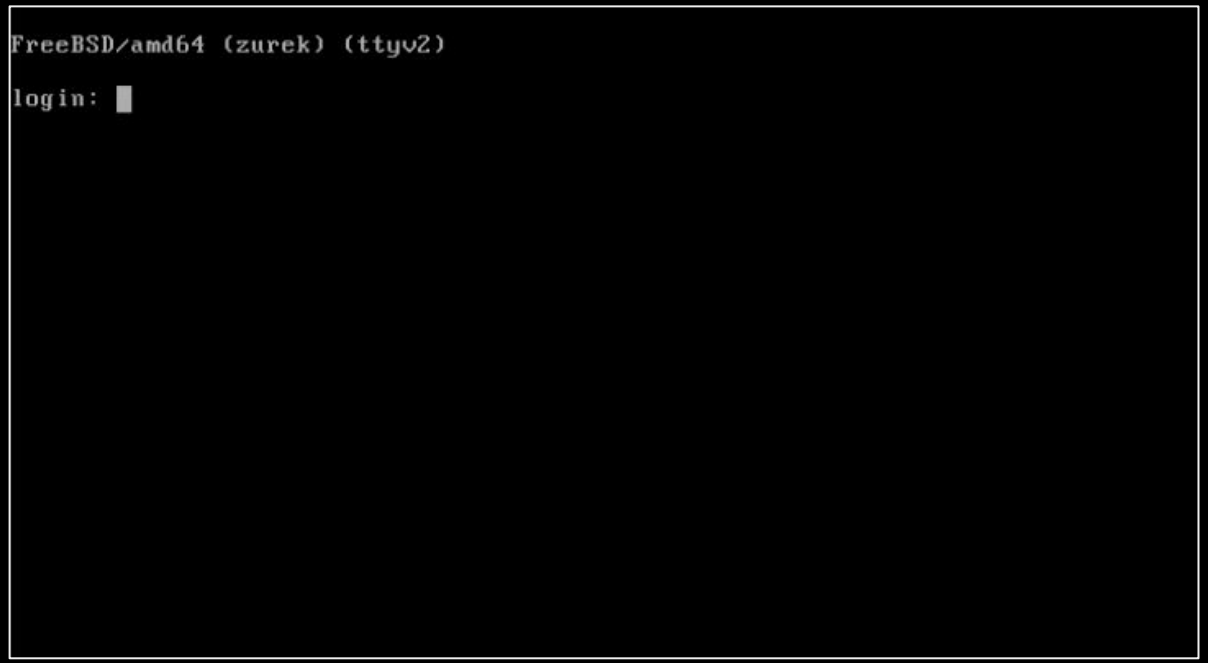
FreeBSD PAM: programowanie własnych modułów uwierzytelniania

Jarosław Żurek

PAM - Pluggable Authentication Module

- zapewnienia usługi uwierzytelniania w sposób ustandaryzowany i bezpieczny;
- modułowa budowa;
- upraszcza konfigurowanie i zarządzanie aktualnymi ustawieniami polityk uwierzytelnienia;
- wdrożenie nowego rozwiązania to jedynie instalacja modułu oraz modyfikacja wpisu w konfiguracji PAM;

Przykładowy moduł PAM we FreeBSD



Obraz 1: Uruchomienie procesu `login(1)`, który wykorzystuje PAM.
Źródło: Opracowanie własne.

Przykładowe moduły PAM we FreeBSD

`pam_unix(8)` (`pam_unix.so`):

- moduł usługi uwierzytelnienia na konto w systemie UNIX;
- używa `getpwnam(3)`, w celu uzyskania danych o użytkowniku ze struktury passwd (m.in. do weryfikacji hasła).

`pam_exec(8)` (`pam_exec.so`):

- pierwszy argument - nazwa programu do wykonania, reszta argumentów - argumenty wywoływanego polecenia;
- można go użyć go do uruchomienia programu podczas logowania (np. montowanie katalog domowy użytkownika).

`pam_guest(8)` (`pam_guest.so`)

- logowanie tzw. gości, z wykorzystaniem wcześniej ustalonych nazw. Do haseł można wprowadzać różnorodne polityki (domyślnie - zezwolenie na jakiekolwiek hasło, o ile nazwa konta jest zdefiniowana jako gość).
- zastosowanie w np. anonimowe loginy do usługi FTP.

Więcej modułów PAM: https://www.freebsd.org/doc/en/articles/pam/pam-freebsd-modules.html

Facilities, a dostępne prymitywy (1/2)

`auth` - uwierzytelniania użytkownika i ustalenie danych uwierzytelniających dla konta.

