ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΥΣ13 ΕΑΡΙΝΟ 2014

Project #3

Rainbow-Tables

(6 ημέρες καθυστέρησης)

ΛΟΥΓΙΑΚΗΣ ΧΡΗΣΤΟΣ - 1115200600289

ΑΝΑΦΟΡΑ

Για το πρώτο κομμάτι της άσκησης, την υποκλοπή δηλαδή μηνυμάτων του dbus, χρησιμοποίησα την εντολή dbus-monitor. Στην αρχή την εκτέλεσα με ανακατεύθυνση εξόδου ως εξής:

$dbus-monitor > dbus-results.txt

τις ώρες που λειτουργούσε το service του project, για να δω την μορφή και τα περιεχόμενα των μηνυμάτων. Το αρχείο dbus-results.txt που δημιουργήθηκε το χρησιμοποίησα αργότερα για testing του rainbow table που έφτιαξα.

Για το δεύτερο κομμάτι της άσκησης, την δημιουργία του rainbow table, αρχικά ξεκίνησα να γράφω σε python. Αφού υλοποίησα τον αλγόριθμο για την δημιουργία του πίνακα, εκτελώντας το παρατήρησα ότι ήταν ιδιαίτερα αργό, χωρίς να έχω γράψει στον κώδικα κάτι που να το καθυστερεί τόσο. Έγραψα λοιπόν ένα μικρό κομμάτι κώδικα που έκανα απλά 1000 επαναλήψεις υπολογίζοντας σε κάθε επανάληψη ένα hash χρησιμοποιώντας την blake. Πριν και μετά την επανάληψη κράτησα σε δύο μεταβλητές το χρόνο και στο τέλος έκανα την αφαίρεση και το αποτέλεσμα ήταν 0,6 δευτερόλεπτα. Ο χρόνος αυτός ήταν πολύ μεγάλος για να παραχθεί ένα αποδοτικό rainbow table σε λογικό χρόνο. Οπότε πήρα την απόφαση να αλλάξω την γλώσσα υλοποίησης και να χρησιμοποιήσω C, όπου πράγματι οι διαφορές σε ταχύτητα ήταν τεράστιες.

Μια άλλη απόφαση που πήρα, η οποία με διευκόλυνε πολύ και τα αποτελέσματα (από άποψη ταχύτητας) ήταν πλήρως ικανοποιητικά, ήταν να χρησιμοποιήσω βάση δεδομένων για την αποθήκευση του πίνακα. Η βάση που χρησιμοποίησα ήταν η MySQL, και το σχήμα του πίνακα που έφτιαξα είναι το εξής:

```
+-------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key|Default| Extra |
+-------+-------------+------+-----+---------+-------+
| plain | varchar(6)  | NO   | UNI | NULL  |       |
| hash | varchar(64)| NO   | PRI | NULL |        |
+-------+-------------+------+-----+---------+---------+
```

Ο κώδικας που έγραψα για την άσκηση χωρίζεται σε δύο αρχεία. Το ένα ονομάζεται rainbowTable.c και είναι το πρόγραμμα που δημιουργεί το rainbowTable και το άλλο findPlain.c και είναι αυτό που δέχεται σαν είσοδο ένα hash και ψάχνει να βρεί τον κωδικό χρησιμοποιώντας το rainbow Table. Για το compile αυτών των δύο αρχείων εκτέλεσα τις εξής εντολές:

$gcc -o rainbowTable rainbowTable.c blake_ref.c `mysql_config --cflags --libs`

$gcc -o findPlain findPlain.c blake_ref.c `mysql_config --cflags --libs`

Η τελική μορφή του κώδικας φαίνεται παρακάτω:

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////rainbowTable.c////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <mysql/mysql.h>


#include "blake_ref.h"


#define CHAIN_NUM 10000000

#define CHAIN_LEN 2000


//the character set

static const char scope[] =


"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@";
```

```
//the functions used
void generateRandomPassword(BitSequence*);
void reduce(int, BitSequence*, BitSequence*);

int main(int argc, char* argv[])
{
    int i, j, k;
    BitSequence plain[6];
    BitSequence startingPlain[6];
    BitSequence hash[64];

    MYSQL *conn;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;

    srand((unsigned int) time(0));

    char *server = "localhost";
    char *user = "root";
    char *password = "root";
    char *database = "test";

    conn = mysql_init(NULL);
    //connect to database
    if (!mysql_real_connect(conn, server, user, password, database, 0, NULL, 0))
```

