Die Pforten zur Linux-Maschine: Pluggable Authentication Modules

# Hereinspaziert

Pluggable Authentication Modules geben dem Administrator die Freiheit, in jedem PAM-tauglichen Programm beliebige Authentifizierungsverfahren einzusetzen und netzwerkweit einheitliche Benutzerkennungen zu verwenden. Auch die User freut's, sie müssen sich nur noch einen Account merken. Dirk von Suchodoletz, Martin Walter

**Beim Thema** Authentifizieren am Linux-Rechner denken die meisten Admins am ehesten an die Systemdateien »/etc/passwd« und »/etc/shadow«. Deren Konzept ist so alt wie Unix und erfüllt heute noch seinen Zweck. Inzwischen erledigen Linux-Maschinen aber viele Aufgaben – vom Web-, FTP- oder Fileserver über Mailserver und Proxy bis hin zum CVS-Server –, die ebenfalls ihre Benutzer authentifizieren müssen. Häufig bringen diese Dienste ihre eigene Userverwaltung mit. Allerdings verlieren Admins schnell den Überblick und die Anwender das Verständnis, wenn jeder Dienst eine eigene Benutzername-Passwort-Kombination verlangt.

Zu dieser Entwicklung gesellen sich weitere Neuerungen: Die Tastatur hat ihre Rolle als alleiniges Eingabemedium für die Zugangskennung verloren. Chipkarten, biometrische Merkmale oder Zertifikate auf einem USB-Stick kommen vermehrt zum Einsatz. Manche Systembetreiber möchten einen Login nur zu bestimmten Zeiten erlauben. Auch für das Mounten des Homeverzeichnisses kann eine Authentifizierung nötig sein.

## Zu viel für das alte System

Mit diesen Aufgaben ist das herkömmliche System überfordert: Wenn sich jede Anwendung selbst um Identifizierung und Authentifizierung ihrer Nutzer kümmert, steigt der Programmieraufwand unnötig. Für jede neue Authentifizierungsmethode muss jedes Programm (im Sourcecode) angepasst werden. Mögliche Fehler im Login-Verfahren sind in jedem Programm zu beheben – Anwendungsentwickler kümmern sich aber selten um Neuerungen und Hintergründe der Sicherheitstechniken.

PAM lagert daher die Authentifizierung in eine modulare Bibliothek aus. Vorteil: Die Anwendung muss sich nicht mehr um diese Aufgaben kümmern. Neue Verfahren lassen sich einbinden, ohne die Applikation zu ändern. Auch ist die Konfiguration einheitlich – egal ob Konsolen- oder X11-Login, Telnet, FTP oder SSH. PAM kann sogar mehr als die klassische Passwortdatei-Variante. Neben dem Authentication-Management übernimmt es auch das Account-, Session- und Password-Management.

## PAM und die Applikation teilen sich die Arbeit

Da die PAM-Bibliothek jedoch keine Mechanismen besitzt, um Benutzereingaben oder Authentifizierungs-Token entgegenzunehmen, muss sich die Applikation darum kümmern. Nur die Applikation selbst weiß, auf welchem Weg sie zu diesen Daten kommt. Das Lesen von der Rechnerkonsole, aus einer Netzwerkverbindung oder vom Displaymanager-Login implementieren die jeweiligen Programme sowieso.

Eine wichtige Einschränkung hat PAM jedoch: Es ist rein passiv und muss immer von einer Applikation aufgerufen werden. PAM kann den Benutzer daher nicht automatisch einloggen, wenn er seine Chipkarte einsteckt oder den Finger auf einen biometrischen Leser legt. Erst muss die Applikation aktiv werden und PAM anstoßen.

Die schematische Darstellung in Abbildung 1 zeigt, wie der Kommunikationsfluss zwischen Applikation und PAM funktioniert. Jedes Programm lädt bei Bedarf die PAM-Bibliothek. Diese Library wiederum greift auf PAM-Module

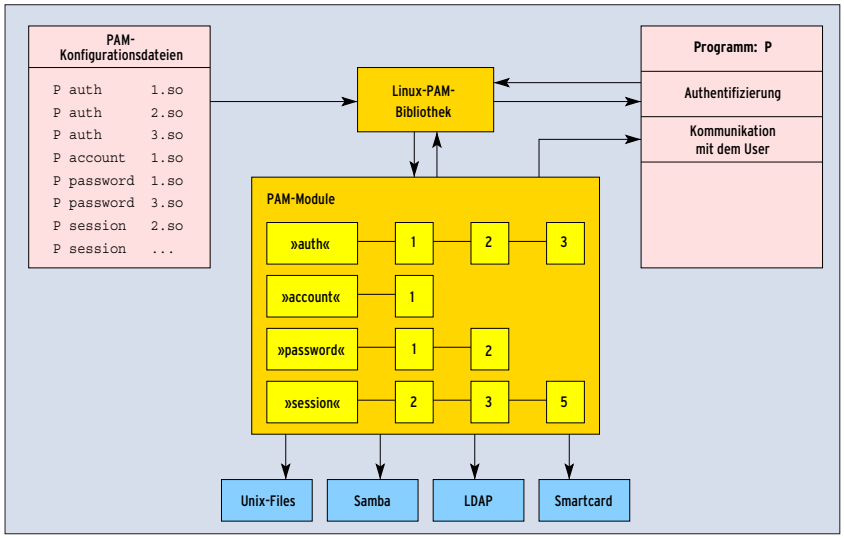


**Abbildung 1: Das Programm P (rechts) authentifiziert seine Benutzer mit Hilfe der PAM-Bibliothek (Mitte). Diese Library liest ihre Konfigurationsdatei (links), lädt die dort aufgeführten Module und greift bei Bedarf auf weitere Dienste wie Samba oder LDAP zu (unten).**

zurück. Jede Applikation kann ihre eigene PAM-Konfiguration erhalten. Das PAM-Modul kommuniziert mit der PAM-Bibliothek, um seine Parameter auszulesen, und mit der Applikation, um an die Benutzerdaten wie Accountname und Passwort zu gelangen. Manche Module greifen zudem auf vorhandene Dienste zu oder lesen typische Unix-Dateien wie »/etc/passwd«.

