Dennis Stender

Universität Hildesheim
Systemadministration II
Sommersemester 2001
Dozent: Ralf Hesse

PAM

Modulare Benutzerauthentifizierung unter Unix

Inhaltsverzeichnis

Dennis M. Stender
Samelsonplatz 5
31141 Hildesheim

I. Einführung

Historie

Die Anmeldung von Benutzern an ein Unix-System basierte lange Zeit auf einem einfachen Textfile, der Datei *passwd* im Verzeichnis */etc*. Das vom Benutzer eingegebene Passwort wurde dabei mit dem korrespondierenden Eintrag in der Datei verglichen und bei Übereinstimmung der Besitzer zur Anmeldung zugelassen.

In einer Zeit, in der die Bedeutung von Netzwerken immer größer wird und Rechner immer öfter global vernetzt sind, sind modernere Methoden gefragt. Die Sicherheitsbedürfnisse von Anwendern und Administratoren steigen und mit den Anforderungen der wachsenden Netzwerkumgebungen müssen auch die Authentifizierungsmechanismen skalieren. Neue Technologien ermöglichen es heute, einen zugangsberichtigten Benutzer sicherer und eindeutiger zu identifizieren, als es eine Passwortabfrage zu leisten vermag. Der übliche *login*-Prompt jedoch, der seit Ende der 70er Jahre am Anfang einer jeden Unix-(Konsolen)sitzung steht, kann diese Sicherheit nicht ohne Weiteres leisten.

Diese Herausforderung erkannte auch die Firma SunSoft und veröffentlichte schon 1995 im OSF-RFC 86.0 die sogenannten *Pluggable Authentication Modules* (PAM) für ihr Betriebssystem Solaris, einen flexiblen Mechanismus zur Abwicklung von

- An- und Abmeldung der Benutzer
- Überprüfung der Benutzerkonten
- Passwortverwaltung und ähnlichem.

Die genaueren Fähigkeiten und Funktionen von PAM seien in späteren Kapiteln *en détail* beschrieben.

Seitdem hat sich die PAM-Authentifizierung unter vielen Unix-Betriebssystemem durchgesetzt. Für Linux ist die freie Implementierung Linux-PAM verfügbar. Viele Distributionen wie RedHat und Caldera liefern fertige PAM-Pakete mit. Bei SuSE Linux 7.0 ist in der Standardinstallation die Authentifizierung mit PAM mit einer Reihe von weitergehenden Modulen vorkonfiguriert.

Benutzerauthentifizierung

Definitionen

Um die Bedeutung von PAM im Gesamtkontext darzustellen, ist es zunächst nötig, sich mit dem zugrunde liegenden Themenkomplex der Benutzerauthentifizierung zu befassen. Der Begriff der Authentifizierung bezeichnet dabei die Bestätigung der Authentizität eines Benutzers durch einen geeigneten Mechanismus, in der Regel durch eine Passwortabfrage. Durch diesen Vorgang weist sich der Benutzer als diejenige Person aus, die er durch Eingabe der User ID vorgibt zu sein. Im Falle der üblichen Authentifizierung mittels Passwort beglaubigt das System durch Vergleich des eingegebenen Passwort mit dem gespeicherten die Identität den Benutzers.

Um das System konsequent nach außen abzusichern, muß die Authentifizierung an sämtlichen sensitiven Zugangspunkten erfolgen. Als Systemzugangspunkt ist dabei nicht nur der *login*-Prozeß zu sehen, der den Zugang an der Konsole überwacht, sondern weitere Prozesse wie beispielsweise

- ftp
- telnet
- sshd
- xdm
- su

und andere.

An diesen Punkten muß jeweils eine geeignete *Authentifizierungstechnologie* implementiert sein. Neben der bereits beschriebenen Passwortabfrage sind je nach Sicherheitsbedürfnis und technischem Aufwand auch Mechanismen wie die Zugangssteuerung mittels Chipkarte denkbar. Auf der Karte wäre in diesem Fall eine Schlüssel gespeichert, den das System eindeutig einem bestimmten Benutzer zuordnen kann. Noch weitergehend sind neue Technologien auf Basis biometrischer Daten des Benutzers. So könnte der Benutzer durch seinen Fingerabdruck oder einen Scan seiner Netzhaut identifiziert werden.

Anforderungen

Die Motivation zur Entwicklung der PAM-Architektur lag vor allem in dem steigenden Anforderungen, denen die herkömmlichen Mechanismen nicht mehr gerecht werden konnten.

Als wichtigste Anforderung an die Benutzerauthentifizierung ist zweifelsohne die Sicherheit und Zuverlässigkeit des verwendeten Mechanismus zu sehen – ist doch die Absicherung des Rechnersystems ihre Hauptaufgabe. Dennoch ist ein Blick auf die Ansprüche an die Authentifizierung aus verschiedenen Perspektiven sinnvoll.

So ist für den Systemverwalter auch von entscheidender Bedeutung, welchen administrativen Aufwand die Authentifizierung mit sich bringt. Eine zentralisierte Benutzer- und Passwortdatenbank für möglichst viele genutzte Dienste macht dem Administrator die Arbeit nicht nur bequemer, sondern kann auch Sicherheitsrisiken wie inkonsistente Benutzerdatenbestände verhindern. Als Negativbeispiel sei ein Netzwerk genannt, in dem die Benutzerliste für jeden Rechner und jeden Systemzugangspunkt (*login, ftp*....) dieser Rechner einzeln gewartet werden müsste. Durch den hohen administrativen Aufwand könnte es leicht passieren, dass ein längst aus der Organisation entferntes Benutzerkonto an manchen Systemen oder Zugangspunkten noch Zutritt erhält. Besonders in Zeiten heterogener Netzwerkstruktur wird dieses Problem umso deutlicher.

Im Zuge dieser Entwicklung können sich in vielen Fällen die herkömmlichen Authentifizierungstechnologien einer Benutzer/Passwort-Abfrage als nicht mehr ausreichend erweisen. Daher sollten sich seitens des Administrators verschiedene Zugangstechnologien implementieren und flexibel an ihren Systemzugangspunkt anpassen lassen. Wenn die technischen Vorraussetzungen für die Anmeldung an Arbeitsstationen per Chipkarte gegeben sind, wird sich diese Möglichkeit aber beispielsweise nicht für einen FTP-Zugang eignen. Dem Verwalter müssen also Mittel und Werkzeuge an die Hand gegeben werden, je nach Zugangspunkt unterschiedliche Mechanismen auf demselben System implementieren zu können.