```
{
    fprintf(stderr, "%s\n", mysql_error(conn));
    return -1;
}

//queries that will be used and their length
char* insertQuery = "INSERT INTO rainbow VALUES";
int iq_len = sizeof("INSERT INTO rainbow VALUES");
char* checkPlainQuery = "SELECT * FROM rainbow WHERE plain = ";
int cpq_len = sizeof("SELECT * FROM rainbow WHERE plain = ");
char* checkHashQuery = "SELECT * FROM rainbow WHERE hash = ";
int chq_len = sizeof("SELECT * FROM rainbow WHERE hash = ");
char finalQuery[110];

// for the number of chains do the following
for(i = 0; i < CHAIN_NUM; i++)
{
    k = 0;
    //generate a random password, while it exists generate other
    do
    {
        if(k != 0)
            mysql_free_result(res);

        generateRandomPassword(plain);
```

```
    //build the query to check the plain
    strncpy(finalQuery, checkPlainQuery, cpq_len);
    strncat(finalQuery, "'", 1);
    strncat(finalQuery, plain, 6);
    strncat(finalQuery, "'", 1);
    //execute the query
    if(mysql_query(conn, finalQuery))
    {
        fprintf(stderr, "%s\n", mysql_error(conn));
        return -1;
    }
    //get the results
    res = mysql_use_result(conn);
    k++;
}while ((row = mysql_fetch_row(res)) != NULL);
//free the results
mysql_free_result(res);

//store the starting plaintext
strncpy(startingPlain, plain, 6);

//build the chain for the current row
for(j = 0; j < CHAIN_LEN; j++)
{
```

```
    Hash(256, plain, 48, hash);
    reduce(j, plain, hash);
}
Hash(256, plain, 48, hash);

//build the query to check the hash
strncpy(finalQuery, checkHashQuery, chq_len);
strncat(finalQuery, "'", 1);
char tempstr[3];
for(k = 0; k < 32; k++)
{
    sprintf(tempstr, "%02X", hash[k]);
    strncat(finalQuery, tempstr, 2);
}
strncat(finalQuery, "'", 1);

//execute the query to check if a same hash already exists
if(mysql_query(conn, finalQuery))
{
    fprintf(stderr, "%s\n", mysql_error(conn));
    return -1;
}
res = mysql_use_result(conn);

//if it doesnt already exists insert it in the table
```

```
if((row = mysql_fetch_row(res)) == NULL)
{
    mysql_free_result(res);

    //build the query for the insertion
    strncpy(finalQuery, insertQuery, iq_len);
    strncat(finalQuery, "('", 2);
    strncat(finalQuery, startingPlain, 6);
    strncat(finalQuery, "','", 3);
    char tempstr[3];
    for(k = 0; k < 32; k++)
    {
        sprintf(tempstr, "%02X", hash[k]);
        strncat(finalQuery, tempstr, 2);
    }
    strncat(finalQuery, "')", 2);

    //insert the plain and hash in the table
    if(mysql_query(conn, finalQuery))
    {
        fprintf(stderr, "%s: %s\n", startingPlain, mysql_error(conn));
        return -1;
    }
}
//else repeat the iteration
```

```
        else
        {
            mysql_free_result(res);
            i--;
        }
    }

    //close the connection to the database
    mysql_close(conn);

    return 0;
}

//function that generates a random plain text
void generateRandomPassword(BitSequence* s)
{
    int i;

    for(i = 0; i < 6; i++)
    {
        s[i] = scope[rand() % (sizeof(scope) - 1)];
    }
}

//the reduction function of the rainbow table
```

```
void reduce(int n, BitSequence* plain, BitSequence* hash)
{
    int i, j, h_sum;

    n = n % 12;
    for(i = 0; i < 6; i++)
    {
        h_sum = 0;

        //add every 5 values to produce a random number
        for(j = n * 5; j <= (n+1) * 5; j += 1)
        {
            h_sum += (hash[j] + 1) * (hash[j+1] + 1) + hash[31];
        }

        //use that number to build a new plain text
        plain[i] = scope[h_sum % (sizeof(scope) - 1)];

        n++;
        if(n == 12)
            n = 0;
    }
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////findPlain.c////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <mysql/mysql.h>


#include "blake_ref.h"


#define CHAIN_LEN 2000


//the character set

static const char scope[] =

"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@";


//the functions used

void reduce(int, BitSequence*, BitSequence*);

void* strcat_thread(void* argp);