## Im System versteckt

PAM ist für den Benutzer nie direkt sichtbar, es ist keine eigenständige Applikation. Die PAM-Basisbibliotheken sind normalerweise ein eigenes Paket in der Linux-Distribution. Es gibt spezielle PAM-Module, die als Backend eine Applikation oder ein Protokoll benötigen. Diese sind entweder zusammen mit dem Backend (üblich etwa bei AFS, dem Andrew File System) oder als zusätzliches Paket zu installieren (zum Beispiel PAM-Samba oder PAM-LDAP).

Die PAM-Installation verteilt sich in die drei Verzeichnisse »/lib/security«, »/etc/security« und »/etc/pam.d«. In »/lib/security« landen die Bibliotheksdateien, dieses Verzeichnis enthält alle auf einem System installierten Standard-PAM-Module. Die Dateinamen beginnen mit »pam_« und enden auf ».so«, zum Beispiel »pam_unix.so«. Dieses Modul bildet den klassischen Unix-Passwortmechanismus nach.

Einige Module verfügen über eigene Konfigurationsdateien, die ihr Verhalten unabhängig von der aufrufenden Applikation steuern. Sie liegen unterhalb von »/etc/security«. So ist das File »/etc/security/limits.conf« für das Modul »/lib/

**Tabelle 1: Modultyp**

| Typ | Bedeutung |
|---|---|
| auth | Benutzeridentifizierung und -authentifizierung, sie kann zum Beispiel durch klassische Benutzername-Passwort-Abfragen, biometrische Verfahren oder Smartcards mit PIN erfolgen. Voraussetzungen sind passende Schnittstellen der angeschlossenen Geräte, etwa der Smartcard-Leser. |
| account | Diese Module verwalten den Zugriff auf Accounts unabhängig von der Authentifizierung. So kann ein Dienstbetreiber den Zugriff abhängig von der Uhrzeit oder dem Wochentag steuern. |
| password | Module diesen Typs erlauben es dem Benutzer, sein Passwort oder Token zu aktualisieren. Verwendet das »passwd«-Programm die PAM-Bibliotheken, kann der Admin festlegen, welche Passwörter das Programm ändert und akzeptiert. |
| session | Verwalten der Einstellungen für die Sitzung des Benutzers. Sie können die im System verbrachte Zeit abrechnen oder Limits und Zugriffsberechtigungen auf Unix-Devices setzen sowie zusätzliche Dateisysteme mounten. |

security/pam_limits.so« zuständig. Zusammen regeln sie, wie viele Systemressourcen (etwa Speicher oder CPU-Zeit) dem Benutzer zur Verfügung stehen.

## Für jeden Dienst eine eigene Konfiguration

Für die dienstspezifische Konfiguration der PAM-Bibliotheken gibt es zwei Möglichkeiten: Entweder legt der Administrator alle Informationen in dem gemeinsamen File »/etc/pam.conf« ab oder er definiert für alle Dienste und Applikationen je eine eigene Konfigurationsdatei unterhalb von »/etc/pam.d«. Letzteres ist wegen der Fülle der Dienste bei den meisten Distributionen Standard. Diese Variante hat den Vorteil, dass eine Installationsroutine für neue Pakete nur eine zusätzliche PAM-Konfigurationsdatei anlegen muss.

Jede Datei in »/etc/pam.d« ist für je einen PAM-fähigen Dienst zuständig. Da unterschiedliche Dienste ja auch unterschiedliche Anforderungen haben, erleichtert eine eigene Konfigurationsdatei den Überblick: Login sollte feststellen, ob der Systemadministrator sich von einem sicheren Terminal aus anmeldet.

**Listing 1: Typische PAM-Konfigurationsdatei**

```
01 # Modultyp  Modulsteuerung  Modulpfad       Argumente
02 auth        required         pam_unix2.so    nullok
03 account     required         pam_unix2.so
04 password    required         pam_unix2.so
   strict=false
05 session     required         pam_unix2.so    debug
06 session     required         pam_devperm.so
07 session     required         pam_resmgr.so
```

**Listing 2: PAM-Konfiguration für SSH**

```
01 #%PAM-1.0
02 auth       required    pam_nologin.so
03 auth       sufficient  pam_unix2.so     set_secrpc
04 auth       required    pam_ldap.so      use_first_pass
05 auth       required    pam_storepw.so   use_first_pass
06 account    required    pam_unix2.so
07 password   required    pam_pwcheck.so
08 password   required    pam_ldap.so      use_authtok
09 password   required    pam_unix2.so     use_first_pass
   use_authtok
10 session    required    pam_unix2.so
11 session    required    pam_limits.so
12 session    required    pam_env.so
13 session    optional    pam_mail.so
```

Bei der SSH ist dies dagegen nicht sinnvoll. Hier legt der Admin in »sshd_config« fest, ob sich der Root-User überhaupt anmelden darf.

