /lib/security/pam_generic
OTHER    password required    /lib/security/pam_generic
```

Since the sample file already contains a line-entry for `passwd` in the the password change stack, a new line will be added immediately after it. Here is that particular stack with the new line-entry (bold text):

```
#password change stack
telnet   password required    /lib/security/pam_generic
passwd   password required    /lib/security/pam_generic
passwd   password required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
OTHER    password required    /lib/security/pam_generic
```

Once this line is added, when any other PAM modules for `passwd` participate in a dialog with a user who is changing the password, the driver's PAM module will use the `pam_get_item` function to collect the new password. It writes the new password to the change log so it can be published into the Identity Vault.

The driver's PAM module line-entry requires an option token. In the example above, the option is `mechanism=api`. Automatic configuration assigns this option when the driver's PAM module is running on the local system. Available options for the bidirectional driver's PAM module are discussed later in section 4.5.3.

4.6.2 When passwd line-entry is not present

Following is another sample `pam.conf` file before installation of the Identity Manager driver:

```
#authentication stack
telnet   auth     sufficient  /lib/security/pam_specialtelnet
telnet   auth     required    /lib/security/pam_generic
OTHER    auth     required    /lib/security/pam_generic

#account access stack
OTHER    account  required    /lib/security/pam_generic

#session management stack
OTHER    session  required    /lib/security/pam_generic

#password change stack
telnet   password required    /lib/security/pam_generic
OTHER    password required    /lib/security/pam_generic
```

Since the sample file does not contain a line-entry for `passwd` in the the password change stack, a new line using the `OTHER` keyword is added to the end of the stack to execute the driver's PAM module.

Here, again, is the password change stack with the new line-entry (bold text):

```
#password change stack
telnet   password  required    /lib/security/pam_generic
OTHER    password  required    /lib/security/pam_generic
OTHER    password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
```

Once this line is added, when any PAM modules not specified elsewhere in the password change stack participate in a dialog with a user who is changing the password, the driver's PAM module will use the `pam_get_item` function to collect the new password. It writes the new password to the change log so it can be published into the Identity Vault.

The driver's PAM module line-entry requires an option token. In the example above, the option is `mechanism=api`. Automatic configuration assigns this option when the driver's PAM module is running on the local system. Available options for the driver's PAM module are discussed later in section 4.5.3.

4.6.3 Options for the driver's PAM module

Here again is the syntax for a line-entry in a configuration file. The option token is at the end.

```
service-name  module-type  control-flag  file-path  option
```

Note For more information about all of the tokens in a PAM configuration file line-entry, see section 3.12.

The `option` token indicates an optional parameter to be passed to the module when it is executed. The PAM module that comes with the Identity Manager driver includes four options that are used to distinguish the accounts repository and whether the system is a local host or a remote client. A fifth option is available for debugging.

Option	Description
`mechanism=api`	The PAM module uses the API to send password change information to the driver. This method is used when the PAM module is running on the same system as the driver shim.
`mechanism=soap`	The PAM module uses Simple Object Access Protocol (SOAP) to send password change information to the driver. This method is used when the PAM module is running on a different system from the driver, such as with NIS or NIS+ clients.
`host=`*hostName*	Required for SOAP. Specifies the host name or IP address of the system with the driver.
`port=`*portNumber*	Required for SOAP. Specifies the TCP port number of the system with the driver. The default port is 8091.
`debug=*`	Logs PAM module activity to the `/usr/local/nxdrv/logs/pam_nxdrv.log` file.

Returning to the sample PAM configuration file from section 4.5.1, the following example shows the option that would be used in the password change stack if the system was a remote NIS/NIS+ client:

```
#password change stack
telnet   password  required    /lib/security/pam_generic
passwd   password  required    /lib/security/pam_generic
passwd   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=soap
   host=130.127.5.165 port=8091
OTHER    password  required    /lib/security/pam_ generic
```

4.7 Beyond the default PAM configuration

Default PAM configuration only ensures that Identity Manager will detect password changes submitted through the `passwd` system-entry application. Expanding this detection to other system-entry applications, such as `telnet` and `ftpd`, requires you to edit your PAM configuration file manually.

Note Always make a backup copy before editing a PAM configuration file.

For each system-entry application you wish Identity Manager to recognize, edit the PAM configuration file(s) as follows:

In the password change stack, add a line-entry for the bidirectional driver's PAM module. If the stack already has one or more line-entries for that particular system-entry application, insert the new line-entry after them.

4.7.1 Placement of the line-entry

Here is a password change stack from a sample `pam.conf` file:

```
#password change stack
ftpd     password  required    /lib/security/pam_generic
passwd   password  required    /lib/security/pam_generic
passwd   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
telnet   password  required    /lib/security/pam_generic
OTHER    password  required    /lib/security/pam_generic
```

If you wanted to enable Identity Manager to detect any password changes made through `ftp`, you would add a line-entry after the existing `ftpd` line-entry as follows (bold text):

```
#password change stack
ftpd     password  required    /lib/security/pam_generic
ftpd     password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
passwd   password  required    /lib/security/pam_generic
passwd   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
telnet   password  required    /lib/security/pam_generic
OTHER    password  required    /lib/security/pam_generic
```

If you wanted to enable Identity Manager to detect any password changes made through `telnet`, you would add a line-entry after the existing `telnet` line-entry as follows (bold text):

