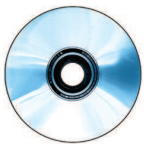
Alkalmazásbiztonság a PAM modul segítségével

A PAM (Pluggable Authentication Module) alapjai,
PAM támogatású alkalmazások fejlesztése és PAM-beállításfájl létrehozása.



A hitelesítés olyan folyamat, ami azt ellenőrzi, hogy az adott entitás valóban az-e, akinek állítja magát. Linux-rendszereken a felhasználók hitelesítésére a `su`, `passwd` vagy `login` alkalmazásokat használjuk, még mielőtt a rendszer erőforrásaihoz engednénk őket. A felhasználói adatok szinte minden Linux-terjesztésben a */etc/passwd* fájlban tárolódnak. Ez egy olyan szöveges állomány, ami a felhasználó `login` nevét, titkosított jelszavát, az egyedi, numerikus felhasználói azonosítót (amit `uid`-nek neveznek), egy numerikus csoportazonosítót (amit `gid`-nek nevezünk), tetszés szerint megadható megjegyzésmezőt (ahol általában a felhasználó valódi nevét, telefonszámát és hasonló dolgokat tárolhatunk), a saját könyvtárát és a kedvenc héjprogramját tartalmazza.
A */etc/passwd* egy sora valahogy így néz ki:

```
aztec:K52xi345vMO:900:900:Aztecsoftware,
➥Bangalore:/home/aztec:/bin/bash
```

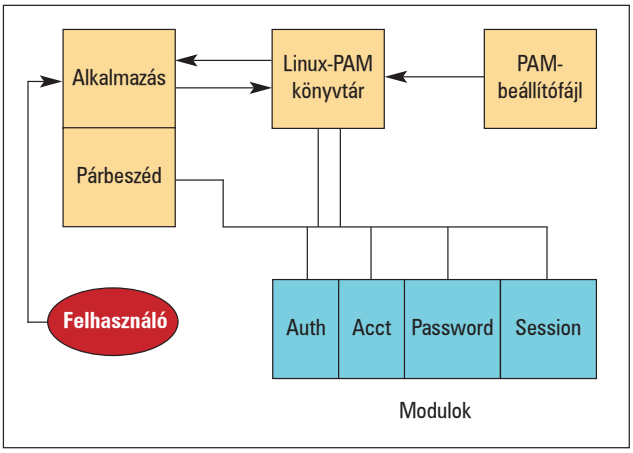
Ha a valóságban belenézünk a */etc/passwd* fájlba, természetesen inkább valami ilyesmit fogunk látni:

```
aztec:x:900:900:Aztec software,Bangalore:
➥/home/aztec:/bin/bash
```

Hová lett a titkosított jelszó?
A */etc/passwd* fájlt minden felhasználó olvashatja, ami bárkinek lehetővé teszi, hogy a rendszer összes felhasználójának titkosított jelszavát összegyűjtse. Bár a jelszavak kódoltak, jelszótörő programokat könnyű beszerezni. A növekvő biztonsági fenyegetések miatt kifejlődtek az árnyékjelszavak.
Ha a rendszeren be van kapcsolva az árnyékjelszó-rendszer, a */etc/passwd* jelszó mezőjébe mindössze egyetlen `x` kerül, a felhasználó valódi titkosított jelszava pedig a */etc/shadow* (árnyék) fájlban tárolódik. Mivel a */etc/shadow* állományt csak a rendszergazda olvashatja, a rosszakaratú felhasználók nem tudják feltörni társaik jelszavait. A */etc/shadow* sorainak tartalmát a felhasználó bejelentkező neve, titkosított jelszava, illetve néhány, a jelszó lejáratával kapcsolatos mező teszi ki. Egy jellegzetes bejegyzés a következőképpen néz ki:

```
aztec:/3GJajkg1o4125:11009:0:99999:7:::
```

