HRVATSKA AKADEMSKA I ISTRAŽIVAČKA MREŽA
CROATIAN ACADEMIC AND RESEARCH NETWORK

PAM autentikacijski modul

CCERT-PUBDOC-2000-09-03

CARNet CERT u suradnji s LS&S

Sigurnosni problemi u računalnim programima i operativnim sustavima područje je na kojem CARNet CERT kontinuirano radi.

Rezultat toga rada ovaj je dokument koji je nastao suradnjom CARNet CERT-a i LS&S-a, a za koji se nadamo se da će Vam koristiti u poboljšanju sigurnosti Vašeg sustava.

CARNet CERT, www.cert.hr - nacionalno središte za **sigurnost računalnih mreža** i sustava.

LS&S, www.lss.hr - laboratorij za sustave i signale pri Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

Sadržaj

1. Uvod

Dodatni autentikacijski modul (eng. *Pluggable Authentication Module* – PAM; u nastavku dokumenta PAM) predstavlja način na koji programi mogu pokretati servise vezane uz korisničku autentikaciju i održavanje korisničkih računa. Autentikacija se obično provodi metodom propisnog odgovora na upit (eng. *challenge-response*). Korištenjem PAM-a administrator može prilagođavati metode korištene od programa za autentikaciju, bez potrebe za ponovnim njihovim prevođenjem.
PAM sistem sastoji se od četiri dijela. Prvi dio, `libpam`, je biblioteka koja implementira PAM API. Drugi dio je PAM konfiguracijska datoteka; `/etc/pam.conf`. Treći dio sastoji se od skupa binarnih objekata koji se mogu dinamički učitavati i koji obavljaju autentikaciju. Posljednji dio sastoji se od sistemskih naredbi koje koriste (ili bi trebale) PAM API kao što su `login`, `su`, `ftp`, `telnet` itd.

2. Libpam biblioteka

Autentikacijske rutine PAM API-ja sastoje se od tri primarne funkcije:

```
pam_start( const char *service_name, const char *username,
           const struct pam_conv *conv, pam_handle_t **pamh_p );

pam_end( pam_handle_t *pamh, int exit_status );

pam_authenticate( pam_handle_t *pamh, int flags );
```

Funkcija `pam_start()` i `pam_end()` započinju, odnosno završavaju PAM sjednicu. Argumenti funkcije `pam_start()` su sljedeći:

- `service_name` – string koji označava određeni servis kako je definirano u pam.conf. datoteci,
- username – korisničko ime korisnika koji se treba autenticirati,
- conv – kazaljka na pam_conv strukturu,
- pamh_p – dvostruka kazaljka na pam_handle_t strukturu; PAM će alocirati i osloboditi memoriju za strukturu i aplikacije toj strukturi ne bi smjele direktno pristupati, ona se koristi unutar PAM-a za rad sa višestrukim PAM sjednicama.

Struktura `pam_conv` izgleda na sljedeći način:

```
struct pam_conv {
    int (*conv)(int num_msg, const struct pam_message **msg,
                struct pam_response **resp, void *appdata_ptr);
    void *appdata_ptr;
}
```

Kazaljka `*conv` pokazuje na funkciju u aplikaciji poznatoj kao PAM funkcija za konverzaciju koja će biti opisana kasnije. Kazaljka `appdata_ptr` pokazuje na aplikacijsko-specifične podatke i ne koristi se često.
Argumenti funkcije `pam_end()` sastoje se od istog `pam_handle_t*` definiranog pozivom `pam_start()` i od izlaznog statusa. Izlazni status je obično `PAM _SUCCESS`, ali može biti i drugačiji, ovisno o razlogu neuspješan PAM sjednice. Funkcija `pam_end()` će osloboditi memoriju vezanu uz `pam_handle_t*` i pokušati ponovo iskoristiti rukovatelj, što obično rezultira segmentacijskom pogreškom.
Funkcija `pam_authenticate()` sastoji se od `pam_handle_t*` kazaljke popunjene funkcijom `pam_start()` i opcionalnih zastavica koje mogu biti proslijeđene unutar PAM API-ja.
Neke druge funkcije unutar PAM API-ja dostupne aplikacijama jesu sljedeće (za kompletne informacije valja proučiti sistemsku dokumentaciju):

- `pam_set_item()` – upisuje informaciju o stanju PAM sjednice,
- pam_get_item() – čita informaciju o statusu PAM sjednice,

- pam_acct_mgmt() – provjerava valjanost korisničkog računa trenutnog korisnika,
- pam_open_session() – započinje novu sjednicu,
- pam_close_session() – zatvara trenutnu sjednicu,
- pam_setcred() – upravlja korisničkim pravima
- pam_chautok() – mijenja korisnički autentikacijski znak,
- pam_strerror() – vraća string sa pogreškom, slično kao perror().