```
#password change stack
ftpd     password  required    /lib/security/pam_generic
ftpd     password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
passwd   password  required    /lib/security/pam_generic
passwd   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
telnet   password  required    /lib/security/pam_generic
telnet   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
OTHER    password  required    /lib/security/pam_generic
```

Finally, if you wanted to enable Identity Manager to detect any password changes made through `su`, the requirements would be different, since no other line-entries exist for `su`. You could add this line-entry anywhere in the stack as long as was above any line-entries for `OTHER`:

```
#password change stack
ftpd     password  required    /lib/security/pam_generic
ftpd     password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
passwd   password  required    /lib/security/pam_generic
passwd   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
telnet   password  required    /lib/security/pam_generic
telnet   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
su       password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=api
OTHER    password  required    /lib/security/pam_generic
```

4.7.2 Line-entry syntax for Identity Manager

The line-entry syntax for each system-entry application you wish Identity Manager to monitor includes constants and variables, depending on your operating system and whether PAM is running on a local host or remote client.

Here again are the tokens that comprise a complete line-entry:

```
service-name  module-type  control-flag  file-path  option
```

The following table defines how to determine the values for these tokens:

Token	Value	Comments
`service-name`	*Examples:* `telnet`, `ftp`, `su`	Name of system-entry application
`module-type`	`password`	Always use this value (for the password change stack)
`control-flag`	`required`	Always use this value
`file-path`	`/usr/lib/security/pam_nxdrv`	Use for AIX
	`/usr/lib/security/libpam_nxdrv.1`	Use for HP-UX
	`/lib/security/pam_nxdrv.so`	Use for Linux
	`/usr/lib/security/pam_nxdrv.so.1`	Use for Solaris
`option`	`mechanism=api`	Use when the PAM module is running on the same system as the driver.

Token	Value	Comments
option	mechanism=soap	Use when the PAM module is running on a different system from the driver, such as with NIS or NIS+ clients.
	host=*hostName*	Use with SOAP to specify the host name or IP address of the system with the driver.
	port=*portNumber*	Use with SOAP to specify the TCP port number of the system with the driver. The default port is 8091.
	debug=*	Use to log PAM module activity to the /usr/local/nxdrv/logs/pam_nxdrv.log file.

Following are three examples of line-entries for system-entry applications monitored by Identity Manager:

```
su       password  required    /usr/lib/security/pam_nxdrv       mechanism=api

telnet   password  required    /usr/lib/security/pam_nxdrv.so.1  mechanism=soap
   host=130.128.5.163 port=8092

ftpd     password  required    /usr/lib/security/libpam_nxdrv.1  mechanism=api
```

Each of these examples would need to be added after any line-entries referring to the same service-entry application.

4.8 Logging PAM activity

The bidirectional driver's PAM module generates log messages. You can write these messages to a log for later review by adding the debug=* option to the entry line in the PAM configuration file. Messages will be written to: /usr/local/nxdrv/logs/pam_nxdrv.log.

4.9 Error messages for Identity Manager drivers

Identity Manager drivers use a standard error message protocol. Error messages relating to PAM on the bidirectional driver for Linux and UNIX will begin with the prefix NXPAM.

At this manual's writing, only one NXPAM error message is used to communicate PAM issues:

