Lab 1, Report - Computer Security

Mikael Gordani
Adele Siitova
Group 13

January 2019

1 Introduction

In this assigment, I've been assigned the task of creating a mimic of the UNIX login in C, and answer questions regarding user authentication.

2 Answers to Questions

1

Password ageing is a method used by system administrators to increase security for users and their computers within an organization. Passwords have (typically) a lifespan of 90 days, before they need to change it to a new password. On linux the command *chage [options] LOGIN* is a method of setting a lifespan of a password for a user.

2

The advantage of this method is that it will be hard for an Adversary to gain information about the password, since once a user has used the password, it will not be used again. A disadvantage of this might lie at the user, assuming that each password generated is distinct, they are more likely to write it down, since it will be hard to remember.

3

a) There exists several different approaches:
A biomteric approach, where the authentication is done by something the user is/has e.g Fingerprints or Facial recognition.
A token-based approach, where the user has a smart-card/memory card/device, which it uses to to authenticate itself.
A password-based approach, where there exists a database with hashed passwords and a login-form of some kind, where the users type in their password.

b) If a **biometric approach** is chosen, advantages of this is that a users retina/face structure/fingerprints are distinct (in theory), so it will be hard for someone else to copy. Disadvantage of this approach might be that it might not be the most accurate e.g if there is dust on the scanner or the camera is "filthy" when trying to capture and analyse an users face. This approach is will not either be relibile if a user loses it's eye, or gets in a accident that will reconstructrure its face etc.
For a **token-based approach**, since you've an external device and a password (or PIN) for authentication, an additonal level of security is added. Also with this external device, an adversary can't eavesdrop. A disadvantage of this approach is first a device for all users, which might be expensive. The device will also have to be physically transported to the user, thus there exists a risk that the device might be stolen in the delivery process.
For the **password-approach**, it's the simplest and easiest of all the three listed approaches, the user can invidually choose it's password and does not need to be transported physically to the user (like the token-based approach). Disadvantage of this is that an user are most likely to write it down, which makes it possible for an adversary to get a hold of it. Encryption is also needed for storing the password, since an Adversary might hack the database and get a hold of the passwords.
c) For the **biometric approach**, some sort of backup mechanism should be implementet, a password or a PIN if the biometric way fails.
For the **token-based approach**, to avoid the possibility of theft, instead of transporting the device to the user, make so the user picks up the device.
For the **password-approach**, things that could be done is to educate the users of what a "good" password is. E.g implementing a policy where the password has to have a minimum character length, consists of upper-case letters and atleast a number. Having the password "salted" before hashed is also an approach to avoid dictonary atacks.

4

a) At a university a simple password is sufficient for users to access their accounts, possibly a keycard to enter university facilities.
b) A military facility contains sensitive information, which means that a biometric authentication method would be most secure.
c) For a company a token based authentication method with a pin suffices to enter the company facilities.
d) Personal computers usually do not contain sensitive information, which means that a password is enough.

5

This two-way authentication, which is also knows as Mutual Authentication is were the user trusts the servers certificate and the server trusts the users certificate. It nessecary to avoid certain attacks, e.g Man-in-the-middle attacks.

2.0.1 6

a)
i) Since csec028 executes the program, RUID will be equal to 20716. Which will also be the value of EUID.
ii) After execution SETUID(csec028), the function will check if the EUID matches the caller, which it does, so EUID and RUID will both be 20716.
b)

current user	setuid(UID)	success/failure	user EUID after setuid()
1. root	UID: 0 (root)	0 (success)	0
2. root	UID: 20757 (csec069)	0 (success)	20757
3. root	UID: 20716 (csec028)	0 (success)	20716
4. csec028	UID: 0 (root)	-1 (failure)	20716
5. csec028	UID: 0 20757 (csec069)	-1 (failure)	20716
6. csec028	UID: 20716 (csec028)	0 (success)	20716

Since root has always full access (and owns the file), 1-3 will succeed. For 4-5 it will fail since UID does not match EUID. 6 will succeed, since it's matches the EUID.

2.0.2 7

a) The RUID will be 20716, since its the owner of the process and the EUID will be 0 (root).
b) Say for some programs e.g passwd, will need to access multiple files (e.g /etc/passwd, /etc/shadow). If a user given full permission of passwd (and not SUID), it will also have to get permission for the other files, otherwise it will get permission denied (since it is not root). Having SUID, will let the user get full owner permissions, which is more convinient.

2.0.3 8

a) i) EUID will be root, and the RUID will be 20757.
ii) EUID will be 20757 and RUID will be 20757.
b) To down-grade a users privilliges during execution.