- **`pam_sm_authenticate(3)`** - uwierzytelnia, zwykle poprzez żądanie tokena, porównując go z wartością z bazy lub uzyskaną z serwera;
- **`pam_sm_setcred(3)`** – konfiguje poświadczeń użytkownika stanowiące o jego unikalności np. dodatkowe członkostwo w grupie, Kerberos ticket;

`account` - weryfikacja konta po uwierzytelnieniu.

- **`pam_sm_acct_mgmt(3)`** - realizuje procedury walidujące konto po uwierzytelnieniu np. sprawdza, czy żądane konto jest dostępne, wygaśnięte.

Facilities, a dostępne prymitywy (2/2)

`session` - po zestawieniu sesji realizacja zadań dla uwierzytelnionego użytkownika.

- **`pam_sm_open_session(3)`** – zestawia sesje i realizuje określone zadania: wpis do utmp, start agenta SSH, montowanie katalogu itp.;
- **`pam_sm_close_session(3)`** – kończy sesję i realizuje określone zadania: wpis do utmp, zatrzymaj agenta SSH, odmontowanie katalogu itp.;

`password` - zarządzanie hasłami związanymi z kontem (np. hasło wygasło lub użytkownik chce je zmienić).

- **pam_sm_chauthtok(3)** - zmienia token uwierzytelniania (opcjonalnie sprawdzenie trudności hasła, czy był wcześniej używany, itp).

Pliki konfiguracyjne:

`/etc/pam.d/`

`/etc/pam.conf` (ignorowany, jeśli katalog `pam.d` istnieje).

Więcej o konfiguracji: https://www.freebsd.org/doc/en/articles/pam/pam-config.html

Krótkie wprowadzenie (1/2)

- FreeBSD - **system wielozadaniowy**

- **program** - proces nadrzędny (uruchamiający) -> rodzic
- **podprogram** - podproces (potomny) -> dziecko

Krótkie wprowadzenie (2/2)

```
init(8)                    - główny proces rozruchowy
     |
     -> getty(8)           - powołanie tty(4) (uruchomienie /dev/ttyX)
          |
          -> login(1)
             |
             -> PAM        - pobranie wpisu o wskazanym użytkowniku z /etc/passwd
                           - weryfikacja hasła z bazą master.passwd(5) [0600 root:wheel]
```

Współcześnie **login(1):**

- korzysta z PAM API;
- możliwość dostosowywania przez modyfikację w plikach konfiguracyjnych PAM;

Więcej: `init(8), getty(8), tty(4), ttys(5), login(1), master.passwd(5), pam_unix(8), pam(3), pam.d(5).`

Jak wygląda `pam_unix(8)` dla programisty?

```
PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{
        login_cap_t *lc;
        struct passwd *pwd;
        int retval;
        const char *pass, *user, *realpw, *prompt;

        if (openpam_get_option(pamh, PAM_OPT_AUTH_AS_SELF)) {
                user = getlogin();
        } else {
                retval = pam_get_user(pamh, &user, NULL);
                if (retval != PAM_SUCCESS)
                        return (retval);
        }
        pwd = getpwnam(user);

        PAM_LOG("Got user: %s", user);

        if (pwd != NULL) {
                PAM_LOG("Doing real authentication");
                realpw = pwd->pw_passwd;
                if (realpw[0] == '\0') {
                        if (!(flags & PAM_DISALLOW_NULL_AUTHTOK) &&
                            openpam_get_option(pamh, PAM_OPT_NULLOK))
                                return (PAM_SUCCESS);
                        PAM_LOG("Password is empty, using fake password");
                        realpw = "*";
                }
                lc = login_getpwclass(pwd);
        } else {
                PAM_LOG("Doing dummy authentication");
                realpw = "*";
                lc = login_getclass(NULL);
        }
        prompt = login_getcapstr(lc, "passwd_prompt", NULL, NULL);
        retval = pam_get_authtok(pamh, PAM_AUTHTOK, &pass, prompt);
        login_close(lc);
        if (retval != PAM_SUCCESS)
                return (retval);
        PAM_LOG("Got password");
        if (strnlen(pass, _PASSWORD_LEN + 1) > _PASSWORD_LEN) {
                PAM_LOG("Password is too long, using fake password");
                realpw = "*";
        }
        if (strcmp(crypt(pass, realpw), realpw) == 0)
                return (PAM_SUCCESS);

        PAM_VERBOSE_ERROR("UNIX authentication refused");
        return (PAM_AUTH_ERR);
}
```

Listing 2: Implementacja `pam_sm_authenticate(3)` z modułu `pam_unix(8)`, **Źródło**: `lib/libpam/modules/pam_unix/pam_unix.c`