```
NXPAM001W Password Change was not submitted for user
```

Explanation: When a user changes the password using a PAM-enabled application, the PAM module for the driver submits the password change to the change log. An error occurred that prevents the change being submitted to the change log.

Possible causes:

- If the PAM module is running locally on the same system with the driver shim, certain files or directories could be missing, such as the /usr/local/nxdrv/keys/lpwd1f40 driver shim key file or the /usr/local/nxdrv/changelog change log directory.
- If the PAM module is running remotely from the system with the driver shim, the PAM module could not connect to the driver shim. This could be caused by a network problem or a problem with the driver shim.
- The PAM module might not be configured properly.

Actions:

- Ensure that the PAM module is installed and configured correctly.
- Ensure that the driver shim is running and healthy.
- If the PAM module is running remotely, verify connectivity to the driver shim system.

5.0 PAM AND THE IDENTITY MANAGER FAN-OUT DRIVER

Customers who want to connect a Linux/UNIX system to Novell Identity Manager can accomplish this goal by installing the Identity Manager Fan-Out Driver for Linux and UNIX.

Note Novell offers two driver designs, bidirectional and Fan-Out, enabling customers to choose the most optimal method for connecting their systems to an Identity Manager solution. For information about how PAM works with the bidirectional driver, please see section 4.0. For information not specific to PAM on installation, configuration and maintenance of drivers, refer to the Identity Manager driver administration documentation.

5.1 Fan-Out Driver design summary

Design of the Fan-Out Driver is discussed in detail in its accompanying system administration documentation. However, a quick summary here may be helpful before examining how it integrates with PAM on Linux and UNIX systems.

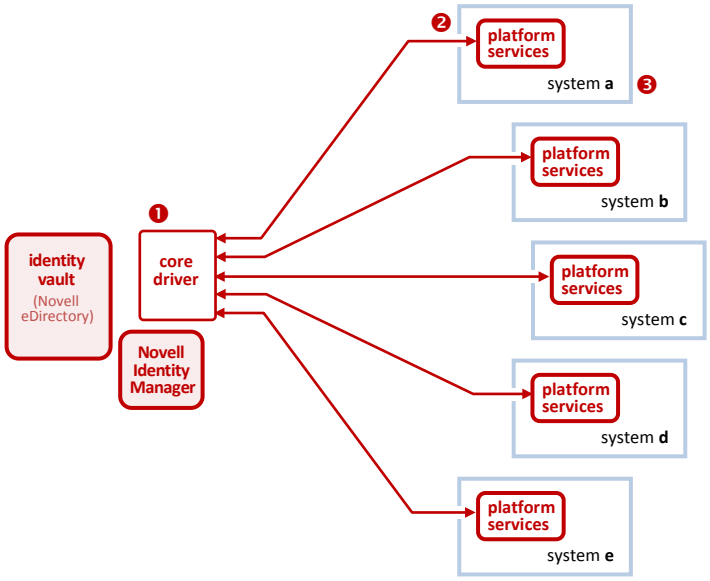
To offer significant scalability, the Fan-Out Driver's design ventures beyond the standard bidirectional architecture used by other Identity Manager drivers.

Unlike the one-driver to one-system relationship provided by a bidirectional driver, a single Fan-Out Driver can work for Identity Manager as an interface to many systems—up to 100 systems per driver.

Developers accomplished this scalability by separating the Fan-Out Driver into two parts:

- A **Core Driver**, which presents itself to Identity Manager as a single driver.
- A set of software components called **Platform Services**, installed on each connected system, that interfaces with the Core Driver.

Figure 7 How the Fan-Out Driver works



❶ Core Driver
The Core Driver interfaces with Identity Manager as a single driver. It uses the standard subscriber channel to receive updates from Identity Manager. It uses LDAP to return information directly to the Identity Vault.

❷ Platform Services
Platform Services is installed on each system that is being synchronized with Identity Manager through the Fan-Out Driver. Different versions of platform services are available for Linux, UNIX, z/OS (IBM mainframes), and i5/OS (OS/400).

Note The versions for Linux and UNIX leverage PAM.

❸ Systems
One Core Driver can support up to 100 systems (with 100 individual installations of Platform Services). Each installation of Platform Services communicates only with the Core Driver.

Different versions of Platform Services are available for the operating environments of Linux and UNIX, z/OS (IBM mainframes) and i5/OS (OS/400). The makeup for each version is tailored to the native application interfaces of each environment.

The version of Platform Services that is used on Linux and UNIX includes software components that can work with PAM.

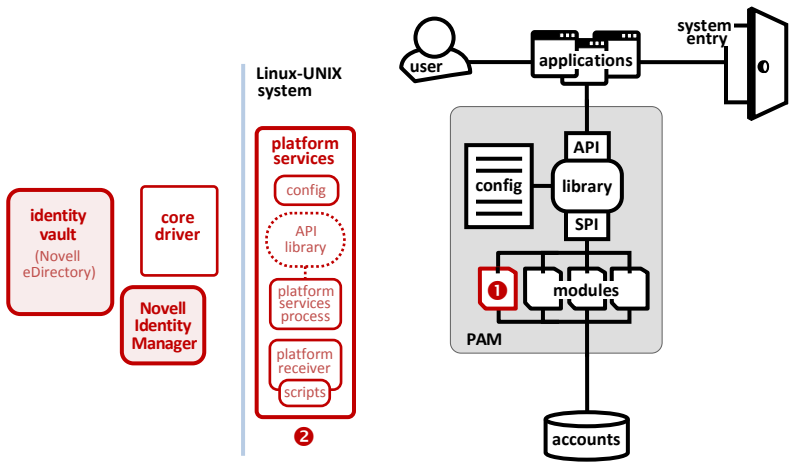
5.2 How the Fan-Out Driver for Linux and UNIX works with PAM

The previous section explained how each connected system using the Fan-Out Driver has its own installation of Platform Services, a software package that is tailored to work with the local environment. Since the basic job of the driver is to synchronize authentication and authorization information between a connected system and Identity Manager, it follows that the version of Platform Services on a Linux or UNIX connected system would include the ability to leverage PAM.

5.2.1 What gets installed

Figure 8 shows a Linux-UNIX system after Platform Services has been installed. This illustration only highlights the pieces that relate to PAM; while other components do exist, they are omitted for a more focused discussion.

Figure 8 Adding Platform Services to a Linux/UNIX system.



❶ **Identity Manager module (`pam_ascauth`)**
This module is plugged in to the local PAM framework as part of the platform services installation. Any system-entry application that you configure to use this module will be redirected to Identity Manager.

❷ **Core Driver interface**
The other parts of Platform Services associated with PAM are the configuration file (`asamplat.conf`), the API library, the platform services process (`asampsp`), and the platform receiver (`asamrcvr`).

Note Installation of these components does not cause any immediate changes in PAM functionality. The added software is dormant until necessary changes to the PAM configuration file(s) are made.

Installing the Platform Services for Linux and UNIX adds the following components to support your PAM framework:

- A PAM module (shared library) is added. Its base name is `pam_ascauth`, with various possible name extensions depending on the flavor of Linux or UNIX you are running. It includes modules that can be invoked for each of the four PAM services (authentication, account access, password change, session management).
- Other software components to facilitate communication between PAM and the Core Driver:
  - **Configuration File** – An editable text file (`asamplat.conf`) referenced by other components of Platform Services for values, settings, and instructions.
