Authentication Lab

PAM: Pluggable Authentication Modules (Linux)

Question 1 (30): PAM Configuration I. Here is a sample configuration file in `system-auth`, explain the meaning of each configuration line.

```
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        required      /lib/security/$ISA/pam_env.so
auth        sufficient    /lib/security/$ISA/pam_unix.so likeauth nullok
auth        required      /lib/security/$ISA/pam_deny.so

account     required      /lib/security/$ISA/pam_unix.so
account     sufficient    /lib/security/$ISA/pam_succeed_if.so uid < 100 quiet
account     required      /lib/security/$ISA/pam_permit.so

password    required      /lib/security/$ISA/pam_cracklib.so retry=3 minlen = 8
password    sufficient    /lib/security/$ISA/pam_unix.so nullok use_authtok md5
shadow
password    required      /lib/security/$ISA/pam_deny.so

session     required      /lib/security/$ISA/pam_limits.so
session     required      /lib/security/$ISA/pam_unix.so
```

Question 2 (30): **PAM Configuration II**. What is the functionality of the `pam_unix` module? Configure PAM such that one needs to type password when substituting user (i.e., running the `su` command) in all cases, even if `su` is run by the root (usually, when root runs `su`, it does not need to type passwords).

Question 3 (40): PAM-Aware Programs. In order to use PAM, we need to make our implementation PAM-aware. Here is a `su` command that uses PAM.

a) Describe the flow of the main function related to PAM. You may want to give a flow chart.
b) Explain the PAM-related function calls in the program.

Note that it is not necessary to compile and run it; there are already some comments in the program to help you understand the file; more detailed comments you give, more credits you will get.

```
/*su.c*/
#include <err.h>
#include <pwd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <unistd.h>

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

static struct pam_conv pamc = {
   misc_conv,
   NULL
};

/*#include <security/openpam.h> */  /* for openpam_ttyconv() */
extern char **environ;
static pam_handle_t *pamh;

static void
usage(void)
{

   fprintf(stderr, "Usage: su [login [args]]\n");
   exit(1);
}

int main(int argc, char *argv[])
{
   char hostname[MAXHOSTNAMELEN];
   const char *user, *tty;
   char **args, **pam_envlist, **pam_env;
   struct passwd *pwd;
   int o, pam_err, status;
   pid_t pid;

   while ((o = getopt(argc, argv, "h")) != -1)
      switch (o) {
      case 'h':
      default:
         usage();
      }

   argc -= optind;
   argv += optind;

   if (argc > 0) {
      user = *argv;
      --argc;
```

```
    ++argv;
} else {
    user = "root";
}

/* initialize PAM */
pam_start("su", user, &pamc, &pamh);

/* set some items */
gethostname(hostname, sizeof(hostname));
if ((pam_err = pam_set_item(pamh, PAM_RHOST, hostname)) != PAM_SUCCESS)
    goto pamerr;
user = getlogin();
if ((pam_err = pam_set_item(pamh, PAM_RUSER, user)) != PAM_SUCCESS)
    goto pamerr;
tty = ttyname(STDERR_FILENO);
if ((pam_err = pam_set_item(pamh, PAM_TTY, tty)) != PAM_SUCCESS)
    goto pamerr;

if ((pam_err = pam_authenticate(pamh, 0)) != PAM_SUCCESS)
    goto pamerr;
if ((pam_err = pam_acct_mgmt(pamh, 0)) == PAM_NEW_AUTHTOK_REQD)
    pam_err = pam_chauthtok(pamh, PAM_CHANGE_EXPIRED_AUTHTOK);
if (pam_err != PAM_SUCCESS)
    goto pamerr;

/* establish the requested credentials */
if ((pam_err = pam_setcred(pamh, PAM_ESTABLISH_CRED)) != PAM_SUCCESS)
    goto pamerr;

/* authentication succeeded; open a session */
if ((pam_err = pam_open_session(pamh, 0)) != PAM_SUCCESS)
    goto pamerr;

/* get mapped user name; PAM may have changed it */
pam_err = pam_get_item(pamh, PAM_USER, (const void **)&user);
if (pam_err != PAM_SUCCESS || (pwd = getpwnam(user)) == NULL)
    goto pamerr;

/* export PAM environment */
if ((pam_envlist = pam_getenvlist(pamh)) != NULL) {
    for (pam_env = pam_envlist; *pam_env != NULL; ++pam_env) {
        putenv(*pam_env);
        free(*pam_env);
    }
    free(pam_envlist);
}

/* build argument list */
if ((args = calloc(argc + 2, sizeof *args)) == NULL) {
    warn("calloc()");
```

```
        goto err;
    }
    *args = pwd->pw_shell;
    memcpy(args + 1, argv, argc * sizeof *args);

    /* fork and exec */
    switch ((pid = fork())) {
    case -1:
        warn("fork()");
        goto err;
    case 0:
        /* child: give up privs and start a shell */

        /* set uid and groups */
        if (initgroups(pwd->pw_name, pwd->pw_gid) == -1) {
            warn("initgroups()");
            _exit(1);
        }
        if (setgid(pwd->pw_gid) == -1) {
            warn("setgid()");
            _exit(1);
        }
        if (setuid(pwd->pw_uid) == -1) {
            warn("setuid()");
            _exit(1);
        }
        execve(*args, args, environ);
        warn("execve()");
        _exit(1);
    default:
        /* parent: wait for child to exit */
        waitpid(pid, &status, 0);
        pam_err = pam_close_session(pamh, 0);
        pam_end(pamh, pam_err);

        exit(WEXITSTATUS(status));
    }
pamerr:   fprintf(stderr, "Sorry\n");
err:
    pam_end(pamh, pam_err);
    exit(1);
}
```