Implementacja dostępnych prymitywów

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


PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{

        return (PAM_SUCCESS);
}


PAM_EXTERN int
pam_sm_setcred(pam_handle_t *pamh __unused, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{

        return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_acct_mgmt(pam_handle_t *pamh __unused, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{

        return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_chauthtok(pam_handle_t *pamh __unused, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{

        return (PAM_SUCCESS);
}
```

```c
PAM_EXTERN int
pam_sm_open_session(pam_handle_t *pamh __unused, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{

        return (PAM_SUCCESS);
}

PAM_EXTERN int
pam_sm_close_session(pam_handle_t *pamh __unused, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{

        return (PAM_SUCCESS);
}

PAM_MODULE_ENTRY("pam_myownpam");
```

Listing 1: Szkielet implementacji biblioteki współdzielonej modułu PAM.
Źródło: Opracowanie własne.

Pierwsze kroki z własnym modułem PAM (1/5)

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

#include <string.h>

#define PASS_OK "p4ssw0rd"


PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{
        int err;
        const char *password, *user;
        const char pass_ok[] = "s3cur3";

        /* Get login name. */
        err = pam_get_user(pamh, &user, NULL);
        if (err != PAM_SUCCESS) {
                return (err);
        }

        /* Get password. */
        err = pam_get_authtok(pamh, PAM_AUTHTOK, &password, "OTP: ");
        if (err == PAM_CONV_ERR) {
                return (err);
        }

        /* Check if password is matched. */
        if ((strcmp(password, "h4ck3d")) == 0 ||
            (strcmp(password, PASS_OK)) == 0 ||
            (strcmp(password, pass_ok)) == 0) {
                return (PAM_SUCCESS);
        }

        return (PAM_AUTH_ERR);
}
```

Listing 3: Implementacja niebezpiecznej funkcji do uwierzytelniania.
Źródło: Opracowanie własne.

```
# objdump -j.data -j.rodata -s pam_myownpam.so

pam_myownpam.so:     file format elf64-x86-64-freebsd

Contents of section .rodata:
 07f6 73336375 7233004f 54503a20 00683463  s3cur3.OTP: .h4c
 0806 6b336400 70347373 77307264 00        k3d.p4ssw0rd.
Contents of section .data:
 200b40 00000000 00000000 480b2000 00000000  ........H. .....
 200b50 48092000 00000000                     H. .....
# 
```

Obraz 2: Podgląd sekcji `.rodata`, zawierającej zahardkodowae stringi, przy użyciu narzędzia `objdump(1)`.
Źródło: Opracowanie własne.

```
# strings pam_myownpam.so
P`^B
_fini
_init
_Jv_RegisterClasses
__cxa_finalize
pam_sm_acct_mgmt
pam_sm_authenticate
pam_sm_chauthtok
pam_sm_close_session
pam_sm_open_session
pam_sm_setcred
pam_get_authtok
pam_get_user
strcmp
libc.so.7
_edata
__bss_start
_end
pam_myownpam.so
FBSD_1.0
AVSH
[A^]
fff.
s3cur3
OTP:
h4ck3d
p4ssw0rd
# 
```

Obraz 3: Podgląd biblioteki współdzielonej przy użyciu narzędzia `strings(1)`.
Źródło: Opracowanie własne.

Pierwsze kroki z własnym modułem PAM (2/5)

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

#include <string.h>

#define PASS_OK "p4ssw0rd"


PAM_EXTERN int
pam_sm_authenticate(pam_handle_t *pamh, int flags __unused,
    int argc __unused, const char *argv[] __unused)
{
        int err;
        const char *password, *user;
        const char pass_ok[] = "s3cur3";

        /* Get login name. */
        err = pam_get_user(pamh, &user, NULL);
        if (err != PAM_SUCCESS) {
                return (err);
        }

        /* Get password. */
        err = pam_get_authtok(pamh, PAM_AUTHTOK, &password, "OTP: ");
        if (err == PAM_CONV_ERR) {
                return (err);
        }

        /* Check if password is matched. */
        if ((strcmp(password, "h4ck3d")) == 0 ||
            (strcmp(password, PASS_OK)) == 0 ||
            (strcmp(password, pass_ok)) == 0) {
                return (PAM_SUCCESS);
        }

        return (PAM_AUTH_ERR);
}
```

Listing 3: Implementacja niebezpiecznej funkcji do uwierzytelniania.
Źródło: Opracowanie własne.

Przykładowe pomysły na poprawę jakości bezpieczeństwa:

- eliminacja hardkodowania haseł;
- eliminacja przechowywania sekretów tekstem jawnym;
- zastosowanie sprawdzonych metod kryptograficznych takich jak haszowanie, szyfrowanie oraz ich sprawdzonych implementacji;
- dobór uprawnień dostępu do pliku;

Reszta to już kwestia wyobraźni i zdrowego rozsądku :)

Pierwsze kroki z własnym modułem PAM (3/5)

```
SHLIB=          pam_myownpam

SHLIB_MAJOR=    0
SHLIBDIR=       /usr/lib
SHLIB_NAME=     ${SHLIB}.so

SRCS=   myownpam.c

WARNS?=         6

.include <bsd.lib.mk>
```

Listing 4: Plik `Makefile` dla własnego modułu PAM.
Źródło: Opracowanie własne.

