Parallel and Distributed Computing (CSE4001) – PROJECT COMPONENT

School of Computer Science and Engineering

VIT® UNIVERSITY (Estd. u/s 3 of UGC Act 1956) VELLORE ▪ CHENNAI www.vit.ac.in

PROF. MANOOV

TEAM MEMBERS:-

YASHWARIYA SINHA	**(15BCE0178)**
SHUBHANKER AGARWAL	**(15BCE0186)**

TITLE: SHA 512 PASSWORD CRACKER

Abstract:

As we all know that parallel processing is one of biggest research areas help in increasing efficiency and reduce maintainance.so in our project we are going to implement a parallel password cracker using SHA512 print of the user.SHA512 is a hash algorithm(a cryptographic technique) that produces a “fingerprint” or a “message digest” of a file or an action. Like human fingerprints, the resulting character string is meant to be unique and only that file can produce that fingerprint.

In Linux when we save a password it is saved with a unique user fingerprint encrypted by SHA512. By using this unique identity our algorithm tries to decrypt its password by matching it with a preinstalled kali dictionary file.

The algorithm works on multithreading, so multiple threads simultaneously compares the password and when one encounters it the result is displayed. The system can be used for hacking purposes and password recovery.

Motivation:

Many of us use Linux systems daily. Are they really secure. Let's suppose a Linux or Unix system has been setup which does not have a high level of physical security. Someone can boot this system using their own CDROM, and if the hard disk is not encrypted, using a password to be entered at boot time, an attacker can mount the hard disk using another operating system and obtain a copy of your password containing files.

So to increase the security let us first discuss how can one crack the system and barge in.

<u>Theory:</u>

SHA: The Secure Hash Algorithms are a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS)

It includes- SHA0 , SHA1 , SHA2 , SHA3 families. SHA 512 is second category cryptography performing hashing in 64 bit word format.

Linux Passwords and shadow file :

Traditional Unix systems keep user account information, including one-way encrypted passwords, in a text file called ``/etc/passwd". As this file is used by many tools (such as ``ls") to display file ownerships, etc. by matching user id #'s with the user's names, the file needs to be world-readable. Consequentally, this can be somewhat of a security risk.

Another method of storing account information, one that I always use, is with the shadow password format. As with the traditional method, this method stores account information in the /etc/passwd file in a compatible format. However, the password is stored as a single "x" character (ie. not actually stored in this file). A second file, called ``/etc/shadow", contains encrypted password as well as other information such as account or password expiration values, etc. The /etc/shadow file is readable only by the root account and is therefore less of a security risk.

While some other Linux distributions forces you to install the Shadow Password Suite in order to use the shadow format, Red Hat makes it simple. To switch between the two formats, type (as root):

“/etc/passwd”

With shadow passwords, the ``/etc/passwd" file contains account information, and looks like this:

```
smithj:x:561:561:Joe Smith:/home/smithj:/bin/bash
```

Each field in a password entry is separated with ":" colon characters, and are as follows:

- Username, up to 8 characters. Case-sensitive, usually all lowercase
- An "x" in the password field. Passwords are stored in the ``/etc/shadow" file.
- Numeric user id. This is assigned by the ``adduser" script. Unix uses this field, plus the following group field, to identify which files belong to the user.
- Numeric group id. Red Hat uses group id's in a fairly unique manner for enhanced file security. Usually the group id will match the user id.
- Full name of user. I'm not sure what the maximum length for this field is, but try to keep it reasonable (under 30 characters).
- User's home directory. Usually /home/username (eg. /home/smithj). All user's personal files, web pages, mail forwarding, etc. will be stored here.
- User's "shell account". Often set to ``/bin/bash" to provide access to the bash shell (my personal favorite shell).