Sinnvollerweise trägt die Konfigurationsdatei den gleichen Namen wie der Dienst. Leider stimmt das nicht immer. Einige Netzwerkdienste verzichten auf das abschließende »d« wie etwa beim »ppp«. Oder sie verwenden abweichende Namen, etwa »samba« für den »smbd«. Findet es keine passende Konfigurationsdatei, greift PAM auf »/etc/pam.d/other« zurück. Da der Administrator kaum wissen kann, welcher Dienst eventuell diese Datei benutzt, stellt er sie am besten ziemlich restriktiv ein.

## Die vier Modultypen

Alle PAM-Konfigurationsdateien haben einen ähnlichen strukturellen Aufbau (Listing 1). Der Modultyp (erste Spalte) bestimmt, welche Managementfunktion ein Eintrag erfüllt. PAM kennt die vier Modultypen »auth«, »account«, »password« und »session«, siehe Tabelle 1. In die »auth«-Kategorie fallen auch spezielle Module, die zwar Benutzerkennung und Passwort abgreifen, aber keinen Login ausführen. Sie mounten beispielsweise das Homeverzeichnis von einem Samba-Server oder holen ein AFS-Token. Das Beispiel am Ende des Artikels fällt ebenfalls in diese Kategorie.

## Modulsteuerung

Die Spalte »Modulsteuerung« in Listing 1 legt das Verhalten von PAM fest. Sie bestimmt, was passiert, wenn ein Modul mit einer Erfolgs- oder einer Fehlermeldung endet (siehe Tabelle 2 sowie Abbildungen 2a und 2b). PAM kennt vier Standardbedingungen, die der Admin bei Bedarf verfeinern kann. Dieses Verfahren erlaubt so genannte Modulstapel, auch als Stacking bezeichnet. PAM arbeitet die Module in der Reihenfolge ab, in der sie in der Konfigurationsdatei gelistet sind. Unter bestimmten Bedingungen erreicht es weiter unten stehende Module aber nicht mehr.

Die Applikation ruft – je nach Bedarf – einen der vier Modultypen auf. Die PAM-Bibliothek geht daraufhin alle Module, die in der Konfiguration zu diesem Typ vermerkt sind, nacheinander durch.

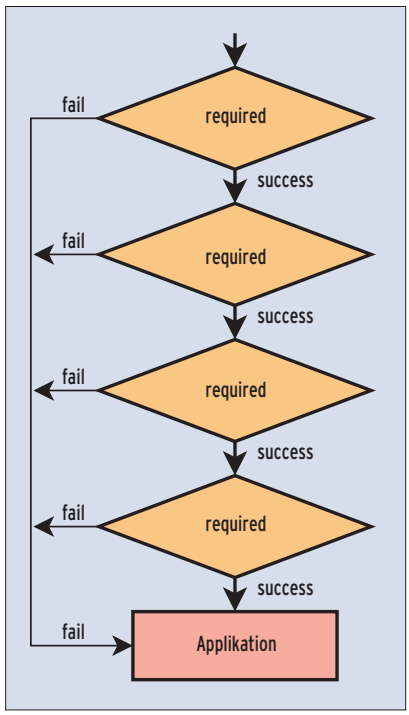


**Abbildung 2a:** Sind mehrere Module als »required« hintereinander geschaltet, dann muss jedes erfolgreich durchlaufen, um zu einem positiven Gesamtergebnis zu kommen. Sobald ein Modul fehlschlägt, lautet das Resultat »fail«.

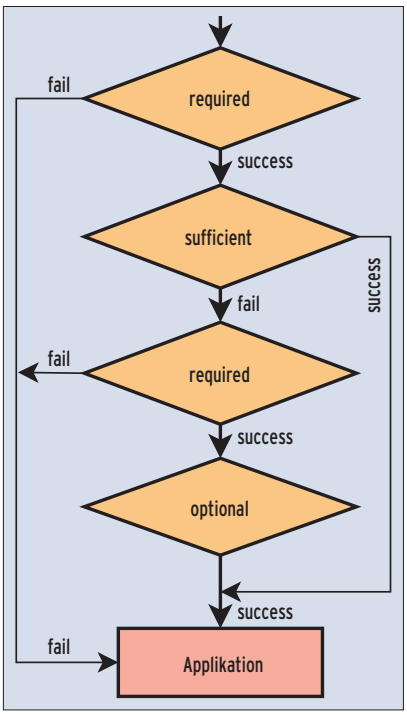


**Abbildung 2b:** Sobald ein »sufficient«-Modul das Ergebnis »success« meldet, bricht PAM den Durchlauf ab und meldet den Erfolg an die Applikation. Das Ergebnis von »optional«-Modulen ignoriert PAM dagegen in den meisten Fällen.

Hat sie das letzte Modul erreicht oder vorzeitig den Vorgang abgebrochen, liefert sie das Gesamtergebnis an die aufrufende Applikation zurück. Das Ergebnis zeigt, ob ein Erfolg (»success«) oder Misserfolg (»fail«) eingetreten ist.

Der Modulpfad in der nächsten Spalte legt fest, wo PAM nach der entsprechenden Teilbibliothek sucht. Standardmäßig erwartet es seine Module in »/lib/security«. In diesem Fall genügt der Modulname. Andernfalls gilt das klassische Unix-Prozedere – PAM interpretiert Pfadnamen mit »/« beginnend als absolut und andere als relativ zu »/lib/security«.

## Gute Argumente

Die letzte Spalte (Argumente) ist optional und enthält nur bei einigen Modulen einen Eintrag. Argumente sind als Liste anzugeben, deren Felder durch Leerzeichen getrennt sind. Es gibt einfache Flags wie »nullok« oder »use_first_pass« sowie Zuweisungen, zum Beispiel »strict=false«. Kommen Leerzeichen in einem Argument vor, ist es in eckige Klammern zu setzen. Soll eine eckige Klammer Bestandteil des Arguments sein, führt ein Backslash-Quoting zum Erfolg.