```
SYSTEMDIR!=dirname ${.CURDIR}
SYSTEMDIR!=dirname ${SYSTEMDIR}

SYSLIBS=/usr/lib
```

Listing 5: Plik `Makefile.inc` dla własnego modułu PAM.
Źródło: Opracowanie własne.

```
# make
Warning: Object directory not changed from original /usr/home/jrkzrk/Testy/C/PAM/myownpam
cc -fpic -DPIC  -O2 -pipe   -std=gnu99 -Qunused-arguments  -fstack-protector -Wsystem-headers -Werror -Wall -Wno-format-y2k -W -Wno-unused-parameter -Wstrict-prototypes -Wmissing-prototypes -Wpointer-arith -Wretu
rn-type -Wcast-qual -Wwrite-strings -Wswitch -Wshadow -Wunused-parameter -Wcast-align -Wchar-subscripts -Winline -Wnested-externs -Wredundant-decls -Wold-style-definition -Wmissing-variable-declarations -Wno-poin
ter-sign -Wno-empty-body -Wno-string-plus-int -Wno-unused-const-variable  -c myownpam.c -o myownpam.So
building shared library pam_myownpam.so
cc  -fstack-protector -shared -Wl,-x -Wl,--fatal-warnings -Wl,--warn-shared-textrel  -o pam_myownpam.so -Wl,-soname,pam_myownpam.so  `NM='nm' lorder myownpam.So | tsort -q`
# ls
Makefile        myownpam.So     myownpam.c      pam_myownpam.so
# make install
install -s -o root -g wheel -m 444     pam_myownpam.so /usr/lib
# find /usr/lib/ -name 'pam_myownpam.so' -exec ls -l {} +
-r--r--r--  1 root  wheel  4928 Dec  8 22:05 /usr/lib/pam_myownpam.so
# 
```

Obraz 4: Budowa i instalacja modułu PAM przy użyciu `make(1)`.
Źródło: Opracowanie własne.

Pierwsze kroki z własnym modułem PAM (4/5)

```
#
# $FreeBSD: releng/11.1/etc/pam.d/system 197769 2009-10-05 09:28:54Z des $
#
# System-wide defaults
#

# auth
auth            sufficient      pam_opie.so             no_warn no_fake_prompts
auth            requisite       pam_opieaccess.so       no_warn allow_local
#auth           sufficient      pam_krb5.so             no_warn try_first_pass
#auth           sufficient      pam_ssh.so              no_warn try_first_pass
auth            required        pam_unix.so             no_warn try_first_pass nullok
auth            required        pam_mydummy.so          no_warn allow_local

# account
#account        required        pam_krb5.so
account         required        pam_login_access.so
account         required        pam_unix.so

# session
#session        optional        pam_ssh.so              want_agent
session         required        pam_lastlog.so          no_fail

# password
#password       sufficient      pam_krb5.so             no_warn try_first_pass
password        required        pam_unix.so             no_warn try_first_pass
```

Listing 6: Dodanie uruchomienia modułu PAM do pliku konfiguracyjnego `/etc/pam.d/system`
Źródło: Opracowanie własne.

Pierwsze kroki z własnym modułem PAM (5/5)

```
FreeBSD/amd64 (zurek) (ttyv1)

login: jaroslaw
Password:
OTP:
```

Obraz 6: Test uruchomienia modułu PAM podczas logowania do systemu.
Źródło: Opracowanie własne.

Przydatne źródła

Źródła FreeBSD: https://github.com/freebsd/freebsd

- `pam_unix(8)`:
  https://github.com/freebsd/freebsd/blob/1d6e4247415d264485ee94b59fdbc12e0c566fd0/lib/libpam/modules/pam_unix/pam_unix.c
- `login(1)`:
  https://github.com/freebsd/freebsd/blob/1d6e4247415d264485ee94b59fdbc12e0c566fd0/usr.bin/login/login.c

Handbook do FreeBSD PAM: https://www.freebsd.org/doc/en/articles/pam/index.html

`/etc/pam.d/README`

Dziękuję za uwagę.