Die Prioritäten seitens des Systemanwenders können sich im Vergleich mit seinem Administrator leicht verschieben. Liegt der Fokus beim Systemverwalter eher auf Sicherheitsaspekten, rückt für den Anwender, der sich mit der Benutzerauthentifizierung letztenendes auseinandersetzen muß, auch die Bequemlichkeit

dieses Vorgang ins Blickfeld. Bei allen nötigen Sicherheitsanforderungen muß der Benutzer der Authentifizierung auch die nötige Akzeptanz entgegenbringen können. Wird sie für ihn zum lästigen Übel, wird er Wege suchen, sich den Vorgang soweit wie möglich zu vereinfachen. Das kann wiederum zu Sicherheitslücken wie einfachen und gar leeren Passwörtern führen, wenn die Sicherheitsrichtlinien dies denn zulassen.

Der Anmeldevorgang sollte sich demnach aus Benutzersicht möglichst einfach gestalten. Durch einheitliche Anmeldedialoge für unterschiedliche Dienste kann ihm die Anmeldung vereinfacht werden. Wenn der Benutzer für möglichst viele Dienste dieselbe Benutzer/Passwort-Kombination verwenden kann, wird er das in der Regel begrüßen. Im Idealfall wird er sich durch eine einzige Passwortabfrage für mehrere Dienste authentifizieren. So wird ein Unix-Benutzer in der Regel nach der Anmeldung am *login*-Prompt auch Zugriff auf andere Resourcen wie sein Mailverzeichnis oder NFS-Server erhalten. Natürlich gilt dabei andererseits auch, dass umso mehr Dienste gleichzeitig kompromittiert werden können, je weniger unterschiedliche Passwörter und Passwortabfragen verwendet werden.

Neben den Erfordernissen der Systembenutzer und –verwalter spielen auch die Interessen von Entwicklern eine Rolle: Programmierern, die in ihrer Software auf Authentifizierung zurückgreifen wollen, ist durchaus daran gelegen, diese nicht selber implementieren zu müssen. Auf der anderen Seiten haben Hardware-entwickler von beispielsweise Chipkartensystemen ein Interesse daran, ihre Technologie mit geringem Aufwand einer Vielzahl von Diensten zur Verfügung zu stellen.

Herkömmliches Authentifizierungsmodell

Die traditionelle Form der Authentifizierung ohne PAM-Module, die noch immer bei vielen Unix-Systemen Verwendung findet, basiert auf Methoden der Standard-bibliothek *libc*.

Beispiel login

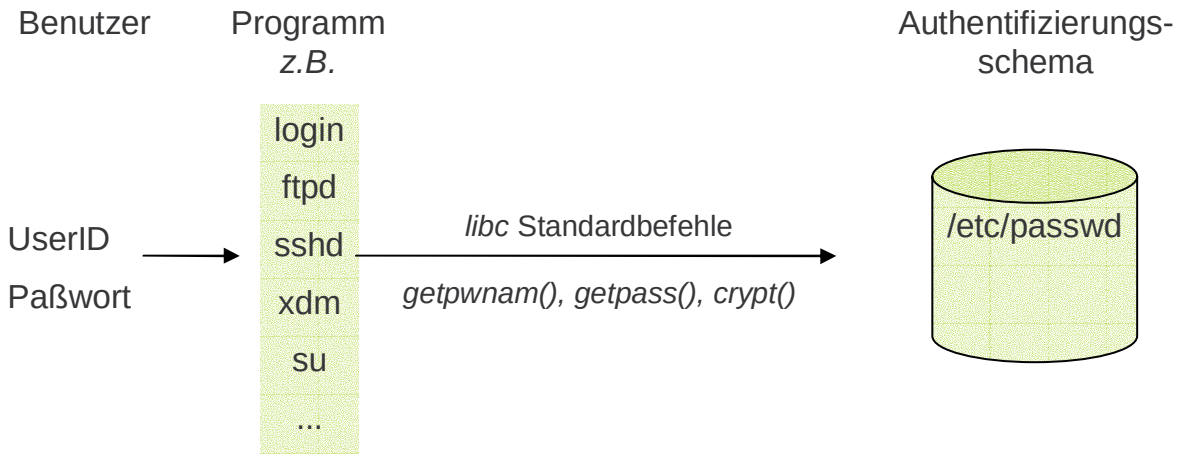


Abbildung 1: Herkömmliche Authentifizierung am Beispiel login

Wenn sich ein Benutzer mit seinem Username beispielsweise am *login*-Programm anmeldet, holt die Funktion *(libc).getpwnam()* den zum Usernamen gehörenden Eintrag aus der Datei */etc/passwd*. *getpass()* erfragt daraufhin das geheime Passwort. Ist dieses nicht in */etc/passwd* ablegt, wird es mittel *getspnam()* Methode aus der */etc/shadow* ermittelt. Die Zugriffsmethoden sind dabei fest in das Login-Programm einkompiliert.

Die *crypt*-Routine aus *libc* verschlüsselt bei der Benutzereinrichtung das Kennwort, um es nicht im Klartext in der */etc/passwd* oder */etc/shadow* zu speichern. Gibt der Benutzer beim Login sein Passwort ein, wird die Eingabe auf die gleiche Weise verschlüsselt. Stimmt das sich daraus ergebene Schlüsselwort mit dem aus */etc/shadow* überein, hat sich der Benutzer erfolgreich authentifiziert.

Beispiel Chipkarte

Soll die Anmeldung nicht durch Passwortabfrage, sondern eine Chipkarte abgewickelt werden, so muß *login* anstelle der *libc*-Methoden spezifische Methoden

zur Authentifizierung mittels Chipkarte implementieren. login fällt damit auch die Hardwaresteuerung des Chipkartenlesers zu.