Die Liste möglicher Argumente ist nicht von PAM vorgegeben, sondern abhängig vom Modul. Manche Argumente werden aber von fast allen Modulen verstanden: »debug« liefert Diagnosemeldungen an das Systemlog, »no_warn« unterdrückt sie. Authentifizierungs- und Passwortmodule kennen zudem »use_first_pass«. Mit dieser Option versucht das Modul, das Passwort vom vorhergehenden »auth«-Modul zu übernehmen. Schlägt dies fehl, meldet es den Status »fail«. Ähnlich wirkt »try_first_pass«, nur fordert es im Fehlerfall die Applikation und damit den Benutzer auf, das Passwort erneut einzutippen.

Falsch eingetragene Argumente ignoriert PAM. Bei falsch formatierten Zeilen sorgt es sicherheitshalber dafür, dass das betroffene Modul die Meldung »fail« als Statuswert zurückgibt.

## Für jeden Zweck ein passendes PAM-Modul

Inzwischen gibt es PAM-Module für fast alle Aufgaben, manchmal sogar verschiedene Implementation für dieselbe Aufgabe. Das PAM-Standardpaket bringt schon eine Menge Module mit. Nur für spezielle Aufgaben wie LDAP-Authentifizierung oder um AFS-Token zu beschaffen sind zusätzliche PAM-Module nötig. Tabelle 3 stellt die gängigen Module zusammen. Darüber hinaus gibt es noch eine ganze Reihe weiterer Module für bestimmte Einsatzszenarien.

Modulen im Betastadium sollte der Admin lieber mit Vorsicht begegnen. Zu dieser kritischen Kategorie zählt zum Beispiel »pam_storepw«, ein Passwort-Caching-Modul. Es arbeitet in der Auth-Kategorie und kann im Stapel mit anderen Authentifizierungsmodulen Benutzername und Passwort abgreifen.

Diese Geheimnisse landen als Klartext in einem für Root lesbaren Verzeichnis »/var/run/pw«. Mit diesen Daten kann der Automounter geschützte Samba-Shares einbinden, die ebenfalls Benutzername und Passwort erfordern. Das Verfahren ist zwar ziemlich riskant,

**Tabelle 2: Modulsteuerung**

| Steuerung | Bedeutung |
|---|---|
| required | Legt fest, dass das Modul mit dem Status »success« enden muss. Nur dann kann das Gesamtergebnis aller Module dieses Typs auf Erfolg lauten. Auch bei einem Misserfolg des Moduls geht PAM die weiteren Module dieser Kategorie durch. |
| requisite | Verhält sich ähnlich wie »required«, gibt die Kontrolle nach einem Fehlschlag aber sofort an die Anwendung zurück. Der Rückgabewert ist der des ersten fehlgeschlagenen Moduls mit Required- oder Requisite-Steuerung. |
| sufficient | Wenn dieses Modul »success« meldet, ist das ausreichend für eine positive Gesamtmeldung. Es dürfen jedoch vorher keine Required- oder Requisite-Module fehlgeschlagen sein. PAM greift nach einem erfolgreichen »sufficient« auf keine weiteren Module mehr zurück. Nur wenn das Modul keine Erfolgsmeldung liefert, ruft PAM die nachfolgenden Module auf. |
| optional | Hat mit seinem Erfolg oder Misserfolg in der Regel keinen Einfluss auf den Gesamtrückgabewert. Nur wenn alle anderen Module im Stapel weder Erfolg noch Misserfolg melden, nimmt PAM das Ergebnis eines »optional«-Moduls. Achtung: Das kann bei falscher Anwendung leicht zu Sicherheitslücken führen. |

»pam_storepw« liefert jedoch eine hervorragende Programmiervorlage für eigene PAM-Erweiterungen.

## PAM-Konfiguration für den SSH-Daemon

Die Datei »/etc/pam.d/sshd« (Listing 2) zeigt ein typisches Beispiel einer PAM-Konfiguration. Sie schaltet vier Authentifizierungsmodule hintereinander: Das erste Modul überprüft – wenn die User-ID nicht 0 ist –, ob eine Datei »/etc/nologin« existiert. Es unterbindet in diesem Fall eine weitere Anmeldung, da das Fehlschlagen eines »required«-Moduls zum Gesamtergebnis »fail« führt. Dies meldet PAM als Gesamtwert an die Applikation zurück.

In Zeile 3 versucht PAM, den Benutzer gegen die Standard-Unix-Dateien zu authentifizieren. Gelingt dies, arbeitet es keine weiteren Module mehr ab. Schlägt dieser Schritt fehl, nimmt PAM einfach das nächste Modul mit der bereits eingegebenen Passwort-Benutzername-Kombination. Klappt die Anmeldung des Benutzers gegen den LDAP-Server, kommt »pam_storepw« (Zeile 5) zum Einsatz. Es erledigt keine Authentifizierung, sondern schreibt Benutzername und Passwort in eine Datei.

Beim SSH-Daemon ist zusätzlich darauf zu achten, dass er die PAM-Authentifizierung überhaupt erlaubt. In der SSH-Konfigurationsdatei »/etc/ssh/sshd_config« muss dazu der Eintrag »UsePAM yes« stehen. Die vorgestellte Konfiguration eignet sich für Umgebungen, in denen es bis auf den Administrator keine lokal eingetragenen User gibt. Sie versucht zunächst eine Authentifizierung mit den klassischen Unix-Dateien. Dies Verfahren ist allerdings nur bei Root erfolgreich.