A Linux megjelenésének kezdetekor, amikor még az alkalmazásoknak kellett azonosítaniuk a felhasználókat, a szükséges adatot egyszerűen csak ki kellett volna olvasni a */etc/passwd* és */etc/shadow* fájlokból. Ha meg kell változtatni a felhasználó jelszavát, elegendő a */etc/passwd* és */etc/shadow* fájlokat átszerkeszteni.
Ez a módszer, bár kétségkívül egyszerű, nehézkes és számos gondot okoz a rendszergazdáknak és az alkalmazásfejlesztőknek. Minden alkalmazásnak, amely felhasználóhitelesítést igényel, tudnia kell, hogyan érheti el a keresett adatot, amikor a többféle hitelesítési módozat közül valamelyikkel kapcsolatba kerül. Ezért aztán az előjogokat adó alkalmazásprogramok erősen kötődnek valamelyik hitelesítési módszerhez. Amint felbukkan egy új hitelesítési módszer, a régiek elavulttá válnak. Azaz ha a rendszergazda meg akarja változtatni a hitelesítési sémát, az egész alkalmazást újra kell fordítania.
E hátrányok kiküszöbölésére olyan rugalmas szerkezetet kell kidolgoznunk, amely elválaszthatja az előjogadó programok és a megfelelően biztonságos hitelesítési sémák fejlesztését.
A Linux Pluggable Authentication Module (PAM) egy ilyen megoldás, amely sikerrel szünteti meg az azonosító sémák és az alkalmazások szoros összefonódását.
Az alkalmazás a programozó szemszögéből nézve: a PAM gondoskodik a hitelesítési feladatról és ellenőrzi a felhasználó személyazonosságát. A rendszergazda szemszögéből: nagy szabadságot jelent, hogy szabadon dönthetünk, melyik hitelesítési sémát választjuk Linux-rendszerünkön PAM-alapú alkalmazásainkhoz.



A PAM-rendszer

Ábránk a PAM-rendszer négy nagy alkotórészét mutatja be. Az első elem a PAM-programkönyvtár, amely a PAM-alapú alkalmazások és modulok fejlesztéséhez szükséges felületet és függvényeket teszi elérhetővé.
A második a PAM-alapú alkalmazás, egy olyan alkalmazás, amely valamilyen szolgáltatást nyújt. A szolgáltatás elérése előtt esetleg azonosítani kell a felhasználót. A hitelesítési lépés előtt az alkalmazás kapcsolatba lép a ***Linux-PAM*** könyvtárral, és meghívja a szükséges hitelesítési szolgáltatásokat. Az alkalmazás maga semmit nem tud a felhasznált hitelesítési módszer részleteiről. Az alkalmazásnak elérhetővé kell tennie egy úgynevezett párbeszéd- (conversation) függvényt, amely lehetővé teszi a betöltött azonosító modul és az alkalmazás közti kölcsönös kapcsolattartást.
A harmadik összetevő a Pluggable Authentication Module, amely valamilyen (tetszőleges) hitelesítési módszer kezelését támogató bináris állomány. Betöltés után a modulok közvet-

lenül tudnak kapcsolatot tartani az alkalmazással az alkalmazás által elérhetővé tett párbeszédfüggvényen keresztül. Ezen keresztül cserélgethetjük a felhasználótól bekért (vagy felajánlott) szöveges adatot.
Az utolsó elem a PAM-beállítófájl. Olyan szöveges állományról van szó, amelyben a rendszergazda megadhatja az egyes alkalmazásokhoz használt hitelesítési sémákat. Linux-rendszereken ez a beállítási adat egyaránt lehet a */etc/pam.d* mappában, vagy egy sor a */etc/conf* beállítófájlban. A PAM-beállításfájlt a rendszer a PAM-könyvtárak alaphelyzetbe állítása során dolgozza fel.
A PAM-könyvtár ezután betölti a megfelelő hitelesítési modult, azt, amelyet az adott modul hitelesítési sémájához állítottunk be.

Linuxos PAM támogatású alkalmazás fejlesztése

PAM-támogatással bíró alkalmazások írásakor először meg kell hívnunk a *PAM*-könyvtár megfelelő hitelesítési függvényét. Egyúttal a párbeszédfüggvényt is meg kell adnunk, amelyen keresztül a modul közvetlenül tud kapcsolatot tartani az alkalmazással.
A PAM API-hitelesítést végző függvényei a következő három lényeges szolgáltatást tartalmazzák:

1. `pam_start()`: az első PAM-függvény, amit az alkalmazásnak meg kell hívnia. Ez a függvény állítja alaphelyzetbe a *PAM*-könyvtárat, illetve beolvassa a *PAM*-beállításfájlt, és betölti a kívánt hitelesítési modult a beállításfájlban megadott sorrendben. A *PAM*-könyvtárra irányuló mutatót ad vissza, amit az alkalmazás azután a könyvtárral való további kapcsolattartása során felhasználhat.
2. `pam_end()`: A *PAM*-könyvtár utolsó függvénye, amit az alkalmazás meghív. Visszatérésekor a *PAM*-könyvtárra irányuló mutató többé már nem lesz érvényes, és az összes hozzárendelt memóriaterület felszabadul.
3. `pam_authenticate()`: ez a függvény szolgál kapcsolófelületül a betöltött modulok hitelesítési folyamatához. Az alkalmazásnak kell meghívnia, amikor a szolgáltatást kérő felhasználót azonosítani akarja.