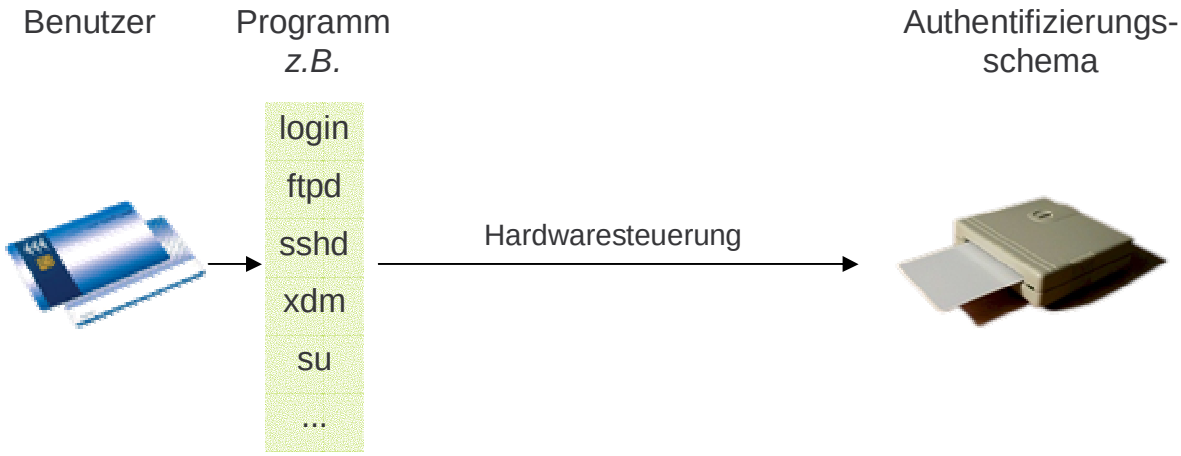


Abbildung 2: Herkömmliche Architektur bei Nutzung von Chipkarten

Nachteile

An diesem Beispiel wird ein großer Nachteil dieser traditionellen Architektur deutlich: Die Authentifizierungsmethoden sind fest in das Programm kompliliert, das die Zugangskontrolle durchführen soll. Bei Umstieg auf ein anderes Authentifizierungs-modell – sei es Chipkarte oder Kerberos-Paßwortserver – muss auch jedes der betroffenen Zugangsprogramme wie *login*, *su* oder *xdm* gegen eine spezifische Version ausgetauscht werden, die auf die verwendete Technologie zugeschnitten ist.

Ein möglicher Ausweg aus der geschilderten Problematik ist daher die Nutzung von austauschbaren Authentifizierungsmodulen, auf die die Zugangsprogramme zugreifen: Pluggable Authentication Modules.

II. PAM im Detail

Arbeitsweise

Statt auf die Benutzerdatenbank oder sonstige Authentifizierungsschemata greifen die Login-Programme in der PAM-Architektur nur auf PAM zu. Ein entsprechendes PAM-Modul übernimmt die Authentifizierung. PAM fungiert damit als Mittler zwischen Programmen und dem jeweils genutzten Authentifizierungssystem. Bei einem Wechsel der Authentifizierungstechnologie muß lediglich das PAM-Modul ausgetauscht werden. Das Zugangsprogramm bleibt gegen PAM gelinkt.

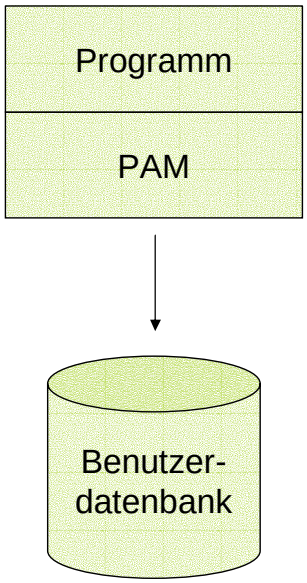


Abbildung 3: PAM-Schichten

Aufgaben

Bisher haben wir als Authentifizierung vor allem den Anmeldeprozeß betrachtet. Zu dem größeren Kontext der Authentifizierung zählen aber auch aber auch differenziertere, weitergehende Aufgaben als lediglich die Authentizität eines Benutzers zu überprüfen. Die Aufgaben von PAM-Modulen lassen sich daher in vier Kategorien gliedern, bezeichnet mit den englischen Bezeichnungen:

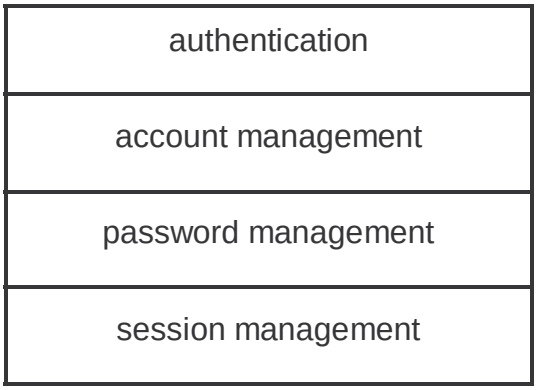


Abbildung 4: Aufgabenkomplexe von PAM

Der Bereich *authentication* beschreibt die bereits betrachtete Aufgabe, die Identität einer Person zu überprüfen. Zudem können im Rahmen der Authentifizierung auch so genannte *credentials*, also Ausweisdokumente, erzeugt werden, mit denen das System die Authentizität des Benutzers „beglaubigt“.

In den Bereich *account management* fallen Aufgaben, die mit dem Konto eines bereits authentifizierten Benutzers zusammenhängen. So könnte beispielsweise nach der Passwortabfrage eine Überprüfung erfolgen, ob das soeben korrekt eingebenene Passwort des Benutzer abgelaufen ist und erneuert werden muß. Eine andere Aufgabe aus dem Bereich des *account management* sind zeitbezogene Kontenbeschränkungen. Wenn sich ein bestimmter Benutzer nur zu bestimmten Zeiten am System anmelden darf, dann muß sich die Person zunächst korrekt als dieser Benutzer authentifizieren, bevor die Anmeldebeschränkung durch das Account-Management festgestellt werden kann.

Zum *password management* gehören sämtliche Aufgaben, die das Ändern eines Passwortes betreffen, wenn es als abgelaufen erkannt worden ist.

Der Bereich session management schließlich fasst sitzungsabhängige Aufgaben zusammen. Das könnte die Protokollierung von Login-Versuchen im Syslog sein oder Funktionen, die durch Sitzungsanfang und –ende die Benutzerzeit abrechnen.

