CPS 633: Computer Security, Lab 1
Fall 2016

Due date: Thu. october 6, 2016 @ 9 AM via D2L

Background: The main purpose of this lab is for you to gain hands-on experience with password schemes, how they are used in (remote) authentication, and to become familiar with dictionary attacks on these schemes. The lab is made of 5 parts: writing a program to generate password hashes, create a dictionary of passwords and their corresponding hashes for use in a brute-force attack, write a challenge-and-response protocol for (online) authentication, and use a real-life password cracker and an associated rainbow table to crack a list of generated passwords. You are also asked to comment on a number of questions on weaknesses of the designed password scheme, and how it can be improved.

1 OLMHash Password Scheme

The basic idea for this part of the lab is to write a password scheme that mimics that of LMHash (and NTLMHash v1) used in older Windows system. You can read more about them at
`http://technet.microsoft.com/en-us/library/dd277300.aspx`, or
`http://en.wikipedia.org/wiki/NTLM`

You are to write a program in C, which we will referred to as *OLMHash*, based on the following requirements and objectives:

- The result of *OLMHash* should be an updatable table of two columns - the user id and the hash of the corresponding user's password. You may store the table in a text file.
- Upon launch, *OLMHash* should prompt for an user id.
    1. If the user id is not found in the table, the program asks the user to enter a new password. The hash of the password is then computed and added to the table alongside the user id.
    2. If the user id already exists in the table, the user is prompted for his old password. The hash of the entered password is computed and compared to the hash of the user's old password stored in

the table. If the hashes match, the user is prompted for a new password. The hash of the new password then replaces that of the old password.

3. The defend against brute-force type attacks, many password systems limit the number of 'wrong' attempts for a given user. Define a user-defined security parameter n in your code, which represents the number of unsuccessful attempts. In the execution of your program, once this limit is reached, the program should output the message "Too many unsuccessful attempts - your account is locked". However, you should ensure that the counter that you use is reset to zero, once you exit the program, i. e. if you re-run your program, up to n attempts are again allowed for any user ids.

- User id's must be at least 4 characters, but less than 32 characters long.
- Passwords must include upper and lower case letters and numbers only, i.e. no special characters.
- Passwords can contain a maximum of 12 characters. If the chosen passwords are longer than 12, any extra characters after the 12th are ignored. If the chosen passwords are less than 12 characters, they will be padded with null characters (\0) to make it *exactly* 12 characters.
- The hashing algorithm first converts all the lower case letters to their upper case equivalent. This will give us 36 possible 'true' characters to pick from.
- The 12 characters of the chosen password are then split into three 4-character blocks, $B_0B_1B_2$. We will use the ASCII representation of each character, i.e. each block B_i contains 32 bits.
- Encryption algorithms (such as DES or AES) could be used to produce the hash of a given password. BUT, here, we will instead use an encryption function, $E()$ which takes 32 bits as input and produces 32 bits as output. To produce the hash, we take each block of 4 characters B_i and compute $h_i = E(B_i)$. The hash of the password is simply the concatenation of h_1, h_2, and h_3, i.e. $h = h_1h_2h_3$. The C code for the function, $E()$, is provided to you below:

```
/********************* E function *************************/

// DES replacement cipher
// The function E takes 4 bytes from *in as input and
// writes 4 bytes to *out
void E(char *in, char *out)
{
  out[0]=(in[0]&0x80)^(((in[0]>>1)&0x7F)^((in[0])&0x7F));
  out[1]=((in[1]&0x80)^((in[0]<<7)&0x80))^(((in[1]>>1)&0x7F)^((in[1])&0x7F));
  out[2]=((in[2]&0x80)^((in[1]<<7)&0x80))^(((in[2]>>1)&0x7F)^((in[2])&0x7F));
  out[3]=((in[3]&0x80)^((in[2]<<7)&0x80))^(((in[3]>>1)&0x7F)^((in[3])&0x7F));
}
```

The encryption function, E, actually consists of a Hill cipher encryption with a 32*32 key matrix, K, with one's on the main diagonal and the super diagonal and zero's everywhere else and performs the matrix multiplication $h_i = B_i * K$.