A hitelesítési eljárásokon kívül a PAM API az alkalmazás által hívható következő függvényeket is elérhetővé teszi:

- `pam_acct_mgmt()`: ellenőrzi, hogy a felhasználó azonosítója érvényes-e.
- `pam_open_session()`: új folyamatot kezdeményez.
- `pam_close_session()`: befejezi a futó folyamatot.
- `pam_setcred()`: kezeli a felhasználó tanúsítványait.
- `pam_chauthtok()`: megváltoztatja a felhasználóazonosító nyelvi egységét (token).
- `pam_set_item()`: menti a PAM folyamatállapotát.
- `pam_get_item()`: visszatölti a PAM folyamatállapotát.
- `pam_strerror()`: hibakiírást ad vissza.

Az alkalmazás ezeket a PAM API-eljárásokat a *security/pam_appl.h* felületen keresztül érheti el.
A párbeszédfüggvény közvetlen kapcsolatot hoz létre az alkalmazás és a betöltött modul között. Ez a modul részéről többnyire annyit tesz, hogy a felhasználótól bekéri a felhasználónevet, a jelszót és így tovább. A párbeszédfüggvény (`conv_func`) alakja a következő:

```
int conv_func (int,const struct pam_message
**, struct pam_response **,void );
```

A betöltött hitelesítési modul a `pam_message` szerkezet segítségével néhány adatot igényel az alkalmazástól, ami ezeket a szükséges adatokat a `pam_response` szerkezet segítségével juttathatja el a modulnak.
Kérdés azonban, honnan tudja a modul a párbeszédfüggvény címét? A megoldás kulcsa a párbeszédszerkezetben rejlik: `struct pam_conv`. Az alkalmazásnak a párbeszédfüggvényre mutató mutató segítségével először be kell állítania a párbeszédszerkezetet. Alaphelyzetbe hozás után a párbeszédfüggvény a `pam_start()` függvény hívásakor értékként kerül a *PAM*-könyvtárhoz. A megkapott mutatóval a modul már megkezdheti adatcseréjét a párbeszédfüggvénnyel.

Rakjuk össze mindezt!

Fejlesszünk ki egy alkalmazást, ami a pillanatnyi időt adja vissza. Legyen ez egy olyan alkalmazás, amely a szolgáltatás előtt azonosítja a felhasználót.
Először is illesszük be a megfelelő fejléceket! A PAM API csatolófelülete a *security/pam_appl.h* fejlécfájl. Ezt követően állítsuk alaphelyzetbe a párbeszédszerkezetet:

```
static struct pam_conv conv = {
      my_conv,     //function pointer to the
                      //conversation function
      NULL
};
```

Majd írjuk meg a `main()` tagfüggvényt. Ehhez előbb be kell töltenünk a *PAM*-könyvtárat. Mint már tudjuk, az alkalmazásnak meg kell hívnia a *PAM*-könyvtár tagfüggvényeit ahhoz, hogy az igényelt hitelesítési feladatot elvégezhesse. Csakhogy mi módon szerezheti meg az alkalmazás a `libpam` PAM-könyvtárleíróját? A `libpam`-et a `pam_start()` függvény állítja alaphelyzetbe, átadva neki a `service_name` (szolgáltatásnév) alkalmazásnevet, az igényelt hitelesítési szolgáltatástípust, az azonosítandó személy felhasználónevét és a `pam_conv` szerkezetre mutató mutatót. A függvény a `libpam` kezelőjével `(*pamh)` `<handle>` tér vissza, amely lehetővé teszi a *PAM* könyvtár további hívásait:

```
pam_handle_t *pamh = NULL;
int retval = 0;

retval =  pam_start(
➥"check_user",NULL,&conv,&pamh);
if(retval != PAM_SUCCESS)
      exit(0);
```