## Lokale Accounts erkennen

Da der Root-Account nicht im LDAP gespeichert ist und kein zentrales Homeverzeichnis besitzt, darf PAM bei diesem Benutzer auch nicht mehr bei den auf »pam_unix2.so« folgenden Modulen vorbeikommen. Das wird von der Modulsteuerung »sufficient« verhindert. Entscheidend ist, dass »pam_unix2.so« nicht so konfiguriert ist, dass es selbst

**Tabelle 3: PAM-Module**

| Modul | Aufgabe |
|---|---|
| pam_access | Stellt eine Art Access Control List zur Verfügung. Das Modul erwartet eine Konfigurationsdatei »/etc/security/access.conf«. Dort pflegt der Admin »*Erlaubnis : Anwender : Ursprung*«-Listen. |
| pam_afs | Dieses Authentifizierungsmodul kennt zwei Betriebsarten. Entweder authentifiziert ein anderes Modul den Benutzer, dann holt »pam_afs« nur noch das Token vom AFS-Server. Oder »pam_afs« authentifiziert den User ebenfalls selbst. |
| pam_chroot | Dieses Modul enthält Account-, Session- und Authentifizierungskomponenten. Es versetzt Benutzer beim Login in einen Change-root-Käfig. |
| pam_cracklib | Setzt Richtlinien für gute Passwörter durch, Benutzer können damit keine erkennbar schlechten Passwörter verwenden. Das Modul arbeitet in der Kategorie »Password«. Es benötigt die Cracklib und passende Wörterbücher. |
| pam_deny | Verhindert immer den Zugriff, dazu sendet das Modul eine entsprechende Rückmeldung an die aufrufende Applikation. |
| pam_devperm | Passt die Zugriffsrechte auf Device-Files an. Damit dürfen lokal eingeloggte User zum Beispiel das Floppy-Laufwerk und die Soundkarte nutzen. Es liest die Konfigurationsdatei »/etc/logindevperm«. |
| pam_env | Passt die Umgebungsvariablen an. Es benutzt dazu die Konfigurationsdatei »/etc/security/pam_env.conf«. Diese kann festlegen, welche »DEFAULT«-Werte für eine Variable gelten oder wie diese überschrieben (»OVERRIDE«) werden sollen. |
| pam_lastlog | Kümmert sich als Modul der Session-Kategorie um die Datei »/var/log/lastlog« und trägt dort den Benutzer ein. Es kann, wenn das nicht schon die Applikation erledigt, auch anzeigen, wann der Nutzer das letzte Mal angemeldet war. |
| pam_ldap | Benutzerauthentifizierung gegen eine LDAP-Datenbank. Je nachdem, wie dieses Modul kompiliert wurde, bezieht es seine LDAP-Konfiguration aus »/etc/ldap.conf« oder »/etc/openldap/ldap.conf«. |
| pam_limits | Erlaubt es dem Systemadministrator festzulegen, wie viel Systemressourcen (Speicher, CPU-Zeit ...) dem Benutzer maximal zur Verfügung stehen. Diese Einschränkungen gelten nicht für User mit der ID 0. Die Konfigurationsdatei für dieses Modul ist »/etc/security/limits.conf«. |
| pam_mkhomedir | Legt bei Bedarf ein Homeverzeichnis für den Benutzer an. |
| pam_mktemp | Kann als Teil des Session- oder Accountmanagements für angemeldete Benutzer ein persönliches Temporärverzeichnis festlegen. Hierzu setzt es die Umgebungsvariablen »TMPDIR« und »TMP«. |
| pam_nologin | Prüft, ob die Datei »pam_nologin« existiert. Wenn ja, zeigt es jedem Benutzer (bis auf den Systemadministrator) den Inhalt dieses Files und verhindert, dass sich der User anmeldet. |
| pam_permit | Liefert immer ein fröhliches »sucess«. Dieses Modul sollte nur unter sehr speziellen Bedingungen zum Einsatz kommen. |
| pam_pwcheck | Stellt Hilfsfunktionen in der »password«-Kategorie für die Passwortänderung zur Verfügung. Es verwendet Einstellungen aus »/etc/login.defs«. Hier kann der Admin mittels Modulargument festlegen, nach welchem Crypt-Verfahren (»md5«, »blowfish« ...) das Passwort verschlüsselt wird. |
| pam_rootok | Ein Modul der Authentifizierungskategorie. Es kann für bestimmte Dienste (wie »su«) dem Admin einen Login ohne Passwort erlauben. Auf diese Weise kann Root zu einem normalen User werden, ohne dessen Passwort wissen zu müssen. |
| pam_securetty | Stellt fest, von welchem Terminal (TTY) sich der Systemadministrator anmelden darf. Die Datei »/etc/securetty« enthält hierfür eine Liste von Devices, von denen aus sich Root anmelden kann. |
| pam_unix | Das Standard-Unix-Authentifizierungsmodul, das ebenfalls Komponenten für »session«, »password« und »account« zur Verfügung stellt. Es benutzt die Standardaufrufe der Systembibliotheken und beschafft Informationen aus »/etc/passwd« und »/etc/shadow«. |
| pam_warn | Hat die Aufgabe, Log-Output zu generieren und an den Syslog weiterzugeben. Es arbeitet in den Kategorien »authentication« und »password«. |
| pam_winbind | Kommt mit der Samba-Suite und erlaubt eine Authentifizierung gegen Windows-Systeme. |

gegen LDAP authentifiziert – siehe »/etc/security/pam_unix2.conf«:

```
auth:      nullok
account:
password:  nullok
session:   none
```

Das Modul »pam_unix2.so« kommt in Listing 2 zusätzlich als Account-, Passwortänderungs- und Session-Modul zum Einsatz. Die Accountfunktion dieses Moduls prüft anhand der Felder in der Shadowdatei, ob diese Benutzerkennung noch gültig oder ob eventuell das Passwort abgelaufen ist. In diesem Fall kann es die Authentifizierung verschieben, bis der Benutzer sein Passwort aktualisiert hat. Oder es kann eine Warnung an den Benutzer ausgeben, dass er sein Passwort ändern möchte.