3 Conclusion

Only using password authentication is generally not secure. Combining password authentication together with other authentication methods such as tokens or biometrics adds another security level to a system. However, combining advanced techniques generally means increasing the costs in implementation and maintenance of the system.
Additionally, no matter how secure an authentication mechanism is there will always be flaws to it in form of human factors.

4 Appendix

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <stdio_ext.h>
#include <string.h>
#include <signal.h>
#include <pwd.h>
#include <sys/types.h>
#include <crypt.h>
#include "pwent.h"
#include "time.h"

#define TRUE 1
#define FALSE 0
#define LENGTH 16

void sighandler() {
  printf("Caught signal! \n");
  /* we need to catch SIGINT(2), SIGQUIT(3), SIGTSTP(20) */
}

/* generating a random number within the range of the argument, limit */
int rand_lim(int limit) {
    int divisor = RAND_MAX/(limit+1);
    int return_value;

    do {
        retval = rand() / divisor;
    } while (return_value > limit);

    return return_value;
}

/* authentication for to many failed login attempts */
void random_captcha(char input[]) {
  char randomletters[53] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
  int random_int;
  for(int i = 0; i < 6; i++) {
    random_int = rand_lim(53);
    input[i] = randomletters[random_int];
  }
}

int main(int argc, char *argv[]) {
  mypwent *passwddata;

  char important[LENGTH] = "***IMPORTANT***";
  char user[LENGTH];
  char *arguments[] = {"/bin/sh", NULL};
  char prompt[] = "password: ";
  char *user_pass;
  char *hashed_pass;

  signal(2, sighandler);
  signal(3, sighandler);
  signal(20, sighandler);

  while (TRUE) {
    srand ( time(NULL) ); /* give our random generator a seed to avoid deterministic captcha */
    /* check what important variable contains - do not remove, part of buffer overflow test */
    printf("Value of variable 'important' before input of login name: %s\n",
        important);

    printf("login: ");
    fflush(NULL); /* Flush all  output buffers */
    __fpurge(stdin); /* Purge any data in stdin buffer */
    if (fgets(user,sizeof(user),stdin) == NULL) /* fgets() to avoid overflow attacks */
      exit(0);

    /* removing the '\n' character from the input when using fgets() */
    size_t ln = strlen(user) -1;
    if(*user && user[ln] == '\n')
      user[ln] = '\0';

    /* check to see if important variable is intact after input of login name - do not remove */
    printf("Value of variable 'important' after input of login name: %*.*s\n",
        LENGTH - 1, LENGTH - 1, important);

    user_pass = getpass(prompt);
    passwddata = mygetpwnam(user);

    if (passwddata != NULL) {

      /* hashes the password from input using crypt() */
      hashed_pass = crypt(user_pass, passwddata->passwd_salt);

      if (!strcmp(hashed_pass, passwddata->passwd)) {

        printf(" You're in !\n");
        printf("Number of attempts: %d\n", passwddata->pwfailed);

        /* upon success, set the number of failed attempts to 0 and increase the pw-age */
        passwddata->pwfailed = 0;
        passwddata->pwage++;

        /* update the credentials */
        mysetpwent(user,passwddata);

        /* reminding the user to perhaps change their password when age>10 */
        if(passwddata->pwage > 10) {
          printf("Age of your password is >10! \n");
        }

        /* checking priviliges
         * if it fails, error is printed
         */

        if((setuid(passwddata->uid)) == -1)
          printf("ERROR, not enough privileged");

        /* executing /bin/sh
         * if it fails, error is printed
         */

        if((execve("/bin/sh",arguments,NULL)) == -1)
          printf("ERROR on executing");

      } else {

      /* if its not a match, the number of failed attempts will increase, and it will be printed */
      passwddata->pwfailed++;
      printf("Attempts %d\n",passwddata->pwfailed);

      /* update the credentials since an attempt has happend */
      mysetpwent(user,passwddata);

      /* we allow up to 3 continious tries until we let the user type in a captcha to prove that it is not a robot */
      if(passwddata->pwfailed > 3) {
        char input[LENGTH];
        char captcha[LENGTH];

        random_captcha(captcha);

        printf("Prove that you are not a robot, please type in the captcha..%s\n", captcha);

        /* user has to input captcha */
        fflush(NULL);
        __fpurge(stdin);
        if(fgets(input,LENGTH,stdin) != NULL) {

        /* removing \n */
        size_t s = strlen(input) -1;
```

```c
        if(*input && input[s] == '\n')
        input[s] = '\0';

        /* comparing input and captcha */
        if(strcmp(input,captcha) != 0) {
          printf("Wrong captcha!\n" );
          break;
            }
        printf("Correct captcha, you may now try again..\n");
          }
        }
      }
    }
    printf("Login Incorrect \n");
  }
  return 0;
}
```