  - **API Library** – This is a subset of the functions and subroutines that Platform Services uses to exchange data with the Core Driver, which, in turn, gets its information from the Identity Vault.
  - **Platform Services Process** – A program (`asampsp`) that, with support from the API library, prioritizes and facilitates data requests/replies to and from the Core Driver.
  - **Platform Receiver** – A program (`asamrcvr`) that receives Identity Manager updates from the Core Driver and uses a set of scripts to sychronize the changes with the local accounts database, which can be files (`/etc/passwd` + `/etc/shadow`), NIS, or NIS+.

Note The API Library is open to customers for integration with local applications. The scripts that work with the Platform Receiver also can be edited for local customization.

5.2.2 How PAM relates to Fan-Out Driver authentication modes

The Fan-Out Driver offers numerous modes of authentication depending on the version of Platform Services you use. When you install Platform Services on a Linux or UNIX system, those modes can be:

- Authentication Redirection
- Authentication Redirection with Local Failover
- Local Authentication
- Name Service Switch Authentication

The first two options listed above employ the Fan-Out driver's PAM module; the last two options don't. It is therefore assumed—because you are consulting this manual—that your system is configured for either of the Authentication Redirection options. As their names imply, these two options ***redirect*** users on the connected system to authenticate through your Identity Manager solution; on Linux and UNIX systems, Platform Services accomplishes this redirection through PAM.

Note To support the other two options, Platform Services includes software components not related to PAM. For more information about all modes of authentication and how and when they are used, please refer to the Fan-Out Driver's administration documentation.

5.2.3 PAM file locations

The file name and installed location for the PAM shared library file depends on which version of Linux or UNIX you are running:

Operating system	PAM module name and location
AIX	`/usr/lib/security/pam_ascauth`
HP-UX	`/usr/lib/security/libpam_ascauth.1`
Linux	`/lib/security/pam_ ascauth.so`
Solaris	`/usr/lib/security/pam_ ascauth.so.1`

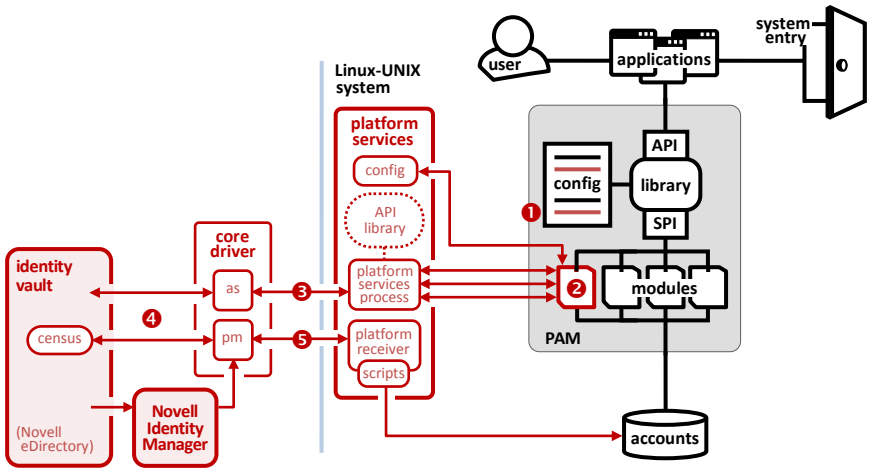
5.2.4 Data flow

Installing Platform Services on the connected Linux or UNIX system does not cause any immediate changes to PAM functionality. The added software is dormant until new line-entries are added to the PAM configuration file(s).

While later sections in this manual examine these new line-entries, this section assumes modifications have already been made to the PAM configuration file(s) for the purpose of demonstrating how data flows through the components of the Fan-Out Driver.

Figure 9 shows a Linux-UNIX system configured to use the PAM module that comes with Platform Services. This illustration only highlights the pieces that relate to PAM; while other components do exist, they are omitted for a more focused discussion.

Figure 9 PAM data flow on a Linux/UNIX system.



❶ PAM configuration file(s)

To enable a system-entry application to invoke the PAM module that comes with the Fan-Out Driver, you must manually add the appropriate line-entries to your PAM configuration file(s). Preconfigured templates are also available for the most common implementations of Linux and UNIX.

The line-entries to be added depend on:

- Which system-entry applications you want to be redirected to Identity Manager
- Which of the four PAM services those system-entry applications use

❷ Identity Manager module (`pam_ascauth`)

This module overrides other PAM modules for any system-entry application configured to use it:

- It routes authentication requests (user name and password) to the Identity Vault via the Platform Services Process.
- It uses the local accounts database for other needed provisioning information.

❸ Platform Services Process (`asampsp`)

Prioritizes and facilitates communication between the PAM module and the Core Driver's Authentication Services (as), checking user names and passwords against those stored in eDirectory.

❹ Core Driver

Parts of the Core Driver that work with PAM include:

- Authentication Services (as) – Checks user names and passwords in eDirectory as requested by the PAM module and as arbitrated by the Platform Services Process.
- Provisioning Manager (pm) – Receives account updates from Identity Manager subscriber channel, forwarding them to the Census and the Platform Receiver.
- Census – An eDirectory object that tracks all users being controlled by the Fan-Out Driver on all systems it connects to. The Census can be viewed through the iManager Web interface.

❺ Platform Receiver (`asamrcvr`)

Receives provisioning updates originating from the Identity Manager subscriber channel. Employs scripts to synchronize local accounts database with any updates to eDirectory.

5.2.5 About the Platform Services configuration file

An editable text file, `asamplat.conf`, contains a set of statements that other components of Platform Services read to employ whatever preferences and settings you wish to be part of your Identity Manager interface.

Note In Figure 9, `asamplat.conf` is represented in the Platform Services box as "config."

The wide range of statements that can be used in `asamplat.conf`, including their optional parameters, is discussed in detail in the Fan-Out Driver's system administration documentation. However, a quick summary of the statements that are referenced by the PAM module is included below for your information:

Statement	Description
AM.GROUP.INCLUDE and AM.GROUP.EXCLUDE	Provides a list of specific groups or group masks to be included or excluded from identity provisioning.
AM.USER.INCLUDE and AM.USER.EXCLUDE	Provides a list of specific user IDs or user ID masks to be included or excluded from identity provisioning.