Als Session-Modul protokolliert in Zeile 10 »pam_unix2.so« den Benutzernamen und den Diensttyp per Syslog. Das Modul »pam_env.so« setzt Umgebungsvariablen. In Zeile 13 ist »pam_mail.so« als »optional« angegeben, darf also ohne Konsequenzen fehlschlagen. Es dient lediglich dazu, dem Benutzer mitzuteilen, ob neue Mail lokal auf der Maschine für ihn vorliegt.

## Sicherheit und Fallstricke

PAM bewacht die verschiedenen Eingänge des Rechners und ist deshalb ein wesentlicher Baustein der Sicherheitsrichtlinien. Eine Fehlkonfiguration kann dazu führen, dass sich Benutzer auch ohne ausreichende Authentifizierung anmelden können. Das illustriert das Beispiel in Listing 3.

Hier sind wieder vier Module hintereinander geschaltet (Stacking). Das erste Modul überprüft, ob »/etc/nologin« existiert. Gibt es diese Datei nicht, liefert es den Status »success«. Selbst wenn nun alle drei folgenden Module fehlschlagen, wird das Gesamtergebnis »success« nicht mehr angezweifelt. Auf diese Weise kann sich ein Benutzer auch mit falschem Passwort anmelden. Noch radikaler wäre der Einsatz des »pam_permit«-Moduls, das immer einen Erfolg der Anmeldung garantiert.

Umgekehrt kann eine Fehlkonfiguration auch komplett verhindern, dass sich irgendjemand an der Maschine anmeldet, siehe Listing 4. Die Kontrolleinstellung »required« sorgt dafür, dass ein Benutzer sowohl in den Unix-Dateien »passwd« und »shadow« als auch im LDAP bekannt sein muss. In verteilten Authentifizierungsarchitekturen liegt genau dies nicht vor: Der Systemadministrator ist immer nur lokal eingetragen, normale Benutzer sinnvollerweise nie. Damit ist jeder User – inklusive Root – ausgesperrt.

Je nach Konstruktion seines Systems sollte der Systemverwalter besonders auf den »su«-Mechanismus achten. In großen Umgebungen traut er vielleicht den lokalen Administratoren nur bedingt und will vermeiden, dass sie in die Homeverzeichnisse beliebiger Nutzer hineinsehen. Eine zusätzliche Authentifizierung zusammen mit AFS oder Samba helfen ihm dabei. Er muss nur einstellen, dass Root sich nicht ohne Passwort in zentral verwaltete Benutzer verwandeln darf. Das Homeverzeichnis steht ohne entsprechende Credentials nicht mehr automatisch zur Verfügung.

## Auf die Rechte achten

Für alle PAM-Dateien und -Verzeichnisse ist darauf zu achten, dass nur Root Schreibrechte besitzt und Eigentümer der Verzeichnisse »/lib/security«, »/etc/security« und »/etc/pam.d« inklusive der enthaltenen Dateien ist. Außerdem sollte man keine PAM-Module einsetzen, die von Benutzern schreibbare Programmbibliotheken verwenden. Sonst könnte es Angreifern gelingen, eigenen Code in PAM einzuschleusen und sich damit Root-Rechte zu verschaffen. Weiterhin setzt der sicherheitsbewusste Admin einen Fallback-Mechanismus für unkonfigurierte Dienste ein: »/etc/pam.d/other«, siehe Listing 5.

Beim Passwortwechsel prüft PAM laut Zeile 6 das neue Passwort per Cracklib, ansonsten meldet sich jeder Modultyp

**Tabelle 4: Rückgabewerte**

| Wert | Bedeutung |
|---|---|
| PAM_SUCCESS | Die Authentifizierung war erfolgreich. |
| PAM_AUTH_ERR | Der Benutzer konnte nicht authentifiziert werden. |
| PAM_CRED_INSUFFICIENT | Die aufrufende Applikation besitzt nicht genügend Rechte, um den Benutzer zu authentifizieren. |
| PAM_AUTHINFO_UNAVAIL | Auf die Authentifizierungsdaten konnte nicht zugegriffen werden (beispielsweise Datenbank nicht erreichbar). |
| PAM_USER_UNKNOWN | Der Benutzername ist nicht bekannt. |
| PAM_MAXTRIES | Zahl der Wiederholungsversuche ist überschritten. |

**Listing 3: Fehler: PAM zu freizügig**

```
01 auth  required  pam_nologin.so
02 auth  optional  pam_unix2.so    set_secrpc
03 auth  optional  pam_ldap.so     use_first_pass
04 auth  optional  pam_storepw.so  use_first_pass
05 [...]
```

**Listing 4: Fehler: PAM zu restriktiv**

```
01 auth  required  pam_nologin.so
02 auth  required  pam_unix2.so    set_secrpc
03 auth  required  pam_ldap.so     use_first_pass
04 auth  required  pam_storepw.so  use_first_pass
05 [...]
```

**Listing 5: Fallback-Mechanismus**

```
01 auth      required  pam_warn.so
02 auth      required  pam_unix2.so
03 account   required  pam_warn.so
04 account   required  pam_unix2.so
05 password  required  pam_warn.so
06 password  required  pam_cracklib.so
07 password  required  pam_unix2.so     use_first_pass
                                        use_authtok
08 session   required  pam_warn.so
09 session   required  pam_unix2.so
```