//thread mutex declaration and initialization

pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

//the command constructor that will finally contain the command
```

```
//that will be execute to connect to the server and insert the passwords
char cmd[10000];

int main(int argc, char* argv[])
{
    int i, j, k;
    BitSequence plain[6];
    BitSequence hash[64];

    MYSQL *conn;
    MYSQL_RES *res = NULL;
    MYSQL_ROW row;

    char *server = "localhost";
    char *user = "root";
    char *password = "root";
    char *database = "test";

    conn = mysql_init(NULL);
    //connect to database
    if (!mysql_real_connect(conn, server, user, password, database, 0, NULL, 0))
    {
        fprintf(stderr, "%s\n", mysql_error(conn));
        return -1;
    }
```

```
//queries that will be used and their length
char* checkHashQuery = "SELECT * FROM rainbow WHERE hash = ";
int chq_len = sizeof("SELECT * FROM rainbow WHERE hash = ");
char finalQuery[110];

//the ssh command that will be used and its length
char* ssh_cmd = " | ssh sdi0600289@sbox.di.uoa.gr\n";
int s_len = strlen(ssh_cmd);
//initialization of the command builder
strcpy(cmd, "echo \"");

//wait for user to input the hash
char input[66] = {0};
printf("Please enter the hash value:\n");
fgets(input, 65, stdin);

//convert to upper case
for(i = 0; i < 64; i++)
    input[i] = toupper(input[i]);

//steps to execute from finish to start
for(i = 0; i <= CHAIN_LEN; i++)
{
    //convert the input in hash form
```

```
char hexbyte[3] = {0};
for(k = 0; k < 64; k += 2)
{
    // Assemble a digit pair into the hexbyte string
    hexbyte[0] = input[k];
    hexbyte[1] = input[k+1];

    // Convert the hex pair to an integer
    sscanf(hexbyte, "%X", (unsigned int*) &hash[k/2]);
}

//built the chain for the current position
for(j = CHAIN_LEN - i; j < CHAIN_LEN; j++)
{
    reduce(j, plain, hash);
    Hash(256, plain, 48, hash);
}

//build the query to check the hash
strncpy(finalQuery, checkHashQuery, chq_len);
strncat(finalQuery, "'", 1);
char tempstr[3];
for(k = 0; k < 32; k++)
{
    sprintf(tempstr, "%02X", hash[k]);
```

```
    strncat(finalQuery, tempstr, 2);
}
strncat(finalQuery, "'", 1);

//execute the query to check if a same hash already exists
if(mysql_query(conn, finalQuery))
{
    fprintf(stderr, "%s\n", mysql_error(conn));
    return -1;
}
res = mysql_use_result(conn);

//if it is in the table find the password
if((row = mysql_fetch_row(res)) != NULL)
{
    strncpy(plain, row[0], 6);

    for(j = 0; j < CHAIN_LEN - i; j++)
    {
        Hash(256, plain, 48, hash);
        reduce(j, plain, hash);
    }
    mysql_free_result(res);

    //create thread that will add the command for the current password
```

```
        //that was found to the command builder
        pthread_t tid;
        char pass[7] = {0};
        strncpy(pass, plain, 6);
        if(pthread_create(&tid, NULL, strcat_thread, pass))
        {
            printf("Error in thread creation!");
            return -1;
        }
        pthread_detach(tid);

        //continue doing the same because maybe it was a false-alarm
    }
    else
    {
        mysql_free_result(res);
    }
}

//add the last piece in the command builder and execute it
strcat(cmd, "\"");
strcat(cmd, ssh_cmd);
system(cmd);

//close the connection to the database
```

```
    mysql_close(conn);

    return 0;
}

//function that generates a random plain text
void* strcat_thread(void* argp)
{
    pthread_mutex_lock(&mtx);

    strcat(cmd, "printf '");
    strncat(cmd, argp, 6);
    strcat(cmd, "\\n' | nc localhost 4433\n");

    pthread_mutex_unlock(&mtx);

    pthread_exit(NULL);
}

//the reduction function of the rainbow table
void reduce(int n, BitSequence* plain, BitSequence* hash)
{
    int i, j, h_sum;
    n = n % 12;
    for(i = 0; i < 6; i++)
```

```
    {
        h_sum = 0;

        //add every 5 values to produce a random number
        for(j = n * 5; j <= (n+1) * 5; j += 1)
        {
            h_sum += (hash[j] + 1) * (hash[j+1] + 1) + hash[31];
        }