AS.USER.INCLUDE and AS.USER.EXCLUDE	Provides a list of specific user IDs or user ID masks to be included or excluded from Authentication Services.
PASSWORDPROMPT	Specifies the prompt issued by the PAM module to request the user's password for authentication.
PASSWORDPROMPTCURRENT	Specifies the prompt issued by the PAM module to request the user's current password for password changes.
PASSWORDPROMPTCHANGE	Specifies the prompt issued by the PAM module to request the user's new password for password changes.
PASSWORDPROMPTCHANGEAGAIN	Specifies the prompt issued by the PAM module to verify the user's new password for password changes.
UPDATEPASSWORD	Specifies that the driver updates the local security system upon a successful check password or change password operation, or when password replication information is received from the Core Driver.
PASSWORDPROMPT	Specifies the prompt issued by the PAM module to request the user's password for authentication.
PASSWORDPROMPTCURRENT	Specifies the prompt issued by the PAM module to request the user's current password for password changes.
PASSWORDPROMPTCHANGE	Specifies the prompt issued by the PAM module to request the user's new password for password changes.
PASSWORDPROMPTCHANGEAGAIN	Specifies the prompt issued by the PAM module to verify the user's new password for password changes.

5.3 A closer look at the PAM module

To enable your system-entry applications to work with Identity Manager, you will need to make modifications to your PAM configuration file(s) in some shape or form. The process for determining these modifications will be easier if you understand the inner workings of the PAM module that comes with Platform Services for Linux and UNIX.

To begin, remember that the container for a PAM module is a shared library file. Shared library files, as discussed in sections 3.3 and 3.6, are compiled files that executable applications (such as `telnet`, `ftpd` and `passwd`) can dynamically link to at run time to access additional functions and subroutines.

Each PAM shared library can actually contain up to four separate modules of instructions—one for each PAM service (authentication, account access, password change, session management). In fact, the PAM shared library file that is included with Platform Services does contain all four of these possible modules.

The remainder of this section looks at the tasks carried out by each of the four modules and ends with a summary of their functionality.

5.3.1 Authentication

Original condition:

When the authentication stack in a PAM configuration file includes a line-entry for the Platform Services PAM module (`pam_ascauth`), the following occurs:

Possible alternatives:

When a line-entry for the Platform Services PAM module (`pam_ascauth`) is included in the authentication stack for the system-entry application that is calling PAM, the following occurs:

When a line-entry in the authentication stack invokes the Platform Services PAM module (`pam_ascauth`) for a particular system-entry application, the following occurs:

When a system-entry application invokes the Platform Services PAM module via a line-entry in the authentication stack, the following occurs:

- The function pam_sm_authenticate is called to invoke the module; the pam_handle (unique session identifier and data) is included as an argument
- Check for inclusion of any options associated with this module (options are listed in section 5.4).
- Confirm that all necessary inputs exist in pam_handle.
  - If calling system-entry application is passwd and the user is root, exit module with return code PAM_IGNORE. **Reason:** This module does not allow for administrative password resets in the Identity Vault (Subsequent PAM modules may allow root to administratively reset passwords in local password store).
  - If pam_handle does not include a user name, exit module with return code PAM_USER_UNKNOWN.
- Check for PAM-specific configuration settings in asamplat.conf (see section 5.2.5).
- Check user against exclude/include list in asamplat.conf.
  - If user is excluded, exit module with return code PAM_IGNORE.
- Check pam_handle to see if password was already submitted through a previous PAM module.
- Check product registration
  - If product registration does not exist or is expired, exit module with return code PAM_AUTH_ERR.
- If no password was found in pam_handle, attempt authentication (as described later)
- Authentication steps:
  - Prompt user for password.
  - Send user name and password to Platform Services Process.
  - Platform Services Process sends a request to Authentication Services in Core Driver.
  - Authentication Services requests verification of the user and password from the Identity Vault (eDirectory).
    - If user and password matches, collect account data from Identity Vault for later use.
  - Authentication Services relays data to Platform Services Process.
  - Platform Services Process relays data to PAM module.
  - If password is correct, store account data in pam_handle for later use.
  - If line-entry in PAM configuration file included the option try_first_pass
    - Exit module only if password is correctwith return code PAM_SUCCESS.
  - If line-entry in PAM configuration file included the option use_first_pass
    - Exit module whether successful or not.
    - If successful, exit module with return code PAM_SUCCESS.
  -
- Whether successful or not, exit module with various possible return codes (described later).pam_sm_authenticate sends return code (successful or unsuccessful) to PAM framework

Once a return code is sent to the PAM framework, the next line-entry in the PAM configuration file can be executed

5.3.2 Account access

When the account access stack in a PAM configuration file includes a line-entry for the Platform Services PAM module (pam_ascauth), the following occurs:

- The function pam_sm_acct_mgt is called to invoke the module; the pam_handle (unique session identifier and data) is included as an argument