Architektur

Die folgende Grafik illustriert die Komponenten der PAM-Architektur:

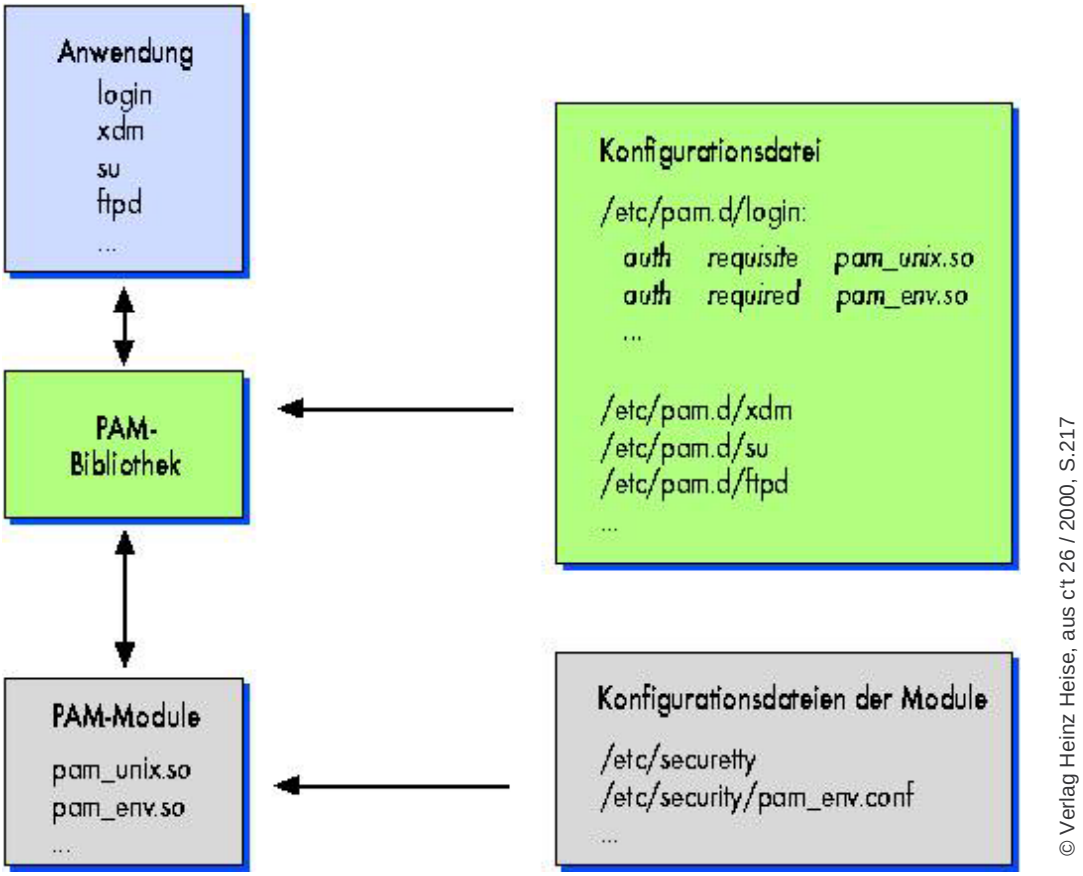


Abbildung 5: PAM Architektur am Beispiel Linux-PAM

PAM-Bibliothek

Mittelpunkt der PAM-Architektur ist die PAM-Bilbliothek, gegen die Anwendungen wie *login* fest gelinkt werden. Wenn die Anwendung einen Authentifizierungsvorgang durchführt, ruft sie die entsprechende Methode der PAM-Bibliothek auf. Durch die zentrale Konfigurationsdatei werden der aufrufenden Anwendung auszuführende PAM-Module zugeordnet. Die Bibliothek ruft diese Module in der angegebenen Reihenfolge auf und gibt eine Erfolgs- oder Misserfolgsmeldung an den Prozeß zurück.

Zur Kommunikation mit Anwendung und Modulen nutzt die PAM-Bibliothek zwei Programmierschnittstellen: Mit den Anwendungen kommuniziert sie durch das *Application Programming Interface* (API), für die Module bietet sie ein *Service Provider Interface* (SPI).

In der API-Schnittstelle sind je nach Aufgabenbereich im Wesentlichen die folgenden Funktionen definiert:

pam_start() – der PAM-Mechanismus wird gestartet

Authentifizierung

pam_authenticate() – wird aufgerufen, wenn der Authentifizierungs-vorgang gestartet werden soll

pam_setcred() - erzeugt die credentials

Account-Management

pam_acct_mgmt() – wird nach der Authentifizierung zur Feststellung von Kontenbeschränkungen etc. aufgerufen

Session-Management

pam_open_session() – aufgerufen zu Beginn einer Benutzersitzung

pam_close_session() – aufgerufen bei Ende einer Benutzersitzung

Paßwort-Management

pam_chauthok() – wenn das Passwort geändert werden muß

pam_end() – Die Schnittstelle zur Bibliothek wird geschlossen.

Hinzu kommen noch einige Funktionen zum Austausch von Daten zwischen der Applikation und der Bibliothek. Mit dem Rückgabewert der Funktionen lässt sich der Ablauf des aufrufenden Programms steuern. Wenn die jeweilige Aufgabe erfolgreich abgeschlossen werden konnte, liefert die Bibliothek das Ergebnis *PAM_SUCCESS*.

Das SPI, die Schnittstelle zwischen Bibliothek und den Modulen, bildet alle schon für das API definierten Routinen ab. Ihre Aufgaben und Aufrufsyntax entsprechen weitestgehend den Pendants der API. Zusätzlich erhalten sie jedoch die in der Konfigurationsdatei zugeordneten Argumente.

PAM-Module

Es existiert eine Vielzahl von vorgefertigten PAM-Modulen; für die Linux-Implementierung ist unter http://www.kernel.org/pub/linux/libs/pam/modules.html eine

Auflistung verfügbar. Anhand des „Module Writers' Guide“ können Entwickler auch eigene Module individuell erstellen.

Die einer Installation zur Verfügung stehenden PAM-Module befinden sich üblicherweise im Verzeichnis */lib/security/*. Ein Modul kann zusätzlich über spezifische Konfigurationsdateien verfügen, beispielsweise nutzt *pam_securetty.so* die Datei */etc/securetty*.

Entsprechend den beschriebenen Aufgaben im Prozeß der Zugangskontrolle werden die PAM-Module in verschiedene Typen unterteilt,

- *auth*-
- *account*-
- *password*- und
- *session*-Module.

Dabei kann ein Modul auch Aufgaben in mehreren Bereichen übernehmen. Das häufig genutzte Modul *pam_unix* beispielsweise stellt Funktionen für sämtliche beschriebenen Aufgabenbereiche zur Verfügung.