3. Konfiguracijska biblioteka

PAM konfiguracijska datoteka obično se nalazi u `/etc/pam.conf` i podijeljena je u četiri dijela: autentikacija, upravljanje korisničkim računima, upravljanje sjednicama i upravljanje zaporkama. Tipični redak datoteke izgleda na sljedeći način:

```
login       auth       required       /usr/lib/security/pam_unix.so.1
try_first_pass
```

Prvo polje je ime servisa. To je servis na koji se referencira prvi argument funkcije `pam_start()`. Ukoliko servis koji funkcija `pam_start()` traži nije definiran u `pam.conf` datoteci, biti će korišten *default* servis "*other*". Druga imena servisa mogu biti `su` ili `rlogin`. Ukoliko je ime servisa definirano više puta, moduli su "ustoženi" (eng. *stacked*) i ponašanje će biti određeno vrijednošću trećeg polja kako je dalje opisano.
Drugo polje označava koju specifičnu akciju koju određeni servis poduzima. Dozvoljene vrijednosti jesu "`auth`" za autentikaciju, "`account`" za upravljanje korisničkim računima, "`session`" za upravljanje sjednicama i "`password`" za upravljanje zaporkama. Sve aplikacije ne moraju pristupati svim akcijama. Na primjer, `su` zahtijeva samo "`auth`" akciju, dok `passwd` treba samo "`password`" akciju.
Treće polje predstavlja kontrolno polje koje valja detaljnije objasniti. Ono indicira ponašanje PAM okruženja ukoliko korisnik ne prođe autentikaciju. Dozvoljene vrijednosti jesu "*requisite*", "*required*", "*sufficient*" i "*optional*".
"Requisite" znači da, ukoliko korisnik ne prođe autentikaciju, PAM okruženje odmah vraća neuspješan rezultat i ne poziva nikakve druge module.

- "Required" bilježi ukoliko korisnik ne prođe autentikaciju, PAM okruženje vraća neuspješan rezultat tek nakon što pozove sve module. To se radi na taj način da korisnik ne zna na kojem modulu je autentikacija bila neuspješna. Za uspješnu autentikaciju svi "required" moduli moraju vratiti pozitivan rezultat.
- "Optional" znači da će korisniku biti dozvoljen pristup i ukoliko je autentikacija neuspješna. U slučaju neuspjeha, poziva se sljedeći modul na stogu.
- "Sufficient" znači da ukoliko korisnik prođe određeni modul, PAM okruženje trenutačno vraća pozitivan rezultat, čak i ukoliko sljedeći moduli imaju "*requisite*" ili "*required*" kontrolne veličine. Kao i kod "*optional*" veličine, "*sufficient*" dozvoljava pristup čak i ako autentikacija da negativan rezultat.

Može se uočiti da ukoliko bilo koji modul vrati pozitivan rezultat, korisnik će biti autenticiran, a jedina iznimka je ukoliko je prethodni "*required*" modul dao negativan rezultat.
Četvrto polje u `pam.conf` datoteci je put do autentikacijskog modula. Put može biti drugačiji u različitim sistemima. Na primjer, PAM moduli nalaze se u `/usr/lib` u Linux-PAM implementaciji, dok se na Solarisu PAM moduli nalaze u `/usr/lib/security`.
Peto polje je lista opcija ovisnih o modulu, koje se prosljeđuju autentikacijskom modulu prilikom njegovog poziva.

4. Moduli

Svaki PAM modul je ustvari biblioteka koja sadrži specifične funkcije. Te funkcije pozivaju se iz PAM okruženja. Funkcije koje biblioteka sadrži jesu sljedeće:

- `pam_sm_authenticate()`,
- pam_sm_setcred(),

- pam_sm_acct_mgmt(),
- pam_sm_open_session(),
- pam_sm_close_session(),
- pam_sm_chauthtok().