**Listing 6: Paranoider Fallback**

```
01 auth      required  pam_deny.so
02 auth      required  pam_warn.so
03 account   required  pam_deny.so
04 account   required  pam_warn.so
05 password  required  pam_deny.so
06 password  required  pam_warn.so
07 session   required  pam_deny.so
08 session   required  pam_warn.so
```

**Listing 7: Konfiguration des SSL-Mount-Moduls**

```
01 auth  required    pam_nologin.so
02 auth  sufficient  pam_unix2.so
03 auth  required    pam_ldap.so      use_first_pass
04 auth  sufficient  pam_sslmount.so
   /usr/local/bin/sslmount
```

mit einer Warnung beim Syslog und führt seine Funktion über das »pam_unix2.so«-Modul aus. Diese »other«-Konfiguration führt zu einer Warnung, erlaubt aber trotzdem das Anmelden unter einer gültigen User-ID.

Eine paranoide Einstellung des Other-Fallbacks könnte wie in Listing 6 gezeigt aussehen. In dieser Konstellation blockiert PAM jeden unkonfigurierten Dienst und schreibt zudem eine Warnung ins Syslog.

## PAM allein ist nicht genug

Wer PAM für Netzwerkdienste einsetzt, sollte wissen, dass PAM weder Passwörter noch Benutzernamen verschlüsselt. Sie gehen – etwa bei FTP – ungeschützt übers Netz. Das Verschlüsseln bleibt Aufgabe der Applikation, etwa SSH.

Nicht für alle Probleme des Admin-Alltags existieren fertige Lösungen. So liegen vielleicht die Homeverzeichnisse der Benutzer auf einem NFS-Server, der erst auf eine authentifizierte Anfrage des Clients einen Mount zulässt. Dieses Konzept wird beispielsweise im Rechenzentrum der Uni Freiburg genutzt, es schützt das unsichere Netzwerkdateisystem gegen unbefugten Zugriff.

## PAM-Module entwickeln

Ebenso könnte eine Maschine in einem abgesicherten IP-Netz stehen. Erst mit dem Login eines Users wird automatisch ein Passwort-authentifizierter Verschlüsselungstunnel zu einem Gateway aufgebaut und dann die Maschine mit dem Internet verbunden. Beide Fälle benötigen die Kombination aus Benutzername und Passwort. PAM könnte diese vom Benutzer erneut abfragen, das wäre jedoch unnötig umständlich und benutzerunfreundlich. Hier helfen angepasste Module weiter.

Eigene PAM-Module sollen Aufgaben in einem vorgegebenen Rahmen erfüllen. Dazu muss ein Modul ein bestimmtes Subset von sechs Funktionen beinhalten. Sie teilen die Managementaufgaben von PAM auf. Ein sauber aufgebautes Modul muss wenigstens für eine der Managementaufgaben alle Funktionen definieren, beispielsweise für die Authentifizierung die beiden vorgeschrieben Routinen »pam_sm_authenticate()« und »pam_sm_setcred()«.

Ein Implementierungsbeispiel ist in Listing 8 zu sehen. In Zeile 35 übernimmt die Funktion »pam_sm_authenticate()« die Aufgabe, einen Benutzer zu authen-

**Listing 8: PAM-Modul für SSL-Mount**

```
01 /* pam_sslmount.so 1.0
02  * (c) 2003 by Martin Walter <mawa@uni-freiburg.de>
03  * based on pam_storepw by F. Lohoff and pam_pwdfile.c by C. P. Botha
04  * needs external perl program sslmount
05  */
06
07 #define PAM_SM_AUTH
08 #include <security/pam_modules.h>
09 #include <syslog.h>
10 #include <stdarg.h>
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14 #include <errno.h>
15 #include <fcntl.h>
16 #include <sys/types.h>
17 #include <sys/stat.h>
18 #include <sys/wait.h>
19 #include <sys/file.h>
20 #define _XOPEN_SOURCE
21 #include <unistd.h>
22
23 /* logging function ripped from pam_listfile.c */
24 static void _pam_log(int err, const char *format, ...)
25 {
26    va_list args;
27    va_start(args, format);
28    openlog("pam_sslmount", LOG_CONS|LOG_PID, LOG_AUTH);
29    vsyslog(err, format, args);
30    va_end(args);
31    closelog();
32 }
33
34 /* expected hook for auth service */
35 PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char  **argv)
36 {
37    char  *passwd, *user;
38    int  p[2],status;
39    pid_t  pid;
40    if (argc < 1 || argv[0][0] != '/') {
41       _pam_log(LOG_ERR,"error: full path to sslmount expected as first argument");
42       return PAM_AUTH_ERR;
43    }
44    pam_get_item(pamh, PAM_AUTHTOK, (void *) &passwd);
45    pam_get_item(pamh, PAM_USER, (void*) &user);
46    if (!passwd || !user) {
47       _pam_log(LOG_ERR,"no password or user to write - got stacked wrong ?");
48       return PAM_AUTHINFO_UNAVAIL;
49    }
50    if (pipe(p)) {
51       _pam_log(LOG_ERR,"can't open pipe");
52       return PAM_AUTHINFO_UNAVAIL;
53    }
54    if (pid=fork()) {
55       close(p[0]);
56       write(p[1],passwd,strlen(passwd));
57       close(p[1]);
58       if (pid == waitpid(pid,&status,0) && WIFEXITED(status)) {
59          if (WEXITSTATUS(status)) {
60             _pam_log(LOG_ERR,"sslmount failed");
61             return PAM_AUTH_ERR;
62          }
63          _pam_log(LOG_ERR,"sslmount succeeded");
64          return PAM_SUCCESS;
65       }
66       _pam_log(LOG_ERR,"waitpid failed");
67       return PAM_AUTH_ERR;
68    }
69    close(0);
70    dup(p[0]);
71    close(p[0]);
72    close(p[1]);
73
74    execl(argv[0],argv[0],user,NULL);
75    _pam_log(LOG_ERR,"error: exec of first argument (sslmount) failed");
76    exit(1);
77 }
78
79 /* another expected hook */
80 PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv)
81 {
82    return PAM_SUCCESS;
83 }
84
85 #ifdef PAM_STATIC
86 struct pam_module _pam_listfile_modstruct = {
87    "pam_pwdfile",
88    pam_sm_authenticate,
89    pam_sm_setcred,
90    NULL,
91    NULL,
92    NULL,
93    NULL,
94 };
95 #endif
```