2 Challenge-Response (Online) Authentication

Microsoft Windows implements a basic Challenge-Response protocol for remote authentication. The reason for its use is to prevent having to 'transmit' the password or even the hash of the password in the open. Otherwise, it can be reused/replayed. If a user (client) wants to be authenticated by the system (server), it has to apply a function with two inputs (for example, encryption), one being a random number sent by the server and the other being the hash of its own password. The user then forwards the result to the server, where the same computation is performed and the user is authenticated, if the results match.

You are to implement a simulation of the Challenge-Response protocol, which proceeds as follows:

1. User begins by sending his user id to the server
2. Server sends to the client a 32-bit random number, r
3. User first computes the hash of his password using the algorithm from the previous section. He then computes $f(r, h)$ where the function

$f(r, h)$ is a bit-wise XOR between the hash and the 32-bit random number, r, and sends the result back to the server

4. Server computes $f(r, h')$, where h' is the hash associated with the user id in the table and compares the result with the one sent by the user. If the results match, print to screen: "access granted". Otherwise, print: "access denied"

Note that the user id must already have a password associated with it in order for the user to be authenticated.

3 OPassword Cracker

This part of lab 1 deals with making a dictionary attack against the password scheme in section 1. This attack will focus on short passwords of length 4 and passwords that only use upper case characters. This program should create a table of hashes of all possible 4-character long passwords. When presented with a hash of a 4-character password containing upper case characters only, it should search the table, and return the actual password. To summarize,

- Create a table of hashes of all possible 4-character long passwords made of upper-case letters only.
- Given the hash of a 4-character password made of upper-case letters, search the table and return the actual password.

4 Real-Life Password Crackers

For this part, we will experiment with a real-life password cracker, *ophcrack* (`http://ophcrack.sourceforge.net`) which is available free for both Windows and Liunx machines. You can download it from source forge site. The site also has free rainbow tables for XP and Vista passwords (`http://ophcrack.sourceforge.net/tables.php`) that can be used with this password cracker. Use the online tool found at `http://www.asecuritysite.com/Encryption/lmhash` to compute a number of LMHash values. Then, use ophcrack and, if required, various rainbow tables to find the passwords used. Take screenshots of your work, and include them in your report. Begin your experiment with short simple passwords of 4 characters or less.

5 Word Problems

- Supposed 'hashed' passwords are salted using 4-bit salt values. Discuss the effect of using these salts in terms of computational and storage requirement of the password cracker described in section 3.

- Passwords in this lab are allowed to be up to 12 character long. What happens if someone picks a null password, i.e. a password containing zero characters? Most people select passwords that are 8 characters or less. What is the impact of this on the security of the system?

- Suppose the encryption algorithm, E, used to produce password hashes is weak. For example, the output of the encryption algorithm used in this lab is simply a linear combination of the input bits. How can an attacker use this knowledge to attack the system? What are some of the property of a good cryptographic hash function that can be used to address this weakness?

- To harden the security of password schemes, many systems will only allow only a limited number of unsuccessful attempts, as required in Part 1. However, the number of attempts are typically considered within a fixed period of time. What should be a typical time period? Discuss pros and cons of making this time shorter or longer.

6 Deliverables

You should submit a .zip file containing a report detailing your finding, observations, screenshots and answers to word questions in section 5. The .zip file should also contain all the programs and functions implemented in this lab, including scripts, source codes, any files created/required, such as password tables or configuration files, together with a README file with instructions on how to build/use your implementations, if it's not self-evident. Your programs should be written in C, and be well commented. Once you have submitted your lab, the TAs will schedule a demo session for your group. All members of your group have to attend this demo session, answer questions about the lab and your implementation. Your (individual) grade for this lab will be partially based on your performance in this demo. Failure to be

present for the demo or lack of knowledge about what is handed in will result in a grade of zero.