        //use that number to build a new plain text
        plain[i] = scope[h_sum % (sizeof(scope) - 1)];

        n++;
        if(n == 12)
            n = 0;
    }
}
```

Ακολουθεί η λογική των προγραμμάτων:

-rainbowTable.c: γίνεται αρχικά η σύνδεση στη βάση. Στην συνέχεια ξεκινάει η επαναληπτική διαδικασία για την δημιουργία του πίνακα. Το πρώτο πράγμα που γίνεται είναι να παραχθεί ένας τυχαίος κωδικός και να ελεγθεί αν υπάρχει ήδη στην βάση αλυσίδα που ξεκινάει με τον ίδιο κωδικό. Αν υπάρχει παράγει νέο μέχρι να παραχθεί κάποιος που δεν υπάρχει ήδη, για αποφευχθούν τα διπλότυπα. Αν δεν υπάρχει συνεχίζει την διαδικασία υπολογίζοντας τους κρίκους όλης της αλυσίδας. Το τελευταίο hash που θα μπει στον πίνακα ελέγχεται και αυτό αν υπάρχει ήδη. Στην περίπτωση που υπάρχει έχει συμβεί κάποιο collision και υπολογίζεται ξανά η αλυσίδα για νέο κωδικό.

Αλλιώς, εισάγεται το ζευγάρι του plain text και του hash στην βάση. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να δημιουργηθούν όλες οι αλυσίδες.

-findPlain.c: και αυτό το πρόγραμμα στην αρχή συνδέεται στην βάση. Στην συνέχεια δέχεται από τον χρήστη ένα hash και το μετατρέπει σε κεφαλαία, επειδή έτσι τα τυπώνει η συνάρτηση blake στη C. Για το hash αυτό υπολογίζει με βήμα προς τα πίσω κάθε φορά τους κρίκους της αλυσίδας, ξεκινώντας από το σημείο που βρίσκεται εκείνη τη στιγμή μέχρι το τέλος. Όταν βρεθεί αποτέλεσμα ίδιο με κάποιο που υπάρχει στη βάση τότε υπολογίζονται οι κρίκοι από την αρχή μέχρι το σημείο που έχει φτάσει ο έλεγχος για να βρεθεί ο κωδικός. Η διαδικασία συνεχίζεται μέχρι να φτιαχτεί όλη η αλυσίδα, για να καλυφτεί η περίπτωση να έχουμε false-alarm λόγω κάποιου internal collision. Για την αυτοματοποίηση της διαδικασίας εισαγωγής του κωδικού στο service του sbox, αποθηκεύεται σε ένα string κάθε φορά ο υποψήφιος κωδικός που βρίσκεται σε μια εντολή της μορφής "printf '*pass*\n' | nc localhost 4433" και χτίζεται μια εντολή όπου στο τέλος έχει την μορφή: echo "*strings*" | ssh sdi0600289@sbox.di.uoa.gr. Αφού δημιουργηθεί η αλυσίδα εκτελείται η εντολή αυτή χρησιμοποιώντας την συνάρτηση system(cmd) του linux. Για την σύνδεση στο sbox χωρίς κωδικό διαμορφώθηκαν κατάλληλα οι ρυθμίσεις της εντολής ssh στον υπολογιστή μου με χρήση των εντολών ssh-keygen και ssh-copy-id. Τα string που ανέφερα παραπάνω δημιουργούνται καλώντας τα str_thread που υπάρχουν στο αρχείο, για συντόμευση της διαδικασίας.

-R function: η reduction συνάρτηση που υπάρχει στα αρχεία, επιλέχθηκε μετά από πολλές δοκιμές με άλλες οι οποίες απορρίφθηκαν λόγω καθυστέρησης ή λόγω μικρού εύρους παραγόμενων κωδικών με αποτέλεσμα πολλά collisions. Μία από αυτές που έφτιαξα και δοκίμασα η οποία είχε καλό αποτέλεσμα αλλά αργούσε είναι η εξής:

```
void reduce(int n, BitSequence* plain, BitSequence* hash)
{
    int i, j;
    char tempstr[3];

    switch(n)
    {
```

```
case 0:
    for(i = 0; i < 6; i += 2)
    {
        sprintf(tempstr, "%02X", hash[i * 6]);
        plain[i] = tempstr[0];
        plain[i+1] = tempstr[1];
    }
    break;
case 1:
    for(i = 0, j = 0; i < 32 && j < 6; i++)
    {
        sprintf(tempstr, "%02X", hash[i]);
        if(tempstr[0] >= 'A' && tempstr[0] <= 'Z')
        {
            plain[j] = tempstr[0];
            j++;
            if(j == 6)
                break;
        }
        if(tempstr[1] >= 'A' && tempstr[1] <= 'Z')
        {
            plain[j] = tempstr[1];
            j++;
        }
    }
```