tifizieren. Sie kann die in Tabelle 4 genannten Rückgabewerte produzieren. Die Funktion »pam_sm_setcred()« (Zeile 80) wird von PAM immer erwartet. Sie übernimmt normalerweise Änderungen der Credentials (Berechtigungen) des Users. Die Funktion teilt der aufrufenden Applikation mit, welche Rechte dem User nun zustehen. Das Beispielmodul authentifiziert aber nicht selbst, sondern übernimmt nur vom vorhergehenden Modul die Authentifizierungsdaten. Damit stößt es eine Applikation an. Die spannenden Sachen passieren also in den Zeilen 35 bis 77, der Rest liefert den notwendigen Rahmen.

Die Aufgabe des Moduls ist es, neben der Pfadangabe für das Homeverzeichnis auch die Authentifizierungsdaten (Benutzername und Passwort) an das externe Programm zu übergeben. Angreifer sollen jedoch nicht die Möglichkeit haben, Passwörter bei der Übergabe abzugreifen. Deshalb gelangen sie durch eine Pipe (Zeilen 50 bis 57) unsichtbar an die mit »fork()« gestartete externe Applikation (Zeilen 54 und 74). Den Pfad zum externen Programm erhält »pam_sslmount.so« über Argumente wie in Listing 7 (Zeile 4) gezeigt.

## Daten vom vorherigen Modul übernehmen

Die Funktion »pam_sm_authenticate()« (Listing 8, Zeile 35) übernimmt vom vorherigen Modul Benutzername und Passwort. Sie liegen in der Variablen »pam_handle_t *pamh« und werden in Zeile 44 und 45 extrahiert. Sind Benutzername und Passwort leer, ging etwas beim Hintereinanderschalten der Module schief und die Funktion bricht mit einer Fehlermeldung ab (Zeile 47).

Die Funktion versucht in Zeile 50, eine Pipe zu öffnen. Danach zweigt sie einen Kindprozess ab (Zeile 54). Das Modul schließt sofort Pipe 0 (Eingabekanal), über den Ausgabekanal schreibt es das Passwort und schließt dann auch diesen Filedeskriptor. Zeilen 58 bis 66 prüfen, ob der Kindprozess erfolgreich war, und Zeile 67 beendet das Modul.

Der Code in Zeile 69 bis 76 kommt nur im Kindprozess zur Ausführung. Zeile 69 schließt den Stdin-Kanal und Zeile 70 trägt den Eingabekanal der Pipe als neuen Stdin ein. Dann schließt der Kindprozess beide Pipe-Endpunkte.

Jetzt ist die externe Applikation an der Reihe: Zeile 74 startet per »execl()« das Kommando. In den ersten beiden Argumenten ist die externe Applikation mit vollem Pfadnamen eingetragen, im dritten Argument die User-ID. Das Passwort wandert durch die Pipe vom Vater- zum Kindprozess und ist somit in »ps«-Ausgaben nicht zu sehen.

## Ein PAM für alles

PAM ist ein mächtiges Werkzeug zur Benutzerauthentifizierung. Es löst in Umgebungen mit vielen Maschinen und Nutzern die klassische Unix-Benutzerüberprüfung ab. Für viele Szenarien liefert es geeignete Module. PAM liest aus verschiedenen Frontends, von der klassischen Tastatureingabe bis zur modernen Chipkarte (siehe Artikel in diesem Heft), und bedient sich einer Reihe von Backends, etwa Passwd/Shadow oder LDAP. Ist das geeignete Modul nicht dabei, kann ein Admin mit wenig Aufwand PAM an seine Wünsche anpassen.

Pluggable Authentication Modules helfen, die Sicherheitsstruktur von Linux-Rechnern in einem Netzwerk zu verbessern. PAM ist dazu jedoch auf die Fähigkeiten der aufrufenden Applikation angewiesen. *(fjl)* ■

### Infos

[1] Überblick und Anleitungen zu PAM: [http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/]

[2] Modul PAM-SSL-Mount mit zugehörigen Skripten: [http://omnibus.uni-freiburg.de/~mawa/sslmount/]

### Die Autoren

**Dirk von Suchodoletz betreute einige Jahre lang das Wohnheimnetz des Studentenwerks Göttingen und baute an der dortigen Universität einen Linux-basierten Rechnerpool für die Studierenden auf. Derzeit ist er Assistent am Lehrstuhl für Kommunikationssysteme an der Fakultät für Informatik der Universität Freiburg.**

**Martin Walter ist seit 15 Jahren Mitarbeiter am Rechenzentrum der Universität Freiburg. Er betreute jahrelang Sun-Maschinen, bis er Mitte der 90er zu Linux wechselte und inzwischen über 200 festplattenlose Linux-Maschinen herrscht.**