Ausgewählte PAM-Module bei SuSE Linux 7.0

auth
pam_unix.so - Standardauthentifizierung über /etc/passwd und /etc/shadow
pam_securetty.so - Erlaubt den root-Zugang nur, wenn er von einem in /etc/securetty eingetragenen Terminal aus erfolgt
pam_nologin.so - verweigert allen Usern außer root den Zutritt, wenn die Datei /etc/nologin existiert
pam_mail.so - prüft, ob der Benutzer ungelesene Nachrichten in seinem Postfach hat

account
pam_time.so - prüft, ob der Zugriff für den Benutzer zur aktuellen Zeit erlaubt ist, festgelegt über /etc/security/time.conf

password
pam_pwcheck.so - zuständig für Passwortänderungen
pam_cracklib.so - achtet bei Passwortänderungen darauf, dass sichere Passwörter gewählt werden

session
pam_limits.so - setzt Beschränkungen auf Ressourcen

Konfigurationsdatei

Die Zuordnung, welche Module von welchem Systemzugangsprozeß genutzt werden sollen, erfolgt durch zentrale Konfigurationsdateien im Textformat. Unter Solaris existiert eine einzige Konfigurationsdatei */etc/pam.conf* für alle Dienste. In der Linux-Implementierung gibt es eine Konfigurationsdatei je Dienst im Verzeichnis */etc/pam.d*, jeweils bezeichnet mit dem Namen des aufrufenden Dienstes. Die dem *login*-Prozeß zugeordneten Module sind unter Linux also in */etc/pam.d/login* aufgeführt.

Die Konfigurationsdatei enthält ein konfiguriertes Modul je Zeile und wird zeilenweise abgearbeitet. Jede Zeile ist in fünf (Solaris) respektive vier (Linux) durch Leerzeichen oder Tabulator getrennte Spalten unterteilt:

```
[Dienst]   Modultyp   Kontrollflag   Modulpfad   Optionen
```

Abbildung 6: Spalteneinteilung der Konfigurationsdatei

- Dienst - (nur bei Solaris), entfällt bei Linux, da durch Dateiname gegeben
- Modultyp – Unterscheidet das Modul nach Aufgabe, entweder *auth*, *account*, *password* oder *session*
- Kontrollflag – definiert, was passiert, wenn ein Modul fehlschlägt (z.B. keine Authentifizierung mittels Passwort gelingt). Mögliche Optionen:
    - requisite – Das Modul muß erfolgreich durchlaufen, sonst Abbruch. Bei Abbruch werden alle übrigen Module übersprungen.
    - required – Das Modul muß erfolgreich durchlaufen, sonst Abbruch. Vor Abbruch werden übrige required Module ausgeführt. Bei Erfolg wird das nächste Modul ausgeführt.
    - sufficient - Bei Erfolg werden sofort Rückmeldung, übrige Module werden übersprungen. Bei Mißerfolg Ausführung des nächsten Moduls.
    - optional - Ausführung des nächsten Moduls unabhängig von Erfolg
- Modulpfad - z.B. /lib/security/pam_unix.so
- Optionen – modulspezifische Parameter

PAM am Beispiel des login-Prozesses

```
#%PAM-1.0
#/etc/pam.d/login
#
auth     requisite        /lib/security/pam_unix.so nullok
auth     required         /lib/security/pam_securetty.so
auth     required         /lib/security/pam_nologin.so
auth     required         /lib/security/pam_env.so
auth     required         /lib/security/pam_mail.so
account  required         /lib/security/pam_unix.so
password required         /lib/security/pam_pwcheck.so      nullok
password required         /lib/security/pam_unix.so         nullok
                                        use_first_pass use_authtok
session  required         /lib/security/pam_unix.so         none
session  required         /lib/security/pam_limits.so
```

Abbildung 7: PAM-Konfigurationsdatei für den Linux login-Prozeß

Obige Abbildung zeigt die voreingestellte Konfigurationsdatei des *login*-Prozesses einer SuSE Linux Distribution in der Version 7.0. Bei *login* handelt es sich um einen Standarddienst zur Benutzeranmeldung, der auf jedem Linux- oder Unix-System läuft. Die für diesen Dienst konfigurierten PAM-Module kommen zum Einsatz, wenn man sich lokal an der Konsole oder via telnet anmeldet.

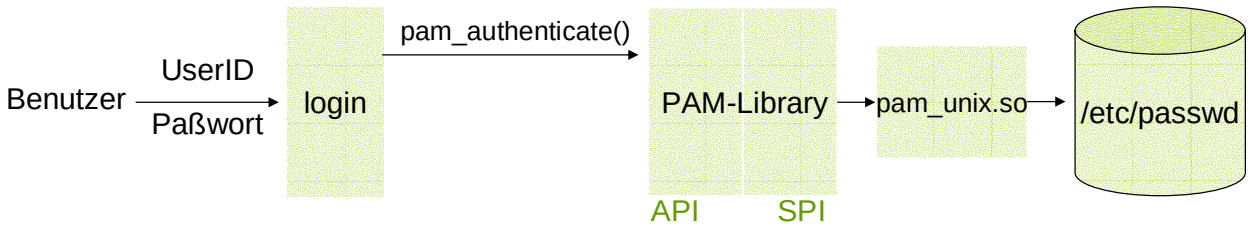


Abbildung 8: Die erste Zeile der Konfigurationsdatei wird ausgeführt.

pam_unix

Das *login*-Programm ruft als erstes die Methode *pam_authenticate()* der PAM-Library über die API auf. Daraufhin aktiviert die Library die konfigurierten Module des Typs *auth*. Beim ersten Modul in der abgebildeten Konfigurationsdatei handelt es sich um das Modul *pam_unix*. Es erzeugt die Abfrage von UserID und Passwort und verifiziert beides anhand der Dateien */etc/passwd* und */etc/shadow*. Die Option *nullok* ermöglicht dabei das Verwenden von „leeren“ Passwörtern.

Da *pam_unix* in der Konfigurationsdatei mit dem Flag *requisite* versehen ist, würde bei einer falschen Eingabe des Passwortes der gesamte Vorgang abgebrochen, ohne die weiteren Module auszuführen. Wenn sich der Benutzer nicht korrekt „ausweisen“ kann, macht es keinen Sinn, noch zu überprüfen, ob er beispielsweise neue Post hat.

Hat der Benutzer ID und Passwort korrekt eingegeben, folgt die Ausführung des nächsten Moduls in der Konfigurationsdatei. Da die nächsten Module als *required* konfiguriert sind, werden beim einem Fehlschlag eines dieser Module noch die übrigen *required*-Module desselben Typs ausgeführt, bevor *login* eine Fehlermeldung erhält.