Ukoliko implementacija ne podržava specifičnu akciju unutar modula, modul treba vratiti PAM_SUCCESS za tu akciju. Na primjer, ukoliko modul ne podržava upravljanje korisničkim računima, funkcija pam_sm_acct_mgmt() vraća PAM_SUCCESS.
Deklaracija za pam_sm_authenticate() je sljedeća:

```
extern int pam_sm_authenticate( pam_handle_t *pamh, int flags,
       int argc, char **argv);
```

Argument pamh* je kazaljka na PAM rukovatelj koje je definiran okruženjem, flags je skup zastavica koje se šalju okruženju aplikacijskim pozivom funkciji pam_authenticate(), dok su argc i argv broj i vrijednost opcionalnih argumenata za određeni servis u pam.conf.
Jednostavna funkcija pam_sm_authenticate() za pam_unix modul mogla bi izgledati na sljedeći način:

```
#include <security/pam_modules.h>
#include <...>

extern int
pam_sm_authenticate( pam_handle_t *pamh, int flgs, int c, char **v )
{
        char *user;
        char *passwd;
        struct passwd *pwd;
        int ret;

        /* ignore flags and optional arguments */

        if ( (ret = pam_get_user( ..., &user )) != PAM_SUCCESS )
           return ret;
        if ( (ret = pam_get_pass( ..., &passwd )) != PAM_SUCCESS )
           return ret;
        if ( (pwd = getpwnam(user)) != NULL ) {
           if ( !strcmp(pwd->pw_passwd, crypt(passwd)) )
              return PAM_SUCCESS;
           else
              return PAM_AUTH_ERR;
        }

        return PAM_AUTH_ERR;
}
```

Naravno, ova funkcija je vrlo pojednostavljena, ali demonstrira osnovnu funkcionalnost pam_sm_authenticate(). Funkcija dohvaća korisničko ime i zaporku iz okruženja, zatim dohvaća korisnikovu kriptiranu zaporku, te konačno poziva crypt() funkciju za korisnikovu zaporku, te rezultat uspoređuje sa kriptiranom zaporkom u sistemu. Ta usporedba određuje uspjeh ili neuspjeh. Funkcije pam_get*() su pozivi okruženju i mogu se razlikovati od sistema do sistema.

5. Aplikacija

Samu PAM aplikaciju je prilično jednostavno implementirati. Dijelovi koji koriste PAM moraju sadržavati par funkcija, pam_start() i pam_end() i funkciju za PAM konverzaciju. Na sreću, PAM

API je dobro definiran i stabilan, tako da konverzacijska funkcija ne predstavlja problematični dio kôda. Jednostavna implementacija su naredbe bila bi sljedeća:

```
#include <security/pam_appl.h>
#include <...>

int su_conv(int, const struct pam_message **,
            struct pam_response **, void *);

static struct pam_conv pam_conv = { su_conv, NULL };

int
main( int argc, char **argv )
{
        pam_handle_t *pamh;
        int ret;
        struct passwd *pwd;

        /* assume arguments are correct and argv[1] is the username
*/

        ret = pam_start("su", argv[1], &pam_conv, &pamh);
        if ( ret == PAM_SUCCESS )
           ret = pam_authenticate(pamh, 0);
        if ( ret == PAM_SUCCESS )
           ret = pam_acct_mgmt(pamh, 0);

        if ( ret == PAM_SUCCESS ) {
           if ( (pwd = getpwnam(argv[1])) != NULL )
              setuid(pwd->pw_uid);
           else {
              pam_end(pamh, PAM_AUTH_ERR);
              exit(1);
           }
        }
        pam_end(pamh, PAM_SUCCESS);

        /* return 0 on success, !0 on failure */
        return ( ret == PAM_SUCCESS ? 0 : 1 );
}

int
su_conv(int num_msg, const struct pam_message **msg,
        struct pam_response **resp, void *appdata)
{
        struct pam_message *m = *msg;
        struct pam_message *r = *resp;

        while ( num_msg-- )
        {
                switch(m->msg_style) {

                case PAM_PROMPT_ECHO_ON:
                     fprintf(stdout, "%s", m->msg);
                     r->resp = (char *)malloc(PAM_MAX_RESP_SIZE);
                     fgets(r->resp, PAM_MAX_RESP_SIZE-1, stdin);
                     m++; r++;
                     break;
```

```
                case PAM_PROMPT_ECHO_OFF:
                        r->resp = getpass(m->msg);
                        m++; r++;
                        break;

                case PAM_ERROR_MSG:
                        fprintf(stderr, "%s\n", m->msg);
                        m++; r++;
                        break;

                case PAM_TEXT_MSG:
                        fprintf(stdout, "%s\n", m->msg);
                        m++; r++;
                        break;

                default:
                        break;
                }
        }
        return PAM_SUCCESS;
}
```