- Instructions in the module are executed:
  - Check for inclusion of any options associated with this module (options are listed in section 5.4)
  - Confirm that all necessary inputs exist in pam_handle
  - If user is root: processing ends; pam_sm_acct_mgt sends return code to proceed as if this module does not exist
  - Check user against exclude/include list in asamplat.conf (see section 5.2.5)
  - If earlier authentication module sent an unsuccessful return code: end processing; pam_sm_acct_mgt sends a return code to proceed as if this module does not exist
  - If earlier authentication module sent a successful return code: get extra account data from pam_handle
  - If account is disabled: processing ends; pam_sm_acct_mgt sends unsuccessful return code to PAM Framework

- If account is expired: processing ends; `pam_sm_acct_mgt` sends return code instructing PAM Framework that conditions exist for invoking the password change module

- `pam_sm_authenticate` sends a successful return code to PAM framework

Once a return code is sent to the PAM framework, the next line-entry in the PAM configuration file can be executed

5.3.3 Password change

Note Remember that password change is unique. Before examining the tasks carried out by this module, you may want to review section 3.14, which lists differences between password change and the other three PAM services.

When the password change stack in a PAM configuration file includes a line-entry for the Platform Services PAM module (`pam_ascauth`), the following occurs:

- The function `pam_sm_chauthtok` is called to invoke the module; the `pam_handle` (unique session identifier and data) is included as an argument; the argument `pam_prelim_check` is also included to signal that the first stage in execution should be carried out
- Instructions designated in the module for the preliminary stage are executed:
  - Check for inclusion of any options associated with this module (options are listed in section 5.4)
  - Confirm that all necessary inputs exist in `pam_handle`
  - Check user against exclude/include list in `asamplat.conf` (see section 5.2.5)
  - Verify user's account presence in Identity Vault by checking data sent to `pam_handle` by earlier authentication module; if not managed, `pam_sm_acct_mgt` sends to return code to proceed as if this module does not exist

Note This is a special check for Sun Solaris, which configures PAM to process both the authentication and password change stacks whenever the `passwd` system-entry application runs. Other implementations of Linux and UNIX only process the password change stack for `passwd`.
??? Missing something here—reason for this exception still not compelling

  - If user is `root` and the system-entry application is `passwd`: processing ends; `pam_sm_acct_mgt` sends to return code to proceed as if this module does not exist
- `pam_sm_ chauthtok` sends a successful return code to PAM framework
- The function `pam_sm_chauthtok` is called to invoke the module a second time; the `pam_handle` (unique session identifier and data) is included as an argument; the argument `pam_prelim_check` is ***not*** included to signal that the second stage in execution should be carried out
- Instructions designated in the module for the second stage are executed:
  - Check for inclusion of any options associated with this module (options are listed in section 5.4)
  - Confirm that all necessary inputs exist in `pam_handle`
  - Verify user's account presence in Identity Vault by checking data sent to `pam_handle` by earlier authentication module; if not managed, `pam_sm_acct_mgt` sends to return code to proceed as if this module does not exist
  - If user is `root` and the system-entry application is `passwd`: processing ends; `pam_sm_acct_mgt` sends to return code to proceed as if this module does not exist
  - Prompt user for old and new passwords, using settings in `asamplat.conf`
  - Check user against exclude/include list in `asamplat.conf` (see section 5.2.5)
  - If the system-entry application is not `passwd`, look in `pam_handle` to check if authentication was also processed through the PAM module that comes with Identity Manager; if it was not, processing ends; `pam_sm_acct_mgt` sends unsuccessful return code to PAM Framework
  - Check `pam_handle` for presence of `pam_change_expired_authtok` setting; this administrative option would require a password to have expired before it can be changed
  - Synchronize local system account database with new password information in Identity Vault if that requirement is set in `asamplat.conf` (see section 5.2.5)
  - Synchronize SAMBA password with information in Identity Vault if that requirement is set in `asamplat.conf` (see section 5.2.5)
- `pam_sm_chauthtok` sends a second successful return code to PAM framework, which enables the password to be reset

Once a return code is sent to the PAM framework, the next line-entry in the PAM configuration file can be executed

5.3.4 Session management

When the session management stack in a PAM configuration file includes a line-entry for the Platform Services PAM module (`pam_ascauth`), the following occurs:

- The function `pam_sm_open_session` is called to invoke the module; the `pam_handle` (unique session identifier and data) is included as an argument
- Instructions in the module are executed:
  - Check for inclusion of any options associated with this module (options are listed in section 5.4)
  - Confirm that all necessary inputs exist in `pam_handle`
- `pam_sm_open_session` sends a successful return code to PAM framework

Once a return code is sent to the PAM framework, the next line-entry in the PAM configuration file can be executed.

Note The session management module does not include any significant functionality. It is included only to ensure that all configration scenarios can be met. *Example:* Either a customer's procedure or a particular system-entry application could require all four stacks in the PAM configuration file(s) to include a line-entry.