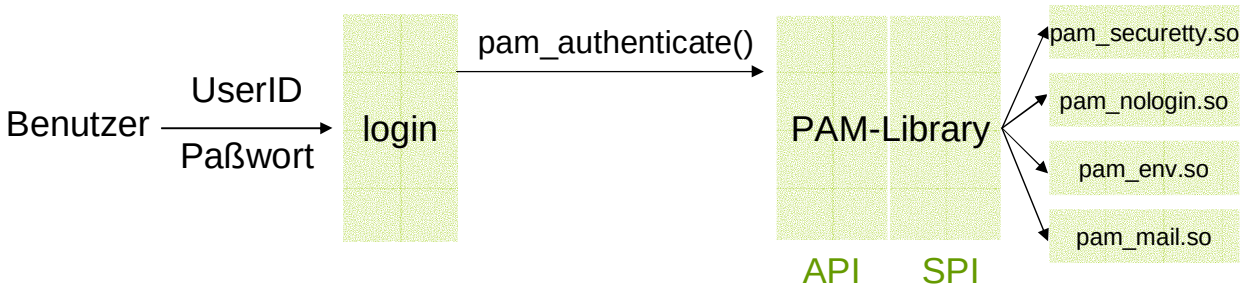


Abbildung 9: Ausführung übriger auth-Module

pam_securetty

Dieses Modul verhindert die Anmeldung als root über unsichere Terminals. Ein sicheres Terminal wird in der Datei */etc/securetty* definiert. So lässt sich der root-Zugang über *telnet* verhindern, da dieses Protokoll Benutzernamen und Passwort unverschlüsselt über das Netz überträgt und beides mittels eines Paket-Sniffers mitgehört werden könnte.

pam_nologin

Wenn die Datei /etc/nologin vorhanden ist, lässt dieses Modul den Zugriff für niemanden außer root zu. Diese Option ist vorrangig für Server interessant, an denen sich keine Benutzer lokal anmelden sollen.

pam_env

Wenn pam_nogin dem Benutzer die Anmeldung erlaubt hat, setzt dieses Modul zugehörige Umgebungsvariablen, die über die Datei */etc/security/pam_env.conf* konfiguriert werden können. *pam_env* ist stets erfolgreich.

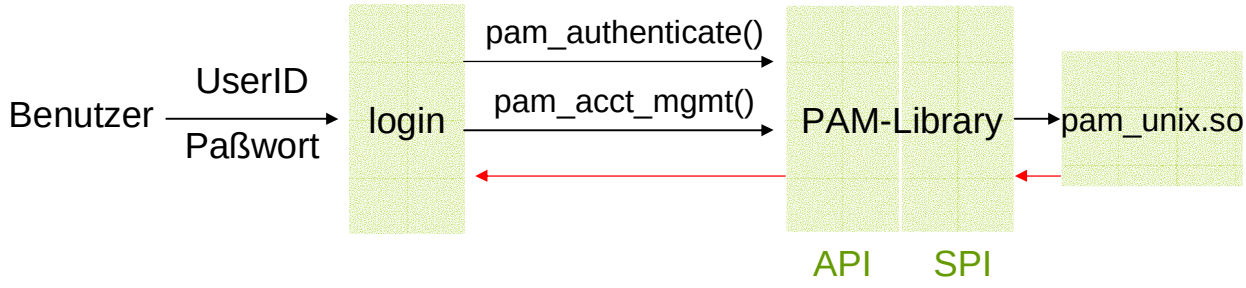
pam_mail

Gibt die die Meldung „You have new mail“ aus, wenn der Benutzer neue Nachrichten in seinem Mail-Verzeichnis hat.

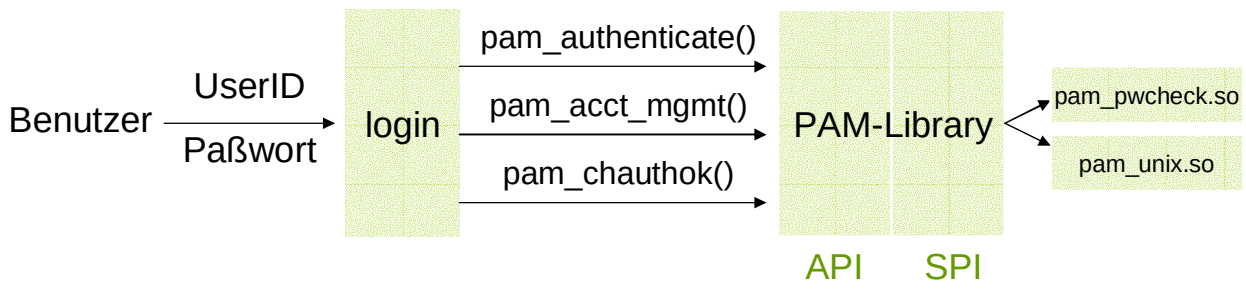
Da die obigen *auth*-Module als *required* definiert worden sind, erfährt der login-Prozeß erst jetzt, ob sie erfolgreich ausgeführt worden sind. Ist dies der Fall, erhält *login* den Rückgabewert *pam_success* und ruft durch die Funktion *pam_acct_mgmt()* die konfigurierten Account-Module auf.

pam_unix als Account Modul

Das einzige in der Konfigurationsdatei definierte *account*-Modul ist wiederum *pam_unix*, das bereits im *auth*-Kontext verwendet worden ist. Wird es als *account*-Modul aufgerufen, überprüft es, ob das Passwort abgelaufen ist. Wenn ja, gibt es eine negative Rückmeldung an *login*.



Auf diese Rückmeldung kann nun login mit dem Aufruf der Methode *pam_chauthok()* reagieren und damit eine Passwortänderung initiieren.



Passwort-Änderungen

Bei Aufruf von *pam_chauthok()* startet die PAM-Library die als Typ *password* konfigurierten Module.

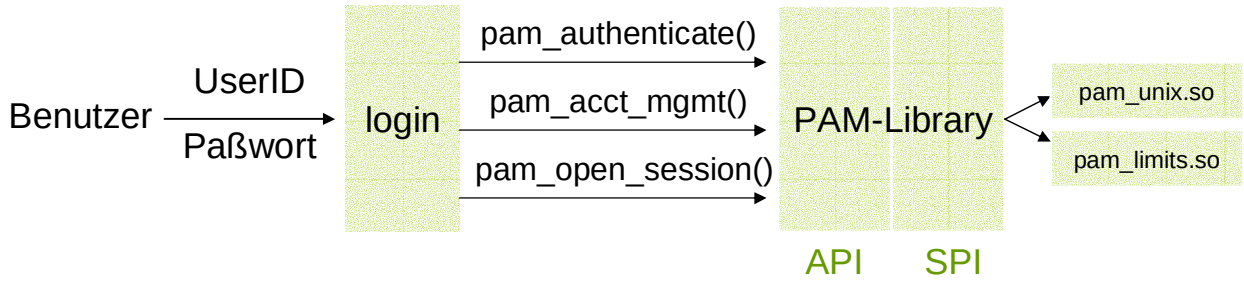
pam_pwcheck stellt anhand der *CrackLib*-Bibliothek sicher, dass der Benutzer kein zu leichtes Passwort wählt. So lässt sich zum Beispiel eine bestimmte Mindestlänge durchsetzen. Das Modul gibt in einem solchen Fall allerdings nur eine Warnung aus. Die Option *nullok* veranlasst es sogar, auch leere Passwörter zuzulassen.

