MSDS 7349
Data and Network Security
Homework Basic Security

Due Week 8

Name:

What to Submit

Your final submission shall be a single pdf document that includes this document, screen captures of your exercises plus your answers to each of the written questions (if any). Note that you are expected to clearly label each section so as to make it clear to the instructor what files and data belong to which exercise/question.

Collaboration is expected and encouraged; however, each student must hand in their own homework assignment. To the greatest extent possible, answers should not be copied but, instead, should be written in your own words. Copying answers from anywhere is plagiarism, this includes copying text directly from the textbook. Do not copy answers. Always use your own words. For each question list all persons with whom you collaborated and list all resources used in arriving at your answer. Resources include but are not limited to the textbook used for this course, papers read on the topic, and Google search results. Note that 'Google' is not a resource. Don't forget to place your name on the document.

Exercise 1 : UNIX Password Cracker

The goal of this exercise is to write a password cracker for the UNIX file system. UNIX stores all passwords in the file `/etc/passwd`. Well, it doesn't store the password itself. Instead, it stores a *signature* of the password by using the password to encrypt a block of zero bits prepended by a *salt* value with a one-way function called `crypt( )`. The result of the `crypt( )` function is stored in the `/etc/passwd` file. For example, for a password of "*egg*" and salt equal to "*HX*", the function `crypt('egg','HX')` returns `HX9LLTdc/jiDE`.

When you try to log in, the program `/bin/login` takes the password that you typed, uses `crypt( )` to encrypt a block of zero bits, and compares the result of this function with the value stored in the `/etc/passwd` file.

The security of this approach rests on both the strength of the `crypt( )` function and the difficulty in guessing a user's password. The `crypt( )` algorithm has proven to be highly resistant to attacks. Conversely, the user's choices for passwords have been found to be relatively easy to guess, with many passwords being words contained in the dictionary.

To write our UNIX password cracker, we will need to use the `crypt( )` algorithm that hashes UNIX passwords. Fortunately, the crypt library already exists in the Python 2.79 standard library (on UNIX-based operating systems). (Note: for Windows-based operating systems, you will need to find the correct way to import the UNIX `crypt()` algorithm.) To calculate the encrypted UNIX password signature, we simply call the function `crypt.cypt( )` and pass it the password and salt as parameters. This function returns the signature as a string.

A simple dictionary attack involves computing the possible signatures generated for each word in the dictionary with a range of salt values.

Let's create our first password cracker using a dictionary attack.

1) Create a file called cracker.py. Start your program by reading in the HW2-passwords.txt file and, for each password found in the file, iterate through each dictionary word found in the HW2-dictionary.txt file and appropriate salt value. Report out the password found, if any, for each user. If no password is found, indicate that no password was found.
2) Using literature review, identify from where you can retrieve the salt value used in generating the signature.

Exercise 2 : Zip File Password Cracker

The goal of this exercise is to write a zip file extractor and password cracker. For this exercise, we will use the zipfile library. You may view information about the zipfile library in Python 2.79 by issuing the command `help('zipfile')` to learn more about the library. Pay close attention to the `extractall( )` method. You may use this method to extract the contents from a zip file.

Let's begin the process of writing a zip file password cracker.

1) Write a quick script to test the use of the zipfile library. After importing the library, instantiate a new ZipFile class by specifying the filename of the password-protected zip file (*evil.zip*). utilize the `extractall( )` method and specify the optional parameter for the password (*secret*). Execute your script and turn in the code and output.
2) Use the `except Exception` exception handler to catch exceptions and print them out when an incorrect password is used. Execute your script with an incorrect password and exception handler and turn in the code and output.
3) Write a script that performs a dictionary attack on the password protected zip file. Execute your script and turn in the code and output. Be sure to provide user feedback on exceptions thrown.

Exercise 3 : Port Scanner

The goal of this exercise is to learn about port scanners for networked systems.

First, create a simple Python-based port scanner. Using the socket library, you will create a script that iterates through a range of IP addresses, and, for each IP address, will identify the active ports available for that IP address. At least ports corresponding to telnet, ftp SSH, smtp, http, imap, and https services should be scanned and identified.

Second, download and install the `nmap` port scanning software from nmap.org. Utilize `nmap` to identify the operating system and the open ports of devices on a range of IP addresses.