5.3.5 Summary

The following table summarizes the tasks carried out by the PAM module that comes with Platform Services for Linux and UNIX.

PAM service	What the module does
Authentication	• If password was not yet submitted, prompt user • Redirect user name, password to Identity Vault (eDirectory) via Platform Services/Core Driver • If Identity Vault returns a go, account data is also included from Identity Vault for later use • If Identity Vault returns a no-go, prompt again or quit, depending on line-entry option • Send successful or unsuccessful return code
Account access	• If user is `root`, quit; return code is successful • If authentication module had unsuccessful return code, quit; return code says ignore this module • If `pam_handle` says account is disabled, quit; return code is unsuccessful • If `pam_handle` says account is expired, quit; return code says run password change module • Otherwise, send a successful return code
Password change	• If Sun Solaris and `passwd`???, verify account existence in Identity Vault • If user is `root` and system-entry application is `passwd`: quit; return code is successful • `pam_sm_ chauthtok` sends a successful return code to PAM framework • If `pam_handle` says account does not exist in Identity Vault, quit; return code is unsuccessful • If user is `root` and the system-entry application is `passwd`: quit; `pam_sm_acct_mgt` sends return code to proceed as if this module does not exist • Prompt user for old and new passwords, using settings in `asamplat.conf` • If not `passwd`, look in `pam_handle` to check if authentication was also processed through the PAM module that comes with Identity Manager; if no, quit; `pam_sm_acct_mgt` sends unsuccessful return code • Check `pam_handle` to see if password must have previously expired to change • Synchronize local system account database with Identity Vault if required • Synchronize local SAMBA password with Identity Vault if required • `pam_sm_authenticate` sends a second successful return code to PAM framework, which enables the password to be reset
Session management	• Check for any options associated with this module • `pam_sm_open_session` sends a successful return code to PAM framework

5.4 Activating Platform Services with your PAM configuration file(s)

The presence of Platform Services on a Linux or UNIX system is not enough to make PAM work with Identity Manager. This functionality must be activated by adding the appropriate line-entries to your PAM configuration file(s).

There are three methods you can follow when making these changes, two of which involve a set of PAM configuration file templates that are part of the Platform Services installation. Here is where those templates are located:

Operating system	PAM templates location
AIX	`/usr/lib/security/???`
HP-UX	`/usr/lib/security/???`

Operating system	PAM templates location
Linux	`/lib/security/???`
Solaris	`/usr/lib/security/???`

You will need to determine the method best suited for updating PAM on each of your systems based on your current setup. The following table summarizes the three methods:

Method	Description	When to use	Applications redirected to Identity Manager
1 Replace current configuration file(s) with provided template(s).	From the pre-configured templates that come with Platform Services, select the one(s) for your specific implementation of Linux or UNIX. Use it/them in place of your current PAM configuration file(s).	If you have never modified the default PAM configuration file(s) that came with your implementation of Linux or UNIX, then this easiest method should work for you.	`ssh`, `passwd`, `su` (default configuration)
2 Transfer line-entries from provided template(s) to current configuration file(s).	From the pre-configured templates that come with Platform Services, select the one(s) for your specific implementation of Linux or UNIX. Copy and paste the line-entries into your current PAM configuration file(s).	If you have modified the default PAM configuration file(s) that came with your implementation of Linux or UNIX, then use this method to keep intact any line-entries you have have already added.	`ssh`, `passwd`, `su` (default configuration)
3 Manually customize current configuration file(s) beyond default system-entry support.	Compose and insert as many line-entries as you need.	If you want Identity Manager to work with more system-entry applications than those included in the templates, then use this method to add the appropriate line-entries for as many applications as you wish.	Any you decide to add

6.0 ADDITIONAL INFORMATION

6.1 History

PAM was created by Sun Microsystems. It first appeared as a public interface in Solaris 2.6. Previous versions of a very simlar framework were in Solaris 2.4 and Solaris 2.5.1.

Sun proposed PAM in an Open Software RFC dated October, 1995. It was adopted as the authentication framework of the Common Desktop Environment. Open-source PAM first appeared in Red Hat Linux 3.0.4 in August 1996. PAM was later standardized as part of the X/Open UNIX standardization process, resulting in the X/Open Single Sign-on (XSSO) standard.

6.2 Finding more information

Following are some of the publicly available resources for PAM.

- For PAM configuration file information specific to your Linux/UNIX implementation, see the online man pages, typically accessed by entering `man pam.conf`.
- One of the few commercially printed books devoted soley to PAM is *The Definitive Guide to PAM for Linux SysAdmins and C Developers*, by Kenneth Geisshirt, 2007, Packt Publishing Ltd. An online copy of this publication is available at ://ftp.ofloo.net/pub/howtos/dev/Packt.Publishing.Pluggable.Authentication.Modules.Dec.2006.pdf.
- For Linux-PAM documentation on the Web, go to ://www.kernel.org/pub/linux/libs/pam/ and refer to "The Linux PAM Guides" as an authoratative reference.