Ha a felhasználónevet nem akarjuk átadni a `pam_start()`-nak, átadhatunk `NULL`-t is. A betöltött hitelesítési modul azt a párbeszédfüggvényen keresztül később be fogja kérni a felhasználótól.
A `main()` tagfüggvény megírásának második lépése a felhasználó hitelesítése. Most jön el az igazság pillanata: eldől, hogy a felhasználó valóban az-e, akinek mondja magát. Hogyan derül ki mindez? A `pam_authenticate()` függvény szolgál csatolófelületként a betöltött modul hitelesítési módszeréhez. Ellenőrzi a felhasználó által adott felhasználónevet és jelszót a megfelelő hitelesítési modul segítségével. Siker esetén `PAM_SUCCESS`-t ad vissza, ha viszont nincs egyezés, valamilyen a hiba okát leíró kódot ad vissza:

```
retval = pam_authenticate(pamh,0);

if(retval == PAM_SUCCESS)
     printf("%s\n","Authenticated.");
```

```
else
      printf("%s\n","Authentication Failed.");
```

Megfigyelhetjük, hogy átadtuk a `pamh` kezelőt, amelyet a korábbi `pam_start()` hívás során kaptunk meg. A művelet harmadik lépése, hogy engedélyezzük a hozzáférést a kívánt szolgáltatáshoz. Most, hogy a felhasználót azonosítottuk, immár jogosul elérni az igényelt szolgáltatást. Példánkban a szolgáltatás a pillanatnyi időt adja vissza:

```
return current_time();
```

Végül eleresztjük a *PAM*-könyvtárat. Miután a felhasználó befejezte az alkalmazás használatát, a *PAM* könyvtárat ki kell törölni a memóriából. Egyúttal a `pamh` kezelőhöz rendelt memóriát is érvényteleníteni kell. Mindezt a `pam_end()` hívással érhetjük el:

```
int pam_ status = 0;
if(pam_end(pamh,pam_status) !=
PAM_SUCCESS) {
     pamh = NULL;
     exit(1);
}
```

A `pam_end()` második értékeként használt `pam_status` a modulfüggő `cleanup()` visszahívásfüggvény (callback) értéke lesz. Ezáltal a modul a leválasztása előtti utolsó pillanatban is el tudja végezni a fontos feladatokat. A függvény sikeres visszatérése esetén a `pamh` kezelőhöz tartozó valamennyi memória felszabadul.

A párbeszédfüggvény létrehozása

Az *1. listában* egy vázlatos párbeszédfüggvény megvalósítását láthatjuk. A párbeszédfüggvény hívásakor átadott érték célja a modul és az alkalmazás közti adatcsere elősegítése. A `num_msg` tárolja a `msg` mutatótömb hosszát. Sikeres visszatérés esetén a `*resp` mutató a `pam_response` szerkezetek tömbjére mutat, ahol az alkalmazás által megadott szöveget találjuk. Az üzeneteket (a modultól az alkalmazásnak) közvetítő függvényt a ***security/pam_appl.h*** határozza meg:

```
struct pam_message {
     int msg_style;
     const char *msg;
};
```

Azáltal, hogy az üzenettömb címét megkaptuk, lehetővé válik, hogy a modulból egyetlen hívással egyszerre több dolgot adjunk át az alkalmazásnak. A `msg_style` érvényes értékei:

- `PAM_PROMPT_ECHO_OFF`: szöveget kér be, kijelzés nélkül (pl.: jelszavak).
- `PAM_PROMPT_ECHO_ON`: szöveget kér be, kijelzéssel (például felhasználónév).
- `PAM_ERROR_MSG`: hibaüzenetet jelenít meg.
- `PAM_TEXT_INFO`: valamilyen szöveget jelenít meg.