Als zweites password-Modul ist wiederum *pam_unix* aktiviert, das die eigentliche Passwortänderung durchführt. Die Parameter *use_first_pass* und *use_authok* bewirken, dass der Benutzer sein altes Passwort bei der Ändernung nicht noch einmal eingeben muß, wie es sonst beim *passwd*-Befehl der Fall wäre.

Ist das Passwort abgelaufen gewesen, so erhält der *login*-Prozeß nach erfolgter Änderung den Rückgabewert *pam_success* auf die *pam_chauthok()*-Methode. Wenn das Passwort noch gültig gewesen ist, so wird vom *account*-Modul *pam_success* bereits beim Aufruf von *pam_acct_mgmt()* zurückgegeben.

Session-Module

Das *login*-Programm teilt nun der Library durch Aufruf von *pam_open_session()* mit, dass die Benutzersitzung geöffnet wird, und die Library aktiviert die *session*-Module.



Der Aufruf von *pam_unix* an dieser Stelle verzeichnet die Benutzeranmeldung im *syslog*-Protokoll (in der Regel */var/log/messages*). *pam_limits* setzt benutzer-abhängige Beschränkungen, die es aus der Datei */etc/limits* liest. So läßt sich für jeden Benutzer außer *root* die maximale Anzahl von geöffneten Dateien, gleichzeitig laufenden Systemprozessen und ähnlichen Parametern beschränken.

Beim Ausloggen ruft *login* die Methode *pam_close_session()* auf, so dass sich an dieser Stelle beispielsweise die Nutzungsdauer erfassen lässt.

III. Praxisbeispiel: SMB-Authentifizierung mit PAM

Das vorangegangene Beispiel der üblichen login-Anmeldung zeigt bereits, wie flexibel konfigurierbar der Anmeldevorgang durch PAM wird. Durch die einzelnen Module lässt sich nahezu jeder Aspekt der Anmeldung am *login*-Prompt an die eigenen Anforderungen anpassen.

Doch erschiene der Aufwand der PAM-Architektur für eine einfache login-Prozedur oft nicht gerechtfertigt, ließen sich mit PAM nicht auch komplexere Aufgabenstellungen einfach bewältigen. Ein weitergehendes Anwendungsbeispiel für PAM sei im folgenden dargestellt.

Problemszenario

In den meisten professionellen Netzwerken ist nicht nur ein einziges Betriebssystem vertreten, sondern es herrscht eine heterogene Netzwerkstruktur unterschiedlicher Systeme aus der Unix- und der Windows-Welt. In einem solchen Netz aus Windows NT-, Solaris- und Linux-Rechnern sehen sich Administratoren und Benutzer mit dem Problem mehrere parallel betriebener Benutzerdatenbanken konfrontiert.

Auf NT-Seite lässt sich zwar durch einen sogenannten Domänencontroller die Benutzerverwaltung zentralisieren, wie es auch auf den Unix-Plattformen durch einen NIS-Server möglich ist. So müssen die Benutzer zumindest nicht an jeder Arbeitsstation gepflegt werden, aber „Pendler“ zwischen beiden Plattformen müssen in beiden Benutzerdatenbanken getrennt eingerichtet und administriert werden. Dies bringt dem Benutzer nicht nur die Unbequemlichkeit, sich gegebenenfalls unterschiedliche Passwörter auf beiden Plattformen merken zu müssen, sondern birgt auch Sicherheitsrisiken, wenn die Administratoren Änderungen nicht in beide Datenbanken gleichzeitig einpflegen.

Es wäre folglich wünschenswert, eine zentrale Benutzerdatenbank gemeinsam für beide Plattformen einrichten zu können. Windows kann jedoch keine NIS-Datenbank nutzen und verfügt auch nicht über eine flexible Schnittstelle wie PAM, um sich diese Fähigkeit anzueignen. Die Linux- und Solaris-Workstations müssen also ein PAM-Modul benutzen, mit sie ihre Benutzer mittels SMB-Protokoll (*Server Message Blocks*) gegen einen Windows NT-Domänencontroller authentifizieren können.

Windows-Authentifizierung mit pam_smb_auth

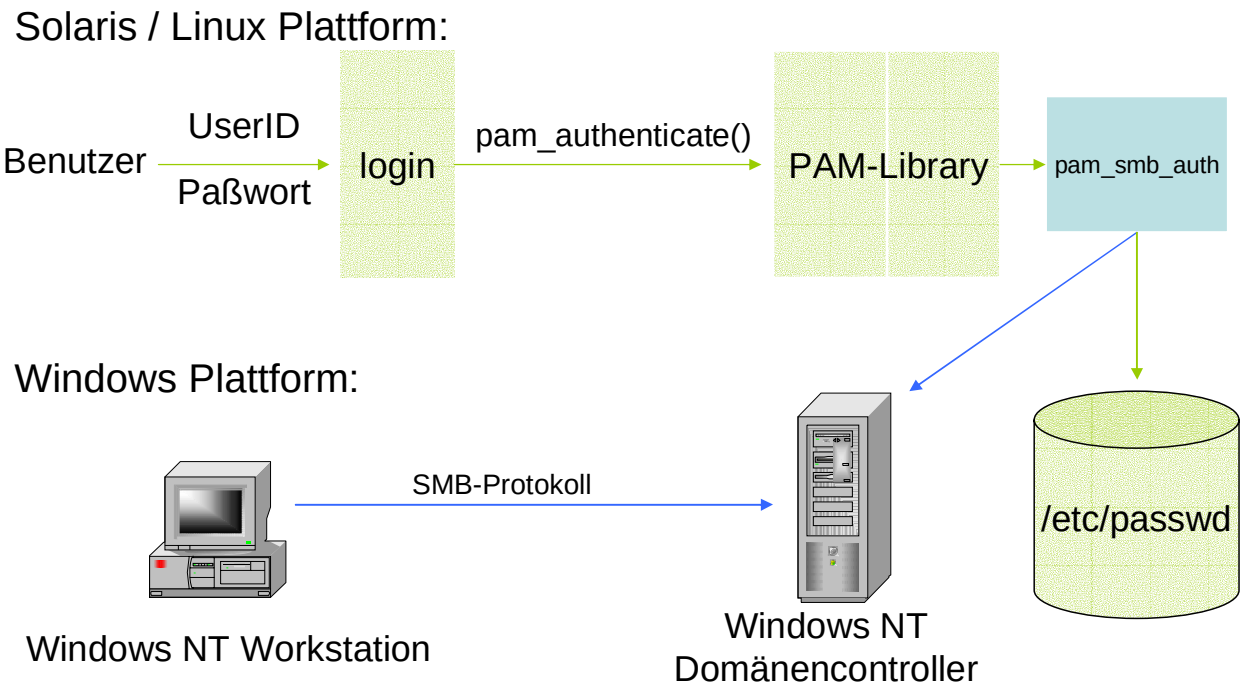
Genau diese Anforderung erfüllt das PAM-Modul *pam_smb_auth* von David Arlie. Es ist für Solaris wie für Linux verfügbar und ist bei SuSE Linux schon im Lieferumfang enthalten.