- For Sun's documentation on Solaris PAM, go to ://www.sun.com/software/solaris/pam/.
- To search for information specific to HP-UX PAM, go to ://docs.hp.com/en/index.html.
- More Web links are available on the Wikipedia.org page devoted to PAM at ://en.wikipedia.org/wiki/Pluggable_Authentication_Modules

For more information on Novell Identity Manager and other Novell products, visit .Novell.com, which includes a comprehensive online documentation library at ://www.novell.com/documentation.

7.0 GLOSSARY

bidirectional driver One of two types of drivers available to customers who want to connect a Linux/UNIX system to Novell Identity Manager. Extends Identity Manager's dual-channel (publisher and subscriber) architecture to one connected system.

Core Driver Component in the Fan-Out Driver that interfaces with Identity Manager using the subscriber channel to receive updates and LDAP to return information directly to the Identity Vault. One Core Driver can support up to 100 systems (with 100 individual installations of Platform Services).

eDirectory Directory services database from Novell that represents and controls every user, group and device in a network through a hieararchy tree of objects and attributes. Designed to the X.500 standard for directory services and therefore supports LDAP communication protocol.

Fan-Out Driver One of two types of drivers available to customers who want to connect a Linux/UNIX system to Novell Identity Manager. The Fan-Out Driver can work for Identity Manager as an interface to many systems—up to 100 systems per driver. It accomplishes this scalability by separating its functions into two basic components: Core Driver and Platform Services.

Identity Manager The comprehensive software solution from Novell that allows organizations to manage the full user lifecycle, from initial hire, through ongoing changes, to ultimate retirement of the user relationship. A foundation for account provisioning, security, user self-service, authentication, authorization, automated workflow and Web services across all platforms, including Linux and UNIX.

Identity Vault An integrated object tree in eDirectory that reflects a full Identity Manager solution, including all users, groups, and the application components that manage them.

line-entry A single line of text in a PAM configuration file. The PAM library uses the line-entries as instructions about which modules to invoke for each system-entry application. Line-entry syntax includes five tokens: service-name, module-type, control-flag, file-path, option.

PAM (Pluggable Authentication Modules) A one-off interface that all system-entry applications in Linux and UNIX defer to for precise control of system-entry. By offering modules that plug in to the PAM framework, any developer can add customized levels of security in compiled programming code that executes in tandem with any of the standard system-entry applications.

PAM API (Application Programming Interface) A group of PAM library functions that enable the system-entry application to communicate with the library. The API is open to customers for integration with local applications.

PAM configuration file(s) A single file or group of files containing simple rules about which modules should be invoked for each system-entry application. These files contain the rules by which the PAM library should link to modules to execute their compiled code.

PAM framework General name given to the five main parts comprising PAM: the library, the API, the SPI, modules, and one or more confinguration files. The framework provides a uniform environment for system-entry information to be exchanged, evaluated and acted upon.

PAM item A standard place-holder used to move data between system-entry applications and PAM modules.

PAM module Software instructions stored in shared library files. Up to four modules can exist in each shared library—one for each of the four services PAM can manage (authentication, account access, password change, and session management). Modules communicate with the library through the PAM SPI.

PAM service One of four areas of processing PAM can provide to system-entry applications. The four services (authentication, account access, password change, and session management) are performed by PAM modules.

PAM return code Response to PAM library after a PAM module executes. The numerous possible return codes can be grouped into three basic categories: `PAM_SUCCESS`, `PAM_IGNORE`, and all other codes.

PAM SPI (Service Provider Interface) A group of PAM library functions that enable the library to communicate with the PAM modules.

Platform Services A set of software components installed on each system that is being synchronized with Identity Manager through the Fan-Out Driver. Different versions of Platform Services are available for Linux, UNIX, z/OS (IBM mainframes), and i5/OS (OS/400). The version of Platform Services that is used on Linux and UNIX includes software components that can work with PAM.

shared library, shared object file The container for a PAM module is a shared object (`.so`) file—also commonly known as a shared library file. Shared library files are compiled files that executable programs (such as `telnet`, `ftpd` and `passwd`) can dynamically link to at run time to follow additional functions and subroutines. Each PAM-module shared library can actually contain up to four separate modules—one for each PAM service (authentication, account access, password change, session management).

stack, stacking Terms used to help describe the sequence in which line-entries in a PAM configuration file are processed. A stack represents all line-entries of a one of the four possible module-types (authentication, account access, password change, session management).

system-entry application Any application, such as `passwd`, `ssh`, `ftp`, `telnet`, that enables a user to request entry to the Linux or UNIX host system. System-entry applications communicate with the PAM library through the PAM API.