A válaszadó szerkezetet (ez az alkalmazástól a modulra irányul) a ***security/pam_appl.h*** csatolásával adhatjuk meg:

```
struct pam_response {
     char *resp; int resp_retcode;
};
```

1. lista Egyszerű párbeszédfüggvény

```
int su_conv(int num_msg,
     const struct pam_message **msgm, struct
     pam_response **resp,void *appdata)
{
  int count;
  struct pam_response *r;
  char *recvpass=(char *)
       ➥malloc(20*sizeof(char));
  *(recvpass+19) = '\0';

  r = (struct pam_response*)calloc(num_msg,
            ➥sizeof(struct pam_response));

  for(count=0;count < num_msg;++count)
  {

       switch(msgm[count]->msg_style)
       {

         case PAM_PROMPT_ECHO_OFF:
           printf("%s",msgm[count]->msg);
           getPassword(recvpass);
           break;
         case PAM_PROMPT_ECHO_ON:
           printf("%s",msgm[count]->msg);
           scanf("%s",recvpass);
           break;
         case PAM_ERROR_MSG:
           printf(" %s\n",msgm[count]->msg);
           break;
         case PAM_TEXT_INFO:
           printf(" %s\n",msgm[count]->msg);
           break;
         default:
           printf("Erroneous Conversation
                  ➥(%d)\n",msgm[count]
                  ➥->msg_style);

       }

       r[count].resp_retcode = 0;
         r[count].resp = recvpass;
  }
       *resp = r;
  return PAM_SUCCESS;
}

void getPassword(char *revcbuf)
{
  int i=0;
  char buf[20];

  while((i = getch()) != '\n')
           buf[i++] = i;
  buf[i] = '\0';

  strcpy(recvbuf,buf);

}
```

Jelenleg a `resp_retcode` értékek nincsenek meghatározva; a szabályos visszatérési érték a 0.

Fordítás és összeépítés

Fordítsuk le az alkalmazást a következő paranccsal:

```
gcc -o azapp azapp.c -lpam -L/usr/azlibs
```

A */usr/azlibs* mappa helyére azt az útvonalat kell írni, ahol a Linux-PAM könyvtármodult, azaz a *libpam.so*-t találjuk. Ez a könyvtárfájl tartalmazza a *pam_appl.h*-ban megadott függvények meghatározásait.

A Pluggable Authentication Module (PAM) fejlesztése

Amikor modulfejlesztési feladatba kezdünk, először is tisztában kell lennünk a megvalósítandó modul típusával.
A modulokat működésük szerint négy különböző csoportba lehet sorolni: hitelesítés (authentication), számla (account), folyamat (session) és jelszó (password). A helyes megadáshoz e négy csoport legalább egyikéből valamennyi odatartozó függvényt meg kell határozni.
Használjuk a `pam_sm_authenticate()` függvényt, ha hitelesítő modult szeretnénk létrehozni, amely lényegében a hitelesítést végzi. Majd használjuk a `pam_sm_setcred()` függvényt. Általános esetben a hitelesítő modul a hitelesítő jelen kívül a felhasználóval kapcsolatos további adatokhoz is hozzáfér. A második függvény az ilyen adatot teszi elérhetővé az alkalmazás számára. Ezt csak akkor szabad meghívni, amikor a felhasználó már azonosította magát, de a folyamatot még nem kezdtük meg.
A számlakezelési modellmegvalósításokban a `pam_sm_acct_mgmt()` lesz az a függvény, amely a munkát elvégzi, és megmondja, a felhasználó jelenleg jogosult-e a szolgáltatás elérésére. A felhasználót e lépés előtt a hitelesítő modullal azonosítani kell.
Az üléseket kezelő modul a `pam_sm_open_session()` meghívásával kezdeményezhet üléseket. Amikor az ülést be kell fejezni, a `pam_sm_close_session()` függvényt hívjuk meg. Az is megoldható, hogy az egyik alkalmazás által megnyitott ülést egy másik zárja be. Ehhez vagy az kell, hogy a modul kizárólag a `pam_get_item()` függvénytől beszerzett adatokat használja fel, vagy az üléshez tartozó adatot az operációs rendszer valamilyen módon megőrzi (például egy fájlban).
Végül a `pam_sm_chauthtok()` a jelszókezelő modult valósítja meg, ahol a függvényt a felhasználó hitelesítési jelének (újra)beállítására használhatjuk fel (tehát megváltoztathatjuk a felhasználó jelszavát). A ***Linux-PAM*** könyvtár ezt a függvényt egymás után kétszer hívja meg. Az azonosítási jel csak a második hívás során változik meg, miután ellenőrizte, hogy egyezik a korábban begépelttel.
Ezeken a lehetőségeken túl a PAM API a következő függvényeket nyújtja a moduloknak:

- `pam_set_item()`: állapotadatot ír ki a PAM-ülésről.
- `pam_get_item()`: állapotadatot olvas vissza a PAM-ülésről.
- `pam_strerror()`: hibaszöveget ad vissza.