Durch das modulare Konzept von PAM ist die Einbindung unkompliziert: Es wird im *auth*-Stack gegen das Modul pam_unix ausgetauscht, so dass sich */etc/pam.d/login* wie folgt verändert:

```
auth        requisite               /lib/security/pam_smb_auth.so       #nolocal
#auth        requisite               /lib/security/pam_unix.so           nullok
auth        required                /lib/security/pam_securetty.so
auth        required                /lib/security/pam_nologin.so
auth        required                /lib/security/pam_env.so
auth        required                /lib/security/pam_mail.so
account     required                /lib/security/pam_unix.so
password    required                /lib/security/pam_pwcheck.so        nullok
password    required                /lib/security/pam_unix.so           nullok
                                                          use_first_pass use_authtok
session     required                /lib/security/pam_unix.so                       none
session     required                /lib/security/pam_limits.so
```

Die erste Zeile ersetzt hier die auskommentierte zweite. Die übrigen Einbindungen von *pam_unix* als *account*-, *password*- und *session*-Modul bleiben jedoch erhalten.

In der Standardkonfiguration authentifiziert es den Benutzer nicht nur gegen den Windows Domänencontroller, sondern akzeptiert auch weiterhin wie das Modul *pam_unix* die Authentifizierungsdaten aus */etc/passwd* und */etc/shadow*:



Um die Anbindung an die Windows Domäne zu konfigurieren, verfügt das Modul über eine dreizeilige Konfigurationsdatei */etc/pam_smb.conf*.

```
DOMAENENNAME
PDC
BDC
```

Die erste Zeile der Datei enthält den Namen der Windows NT-Domäne, die zweite den Primären Domänencontroller dieser Domäne und die dritte den Backup Domänencontroller.

Nachteile

In der abdruckten /etc/pam.d/login wird bereits deutlich, dass die SMB-Authentifizierung nicht völlig autark von der Unix-Authentifizierungsarchitektur arbeitet. Die in der NT-Domäne eingerichteten Benutzer müssen unter selbem Namen auch auf dem Unix-System vorhanden sein, um die Dateiberechtigungen korrekt zuordnen zu können. Bei Nutzung des SMB-Moduls werden zusätzlich zu den Passwörtern der lokalen Benutzer auch die ihrer Pendants aus der Windows-Domäne akzeptiert. Da das Modul aber nicht über einen Mechanismus zur Synchronisierung der Benutzernamen beider Plattformen verfügt, wird die Benutzerverwaltung nicht ausschließlich in die Windows-Domäne verlagert.

Zudem akzeptiert das Modul in seiner Standardkonfiguration beide Passwörter gleichzeitig, sowohl das Windows-Kennwort, wie auch das lokale Unix-Passwort aus */etc/passwd* bzw. */etc/shadow*. Da *pam_smb_auth* nicht als *password* Typ eingebunden ist, kann der Benutzer vom Unix-System aus nur das lokale Passwort ändern. Die Änderung des Windows-Kennwort kann in dieser Konstellation nur von einem Windows-Rechner aus durchgeführt werden. Ändert der Benutzer daher sein Kennwort unter Unix mittels des *passwd*-Befehls, führt dies zur Inkonsistenz beider Passwörter. In diesem Fall gibt es für denselben Dienst zwei unterschiedliche, gültige Passwörter: das Unix- und das Windows-Kennwort.

Zwar verfügt pam_smb_auth über den Parameter *nolocal* (in der abgedruckten Konfiguration auskommentiert), so es dass die lokale (Unix-)Benutzerdatenbank nicht mehr berücksichtigt. Aber die Benutzung dieser Option kann nicht empfohlen werden, weil dann auch der Benutzer *root* über die Windows-Domäne authentifiziert werden müsste. Und sollte zum Beispiel durch eine fehlerhafte Netzwerkverbindung

der Zugriff auf die Domäne nicht möglich sein, kann sich *root* auch nicht mehr am Unix-System anmelden.

IV. Fazit

Zwar mag der Aufwand von PAM für eine einfache login-Konfiguration übertrieben wirken, doch macht die Modularisierung auch in diesem einfachen Anwendungsfall den gesamten Authentifizierungsprozeß transparenter. Durch die Gliederung in verschiedene Module können auch weniger sicherheitsrelevante Optionen wie die Überprüfung auf neue Mails flexibel an die eigenen Anforderungen und Wünsche angepasst werden.

Erst PAM ermöglicht jedoch die einfache Einbindung weitergehender Authentifizierungstechnologien, ohne sämtliche Systemzugangsprozesse austauschen zu müssen. Diese Anpassung an die eigenen Sicherheitsanforderungen kann je nach Organisation einen deutlichen Gewinn an Betriebssicherheit bedeuten.

Das angeführte Beispiel in einem gemischten Windows / Unix-Netzwerk zeigt zudem, wie die PAM-Authentifizierung mit größeren Netzwerken skalieren kann. Dieses Ziel muß in der Praxis nicht notwendigerweise mit einem Windows NT-Server umgesetzt werden. PAM-Module für Kerberos, LDAP-Server oder andere Umgebungen wie Netware können auch weitergehende Ansprüche in einer komplexen Netzwerkstruktur befriedigen.

Literatur

Hetze, Sebastian (1997), Sesam öffne Dich, iX Magazin für professionelle Informationstechnik, Nr. 9, S. 118.

Heuse, Marc(2000), Sesam öffne Dich! c't Magazin für Computer Technik, Nr. 26, S. 216.

Morgan, Andew G. (2001), The Linux-PAM System Administrators' Guide, http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html

Morgan, Andew G. (2001), The Linux-PAM Module Writers' Guide, http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_modules.html

Lai, Charlie (1996), Making Login Services Independent of Authentication Technolgies, http://www.sun.com/software/solaris/pam/pam_ste_talk.pdf