“etc/shadow”

The ``/etc/shadow" file contains password and account expiration information for users, and looks like this:

```
smithj:Ep6mckrOLChF.:10063:0:99999:7:::
```

As with the password file, each field in the shadow file is also separated with ":" colon characters, and are as follows:

- Username, up to 8 characters. Case-sensitive, usually all lowercase. A direct match to the username in the /etc/password file.
- Password, 13 character encrypted. A blank entry (eg. ::) indicates a password is not required to log in (usually a bad idea), and a ``*" entry (eg. :*:) indicates the account has been disabled.
- The number of days (since January 1, 1970) since the password was last changed.
- The number of days before password may be changed (0 indicates it may be changed at any time)
- The number of days after which password must be changed (99999 indicates user can keep his or her password unchanged for many, many years)
- The number of days to warn user of an expiring password (7 for a full week)
- The number of days after password expires that account is disabled
- The number of days since January 1, 1970 that an account has been disabled
- A reserved field for possible future use

So we are clear about the shadow file that it contains hashed passwords. An example for the same is given below.

**root:$6$Ig0456UR$b9rkoxIDTAue2XocjrtHjOEJxHKTClvjOxJx2agKSdUN
b0GVuC2yavlsU42TUUKSP2TDRXFPo6Ok70nkJMYiu.:14825:0:99999:7:::**

In the above line there are fields separated by ":", those are:
-The username: *root*.
-In the next field, we have 3 parts separated by "$":

- *6* means the digest function used is SHA-512.

- *Ig0456UR* Is the salt used in conjunction with the plaintext password to obtain the hash value.

- *b9rkoxIDTAue2XocjrtHjOEJxHKTClvjOxJx2agKSdUNb0GVuC2yavlsU42TUUKSP2TDRXFPo60k70nkJMYiu.* is the 512 bit hashed password obtained with the salt and the plaintext password.

Parallel SHA512 cracker with openmp

The Code

```
#include <stdio.h>

#include <string.h>

#include <crypt.h>

#include <stdlib.h>

#include <omp.h>



void chop(char *word){

  int lenword=strlen(word);

  if(word[lenword-1] == '\n')   word[lenword-1] = '\0';

}



int numlines(FILE *file){

        if(file==NULL)  return -1;

        char ch;

        int lines = 0;

        while (ch != EOF){

                ch = fgetc(file);
                if(ch == '\n')  lines++;

        }

        return lines;

}

void main(int argc, char *argv[]){

  //Parameters error handling

  if(argc!=4){

        printf("Parallel SHA-512 Password Cracker\nUSAGE: ./parshacrk <PATH TO
DICTIONARY> '$6$<SALT>$' '$6$<SALT>$<SHA-512 HASH>'\n");

        exit(-1);

  }

  int found=0;  //0 = password not found ; 1 = password found

  char word[BUFSIZ],salt[BUFSIZ], pwhash[BUFSIZ];

  FILE *words = fopen(argv[1],"r");     //Open dictionary file

  //File error handling

  if(words==NULL){

        printf("Cannot open dictionary file.\n");

        exit(-1);

  }

  strcpy(salt,argv[2]);
  strcpy(pwhash,argv[3]);

  int size = numlines(words); // Number of words in dictionary

  if(fseek(words,0,SEEK_SET)==-1)       exit(-1);       //seek error handling

  //Parallel Region

  #pragma omp parallel for private(word) shared(found) schedule(dynamic)

  for(int i=0;i<size;i++){

        if(fgets(word,BUFSIZ,words) != NULL){

                chop(word);

                if(found==1)    exit(1);

                printf("[*] THREAD %i TRYING: %s\n",omp_get_thread_num(),word);

                char *hash = (char*)crypt(word,salt);

                if(strcmp(hash,pwhash) == 0){

                        printf("[+] PASSWORD FOUND: %s\n",word);

                        found=1;

                }       }  }

  fclose(words);

  if(found==0)  printf("[-] PASSWORD NOT FOUND, EXITING...\n");

  exit(0);

}
```

Basic Explanation

1. First we include the OpenMP header.

2. Convert while loop to a for loop in order to use the constructor "parallel for" (optimized for for loops).

3. Variable word is private for each thread, because each thread will attempt to crack a different password (*private(word)*).

4. Variable found is shared by all threads, because if one of those threads find the password, the program is finished (*shared(found)*).

Conclusion:

So finally we have learnt how to build a kali linux password cracker but what we have gained is the basic knowledge of password cryptography in linux and how can one exploit this against us.

In future our team would like to work in cyber security field to enhance the encryption techniques and prevent such kind of lame attacks hence making the world a better place.