A modulfejlesztéshez szükséges PAM API-függvényeket a *security/pam_modules.h* csatolófelületen érhetjük el.

Rendezzük az eddigieket!

Készítsünk egy olyan modult, ami hitelesítéskezelést végez. Ehhez létre kell hoznunk a hitelesítéskezelési csoportba tartozó függvényeket. Kezdjük a szükséges fejlécek csatolásával. A Linux-PAM könyvtár csatolófelülete a *security/pam_modules.h* fejlécfájl.

2. lista A pam_sm_authenticate() megvalósításának alapjai

```
PAM_EXTERN int pam_sm_authenticate(
            pam_handle_t * pamh, int flags,
            int argc, const char **argv)
{
  unsigned int ctrl;
  int retval;
  const char *name, *p;

  /* get the user ·name· */

  retval = pam_get_user(pamh, &name, "login: ");
  if (retval == PAM_SUCCESS) {
     printf("username [%s] obtained", name);
  } else {
     printf("trouble reading username\n");
     pam_set_data(pamh, "unix_setcred_return",
                  (void *) retval, NULL);
  return retval;
  }

  /* get this user·s authentication token */

  retval = _read_password(pamh, ctrl, NULL,
    ➥"Password: ", NULL, UNIX_AUTHTOK, &p);
  if (retval != PAM_SUCCESS) {

     printf("could not read password
           ➥for %s\n",name);
     name = NULL;
     pam_set_data(pamh, "unix_setcred_return",
              ➥(void *) retval, NULL);
  return retval;
  }

  /* verify the password of this user */

  retval = _verify_password(pamh, name, p, ctrl);
  name = p = NULL;

  pam_set_data(pamh, "unix_setcred_return",
            ➥(void *) retval, NULL);
return retval;
}
```

A következő lépés a felhasználó azonosítása; a 2. *lista* mutatja be a `pam_sm_authenticate()` alapszerkezetének felépítését. E függvény célja, hogy az alkalmazástól bekérje a felhasználónevet és a jelszót, majd a jelszótitkosítási séma alapján hitelesítse a felhasználót.
A felhasználói nevet a `pam_get_user()` meghívásával szerezhetjük meg, ha az alkalmazás a `start_pam()` hívás során a jelszót eddig még nem adta volna meg.
A felhasználói név megszerzése után a felhasználótól be kell kérnünk a hitelesítő jelet (jelen esetben a jelszót), a `_read_password()` meghívásával. A függvény a felhasználó jelszavának beolvasásához az alkalmazás nyújtotta párbeszédfüggvényt használja fel.
A `_read_password()` függvény hívásához először beállítjuk

3. lista A pam_sm_setcred alapjainak megvalósítása

```
PAM_EXTERN int pam_sm_setcred(pam_handle_t *
          pamh,int flags,int argc,
          const char **argv)
{
  unsigned int ctrl;
  int retval;

  ctrl =  _set_ctrl(pamh, flags, NULL,
                  ➥argc, argv);
  retval = PAM_SUCCESS;

  printf("recovering return code from auth
        ➥call");
  pam_get_data(pamh, "unix_setcred_return",
            ➥(const void **) pretval);
  if(pretval) {
      retval = *pretval;
      free(pretval);
      printf("recovered data indicates that "
            ➥"old retval was %d", retval);
  }

  return retval;
}
```

a megfelelő adatokat a pam_message szerkezettömbben, hogy a párbeszédfüggvénnyel kapcsolatot tudjunk tartani:

```
struct pam_message msg[3], *pmsg[3];
struct pam_response *resp;
int i, replies;

/* prepare to converse by */
/* setting appropriate */
/* data in the pam_message */
/* struct array */

pmsg[i] = &msg[i];
msg[i].msg_style = PAM_PROMPT_ECHO_OFF;
msg[i++].msg = prompt1;
replies = 1;
```

Most hívjuk meg a párbeszédfüggvényt, ahol az `i` a párbeszédfüggvény válaszait jelenti:

```
retval = converse(pamh, ctrl, i, pmsg, &resp);
```

A `converse()` függvény tulajdonképpen a modul kezelőfelülete az alkalmazás által nyújtott párbeszédfüggvényhez. Végül meghívjuk a `_verify_password()` függvényt, amivel azonosíthatjuk a felhasználót. A `_verify_password()` függvény pedig a megfelelő titkosítási séma alapján azonosítja a felhasználó okmányait.