Funkcija `su_conv()` je konverzacijska funkcija, odnosno omogućava "konverzaciju" između korisnika i modula. Svaka `pam_message` struktura sadrži stil poruke koji indicira koji tip podataka modul zahtijeva. `PAM_PROMPT_ECHO_ON` i `PAM_PROMPT_ECHO_OFF` indiciraju da modul zahtijeva više informacija od korisnika. Odziv definira sam modul. U slučaju `PAM_PROMPT_ECHO_OFF` modul obično zahtijeva zaporku. Na aplikaciji je da onemogući prikaz znakova. `*_MSG` slučajevi se koriste za prikaz poruka na korisničkom terminalu.
Jednostavnost PAM konverzacije je u tome što svaki znakovni izlaz može biti zamijenjen pozivima različitim sistemima za prikaz bez potrebe za promjenom autentikacijskog modula. Na primjer, funkcija `getpass()` može biti zamijenjena sa npr. `get_gui_passwd()`, ukoliko se želi implementirati GUI orijentirana `su` naredba.
Prava konverzacijska funkcija mora biti robusna. Također, Linux-PAM implementacija sadrži `misc_conv()` funkciju za interakciju putem komandne linije, koja se treba koristiti ukoliko je standardna konverzacija dovoljna. Konačno, aplikacija bi trebala osloboditi svu alociranu memoriju.

6. Konfiguriranje modula

Moguće je definirati korisničke autentikacijske rutine. Na primjer, moguće je modificirati ranije opisani modul da prilikom autentikacije *root* korisnika zahtijeva unos druge zaporke.

```
extern int
pam_sm_authenticate( pam_handle_t *pamh, int flgs, int c, char **v )
{
        char *user;
        char *passwd;
        struct passwd *pwd;
        int ret;

        /* ignore flags and optional arguments */

        if ( (ret = pam_get_user( ..., &user )) != PAM_SUCCESS )
           return ret;
        if ( (ret = pam_get_pass( ..., &passwd )) != PAM_SUCCESS )
           return ret;
        if ( (pwd = getpwnam(user)) != NULL ) {
```

```
            if ( !strcmp(pwd->pw_passwd, crypt(passwd)) )
               ret = PAM_SUCCESS;
            else
               ret = PAM_AUTH_ERR;
         }

         if ( !strcmp(user, "root") ) {
            pam_display_message("root user must enter secondary
password");
            if ( (ret = pam_get_pass( ..., &passwd )) != PAM_SUCCESS
)
               return ret;
            if ( !strcmp(get_second_root_pwd(), crypt(passwd)) )
               ret = PAM_SUCCESS;
            else
               ret = PAM_AUTH_ERR;
         }

         return ret;
}
```

Ovdje se pretpostavlja da postoji funkcija `get_second_root_pwd()` koja vraća neku tajnu šifriranu zaporku. Naravno ovaj primjer je malo nespretan, ali demonstira slobodu koja postoji pri izradi PAM modula. Moduli, pošto postoje u korisničkom prostoru, imaju pristup svim bibliotečnim funkcijama. Ukoliko je npr. a računalo priključen neki biometrijski uređaj i postoji bibliotečna funkcija koja mu može pristupiti, moguće je napraviti sljedeći PAM modul.

```
         thumbprint_t *tp;
         tp = scan_thumbprint();
            /* or scan_retina() if you like James Bond */
         if ( match_print_to_user(tp, user) )
            return PAM_SUCCESS;
```

7. Zaključak

PAM moduli nikako nisu ograničeni pozivom `crypt()` ili neke slične funkcije za korisničku zaporku. Postoji potpuna korisnička sloboda u njihovoj definiciji i konfiguraciji, tako da fizička ograničenja ustvari niti ne postoje.