```
    for(;j < 6; j++)
    {
        plain[j] = scope[hash[j] % (sizeof(scope) - 1)];
    }
    break;
case 2:
    for(i = 0, j = 0; i < 32 && j < 6; i++)
    {
        sprintf(tempstr, "%02X", hash[i]);
        if(tempstr[0] >= '0' && tempstr[0] <= '9')
        {
            plain[j] = tempstr[0];
            j++;
            if(j == 6)
                break;
        }
        if(tempstr[1] >= '0' && tempstr[1] <= '9')
        {
            plain[j] = tempstr[1];
            j++;
        }
    }
    for(;j < 6; j++)
    {
        plain[j] = scope[hash[j] % (sizeof(scope) - 1)];
```

```
    }
    break;
case 3:
    for(i = 0; i < 6; i += 2)
    {
        sprintf(tempstr, "%02X", hash[i]);
        plain[i] = tempstr[0];
        plain[i+1] = tempstr[1];
    }
    break;
case 4:
    for(i = 0, j = 29; i < 6; i += 2, j++)
    {
        sprintf(tempstr, "%02X", hash[j]);
        plain[i] = tempstr[0];
        plain[i+1] = tempstr[1];
    }
    break;
case 5:
    for(i = 0, j = 0; i < 32 && j < 6; i++)
    {
        sprintf(tempstr, "%02X", hash[i]);
        if(tempstr[0] >= 'A' && tempstr[0] <= 'M')
        {
            plain[j] = tempstr[0];
```

```
            j++;
            if(j == 6)
                break;
        }
        if(tempstr[1] >= 'A' && tempstr[1] <= 'M')
        {
            plain[j] = tempstr[1];
            j++;
        }
    }
    for(;j < 6; j++)
    {
        plain[j] = scope[hash[j] % (sizeof(scope) - 1)];
    }
    break;
case 6:
    for(i = 0, j = 0; i < 32 && j < 6; i++)
    {
        sprintf(tempstr, "%02X", hash[i]);
        if(tempstr[0] >= 'N' && tempstr[0] <= 'Z')
        {
            plain[j] = tempstr[0];
            j++;
            if(j == 6)
                break;
```

```
        }
        if(tempstr[1] >= 'N' && tempstr[1] <= 'Z')
        {
            plain[j] = tempstr[1];
            j++;
        }
    }
    for(;j < 6; j++)
    {
        plain[j] = scope[hash[j] % (sizeof(scope) - 1)];
    }
    break;
case 7:
    for(i = 0; i < 6; i += 2)
    {
        sprintf(tempstr, "%02X", hash[i * 9]);
        plain[i] = tempstr[0];
        plain[i+1] = tempstr[1];
    }
    break;
case 8:
    for(i = 0; i < 6; i += 2)
    {
        sprintf(tempstr, "%02X", hash[i * 3]);
        plain[i] = tempstr[0];
```

```
            plain[i+1] = tempstr[1];
        }
        break;
    default:
        for(i = 0, j = 15; i < 6; i += 2, j++)
        {
            sprintf(tempstr, "%02X", hash[j]);
            plain[i] = tempstr[0];
            plain[i+1] = tempstr[1];
        }
  }
  for(i = 0; i < 6; i++)
  {
      if(hash[i] % 2)
          plain[i] = tolower(plain[i]);
  }
}
```

Η συνάρτηση που άφησα τελικά βασίζεται στο γεγονός ότι η συνάρτηση blake βγάζει άλλο αποτέλεσμα για κάθε input, οπότε χρησιμοποίησα την ιδιότητα αυτή, διατηρώντας την ταχύτητα σε πολύ καλά επίπεδα.

Μετά από την δημιουργία του rainbow table, η οποία πήρε περίπου μιάμιση μέρα έκανα δοκιμές στο service ανεπιτυχώς όμως. Το πρόγραμμα προλάβαινε να εισάγει τους κωδικούς αλλά ήταν λάθος. Μάλλον η συνάρτηση R που τελικά χρησιμοποίησα είχε περισσότερα collisions από ότι περίμενα και έπρεπε να δοκιμάσω και με κάποια άλλη καινούρια. Όμως μου τελείωσαν οι μέρες προθεσμίας και δεν πρόλαβα να δημιουργήσω εκ νέου καινούριο πίνακα.