A felhasználói okmányok (Credentials) beállítása

Az azonosító modul általában több felhasználóval kapcsolatos adathoz férhet hozzá, mint amennyi az azonosítójelben megtalálható. A `pam_sm_setcred` függvényt használhatjuk arra, hogy ezeket az adatokat az alkalmazás számára is hozzáfér-

4. lista Példa a converse() megvalósítására

```
static int converse(pam_handle_t * pamh,
          int ctrl, int nargs, struct
          pam_message **message, struct
          pam_response **response)
{
  int retval;
  struct pam_conv *conv;

  retval = pam_get_item(pamh, PAM_CONV,
           ➥(const void **) &conv);
  if (retval == PAM_SUCCESS) {

      retval = conv->conv(nargs, (const
               ➥struct pam_message **)
               ➥message, response,
               ➥conv->appdata_ptr);

      printf("visszatØrØs az alkalmazÆsb l "
             "pÆrbeszØd ");

      if (retval != PAM_SUCCESS &&
          on(UNIX_DEBUG, ctrl)) {
          printf("pÆrbeszØd hiba\n",);

      }
  printf("kØszenÆll, hogy visszatØrjen a
        ➥modulb l " "pÆrbeszØd");

  return retval;
}
```

hetővé tegyük. A `pam_sm_setcred` egyszerű használatára látunk példát a *3. listában*. A példaként felhozott függvény megvalósításában a `pam_sm_authenticate()` visszatérési értékét egész egyszerűen átadjuk az alkalmazásnak.

Kapcsolattartás az alkalmazással

A `converse()` függvény kezelőfelületként működik a modulalkalmazás-párbeszéd során. A `converse()` függvény megvalósítására látunk példát a *4. listában*.
A párbeszédfüggvény mutatóját a `pam_get_item(pamh,PAM_CONV,&item)` hívással érhetjük el. Ezt a mutatót felhasználva a modul közvetlenül beszélgetni kezdhet az alkalmazással.

Statikusan betöltött modulok kezelése

Előfordulhat, hogy egy modul a `libpam`-be statikusan van lefordítva (statically linked). Ez tulajdonképpen valamennyi modulra igaz, amelyek az alap PAM-terjesztésbe tartoznak. Hogy statikusan be lehessen fordítani, a modulnak a függvényeit leíró adatot úgy kell közzétennie, hogy az ne ütközhessen más modulokkal.
A statikus modulokhoz szükséges további kódot érdemes `#ifdef PAM_STATIC` és `#endif` részek közé zárni.
A statikus kód egyetlen szerkezetet tartalmaz: a `struct pam_module`-t. Ezt `_pam_modname_modstruct`-nak nevezik, ahol a `modname` a fájlrendszeren használt modulnév, a bevezető könyvtárnév (általában */usr/lib/security/*) és az utótag (általában *.so*) elhagyásával.

```
#ifdef PAM_STATIC
struct pam_module _pam_unix_auth_modstruct = {
    "pam_unix_auth",
    pam_sm_authenticate,
    pam_sm_setcred,
    NULL,
    NULL,
    NULL,
    NULL,
};
#endif
```

Modulunk immár statikus vagy dinamikus elemként is lefordítható. Fordítsuk le a következő parancsokkal:

```
gcc -fPIC -c pam_module-name.c
➥ld -x  shared -o
➥pam_module-name.so
➥pam_module-name.o
```

Az alkalmazás biztonságossá tétele: a PAM-beállítófájl

A Linux-PAM által vezérelt rendszerbiztonság szempontjából érdekes helyi beállításfájlok a két lehetséges hely közül az egyiken helyezkednek el: egyetlen rendszerfájlban (*/etc/pam.conf*) vagy a */etc/pam.d/* könyvtárban.
A */etc/pam.conf* fájl általános formátumának szerkezete szolgáltatásnév–modultípus–vezérlőzászló (flag) modulútvonal alakú. Megtehetjük azt is, hogy az alkalmazáshoz tartozó PAM-beállításokat a */etc/pam.d* könyvtár külön könyvtárába helyezzük. Ebben az esetben a formátum: `module-t pus`–vezérlőzászló modulútvonal alakot vesz fel. A szolgáltatásnév a fájl nevévé válik. A szolgáltatás neve gyakran az adott alkalmazás hagyományos neve, például azServer.

Modultípus

Négy modultípus létezik: azonosító (auth), számla (account), folyamat (session) és a jelszó (password).

- `auth`: meghatározza, hogy a felhasználó valóban az-e, akinek vallja magát. Ezt általában jelszóval végezzük, de alkalmazható ennél kifinomultabb módszer is, például a biológiai jellegzetességek vizsgálata.
- `account`: meghatározza, hogy a felhasználó jogosult-e használni a szolgáltatást, lejárt-e a jelszava és így tovább.
- `password`: lehetőséget nyújt a felhasználónak, hogy azonosítójelét megváltoztassa. Általában itt is a jelszóról van szó.
- `session`: azok a dolgok, amelyeket a felhasználó azonosítása előtt, és után meg kell tenni. Ide tartozik a felhasználó saját könyvtárának befűzése és leválasztása, a be- és kilépés naplózása, továbbá a felhasználó által elérhető szolgáltatások korlátozása, illetve e korlátozások feloldása. Továbbá négy vezérlőzászló is létezik: `required` (megkövetelt), `requisite` (előfeltétel), `sufficient` (elégséges) és `optional` (tetszőleges, elhagyható).
- `required`: azt jelzi, hogy a modul sikere elengedhetetlen a modultípus-beállítás sikeréhez. A modul sikertelenségét a felhasználó addig nem veheti észre, amíg az összes hátralévő (azonos modultípusú) modul végre nem hajtódott.
- `requisite`: azonos a `required`-del, azzal a különbséggel, hogy a modul sikertelensége esetén az eredményt közvetlenül az alkalmazásnak adja vissza.
- `sufficient`: ha a modul sikerrel járt valamint az összes korábbi modul szintén sikeres volt, további modulokat már nem kell meghívni.
- `optional`: jelzi, hogy a modul sikere nem feltétlenül szükséges a felhasználónak az alkalmazáshoz történő hozzáféréséhez. Értékét csak akkor kell figyelembe venni, ha az előző modulok sorában semmilyen más egyértelmű siker vagy hiba nem fordult elő.

A dinamikusan betöltött objektumfájlok (azaz maga a betölthető modul) elérési útja a modulelérési út. Ha a modulelérési út első karaktere /, az teljes elérési utat feltételez. Ha nem ez a helyzet, a modulútvonal az alapértelmezett útvonallal egészül ki, amely a */usr/lib/security*.
A modul meghívásakor átadott értékek, jelek listája nagyon hasonló ahhoz, ahogy a Linux-héjprogram dolgozza fel a parancsokat. Általában a helyes értékek elhagyhatók és modulfüggők. Végül a beállításfájl megírásához át kell szerkesztenünk a */etc/pam.conf* fájlt és be kell illesztenünk a következő kódot:

```
check_user  auth  required
/lib/security/pam_unix.so
```

Ez azt fogja jelenteni, hogy a szolgáltatásnevekhez a `check_user` és `auth` modultípus elengedhetetlenül szükséges (`required`). A hitelesítéstípus elvégzéséhez betöltendő modul a *pam_unix.so*, amely a */lib/security/* könyvtárban található.

Összegzés

A Linux-PAM használatával többé már nem vagyunk egyetlen azonosítási sémához sem kötve, terveinknek kizárólag saját képzeletünk szabhat határt.

A listák megtalálhatóak a 41. CD Magazin/PAM könyvtárában.

Linux Journal 2002. október, 102. szám

Savio Fernandes (savferns21@yahoo.com) az Aztec Software and Technologies Ltd. alkalmazásában dolgozik programfejlesztőként. Nagy Linux-rajongó és maga is dolgozott a Linux biztonsági rendszerén. Szabadidejében szintetizátorozik és focizik.

KLM Reddy (klmreddy@yahoo.com) az Indiai Technológiai Intézetben programfejlesztőként dolgozik. Számos titkosítási eljárás fejlesztésében részt vett már. Szabadidejében szívesen játszik kabaddit és néz filmeket.

Kapcsolódó címek

Andrew G. Morgan: The Linux-PAM application Developers' Guide ➲ http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_appl.html

Andrew G. Morgan: The Linux-PAM Module Writers' Guide ➲ http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam_modules.html

Linux-PAM Source Code ➲ ftp.kernel.org/pub/linux/libs/pam/