CS 646 – Spring 2018
Project 1 (due February 20 @ 6:00pm)

For any questions regarding the assignment, please email the instructor at rr8@njit.edu

Problem 1 (50 points): Brute force attacks against cryptographic hash functions

The objective of this problem is for students to get familiar with one-way cryptographic hash functions and their security properties. The problem also requires students to get familiar with using OpenSSL, a cryptographic library for C/C++.

Start by installing the openssl library. For this, you can take the easy approach or the hard approach:

Easy:

Install an Ubuntu 16.04.3 virtual machine and then run the following from a terminal:

```
sudo apt-get update
sudo apt-get install gcc
sudo apt-get install libssl-dev
gcc hash_example.c -lcrypto –o hash_example
```

Hard:

Install the source files for openssl, and then compile the library on your local machine. For example, under the Ubuntu Linux distribution, you can use the following steps to do this:

1) Download the latest LTS openssl library sources from:
https://www.openssl.org/source/openssl-<version>.tar.gz

2) Unarchive the tarball:
```
tar xvfz openssl-<version>.tar.gz
```

3) Run the following commands from the directory where the tarball was unarchived. You should read the INSTALL file first:

```
./config
make
make test
sudo make install
```

Task 1: One-way property of hash functions

Warm-up:

We will use the brute-force attack method to see how long it takes to break each of these properties. You are asked to write your own C programs to invoke the message digest functions in openssl's crypto library. Documentation about how to use openssl's functions to compute a hash and a sample code can be found at https://www.openssl.org/docs/man1.0.2/crypto/EVP_DigestInit.html. A file with this code (hash_example.c) has also been posted on Moodle. Please get familiar with this sample code.

You can compile hash_example.c using the following command, which will generate an executable file called hash_example:

```
gcc -I /usr/local/ssl/include/ -L /usr/local/ssl/lib/ -o hash_example
hash_example.c -lm -ldl -lcrypto
```

(you need to replace /usr/local/ssl/include/ and /usr/local/ssl/lib/ with appropriate paths if you have installed openssl in a different location than the default location)

If you followed the easy approach, you can simplify the above command:

```
gcc hash_example.c –lcrypto –o hash_example
```

And you can run the executable as in:

```
./hash_example md5
```

Your task:

A 4-byte array has been hashed using the md5 hash function into the following value (expressed in hexadecimal): acc078c68821d92947bbc24ed5c62024

You need to write a C program "search1.c" that breaks the one-way property of the hash function by executing a brute-force attack that searches over all possible strings of length 32 bits (4 bytes). Thus your program should find the 4-byte value that was hashed into the above value (each of the 4 bytes corresponds to an "unsigned char" in C). You must search through all 4-byte arrays (and not just through all strings that consist of 4 ASCII characters). ***You must zip the search1.c program with the other programs and upload the zip file to the correct place in Moodle***. Please include your names and group number (if applicable) as a comment in the beginning of the file you submit. The zip file should be named "ucid.zip" or "group #.zip". ***In your zip file you also need a text file which contains the 4-byte value found by your program (as a hexadecimal string).***

Task 2: Second pre-image resistance property

For this task, we will try to break the second pre-image resistance property of the md5 hash function. Since most of the hash functions are quite strong against the brute-force attack on this property, it will take us years to break them using the brute-force method. To make the task feasible, we reduce the length of the hash value to 32 bits. We will use the md5 hash function, but we only consider the first 32 bits (first 4 bytes) of the hash value in this task (normally, the output of the md5 hash is an array of 128 bits = 16 bytes).

Write a C program "`search2.c`" that, given the 4-byte string "`harD`" (without the quotes), finds another 4-byte array that hashes to the same value as "`harD`" does (again, consider only the first 32 bits = 4 bytes of the `md5` hash). ***You must submit the `search2.c` program as described above. In your zip file you also need a text file which contains the 4-byte value found by your program (as a hexadecimal string).***

Problem 2 (50 points): Cookie hijacking attack

For this problem you will need to use the **Wireshark** packet capture/analyzer tool.

This problem illustrates how an online account can be hijacked by intercepting authentication cookies over an insecure wireless connection. It has a slight research flavor and you have to do your own investigation in order to determine how to accomplish the required tasks.

A user has created an account with www.stumbleupon.com, which is a discovery engine that finds and recommends web content to its users. The user name for the account is **CS646SPRING2018**. When the user first logs into the account (authentication based on user name and password), the website stores several cookies in the user's browser. These cookies allow the user to skip the initial authentication step on each subsequent visit to www.stumbleupon.com. While the authentication step is made over an encrypted connection (HTTPS), subsequent visits of the user to the website are made over an unencrypted connection (HTTP) and the cookies are sent in the clear with each HTTP request.

You are provided with the file ***capture.cap*** which contains wireless traffic captured when a client connected to the website www.stumbleupon.com over an insecure wireless connection (*e.g.*, at a Starbucks hotspot); thus, it contains the authentication cookies sent in the clear by the client.

Your task is to use the information in this capture file in order to hijack this account on www.stumbleupon.com. To prove successful hijacking, you need to add your page to the list:

- from the upper right link click "My Profile" then "Add a page"
- type some URL of your choice that is not already in the list of pages, and in the "Tag" field write just your UCID (e.g. I would write rr8, NOT rr8@njit.edu)

Note: Every individual student or team member is to perform this hijacking attack. Therefore, if your group contains three members, then there should be three pages added to the CS646SPRING2018 Stumbleupon account. One for each team member.

Since all the students in the class will try to hijack the same account, **please do not interfere with the pages and tags added by other teams**! To accomplish the task, you will have to identify the packets containing the cookie(s), find the cookie(s), add the cookie(s) to your browser (with an appropriate expiration date), visit the website www.stumbleupon.com, and add a new page for this account (with an appropriate tag).

To check if you successfully added a page (and its comment), you can see all the added pages at:
http://www.stumbleupon.com/stumbler/CS646SPRING2018/comments

Specifically, for the assignment you need to:

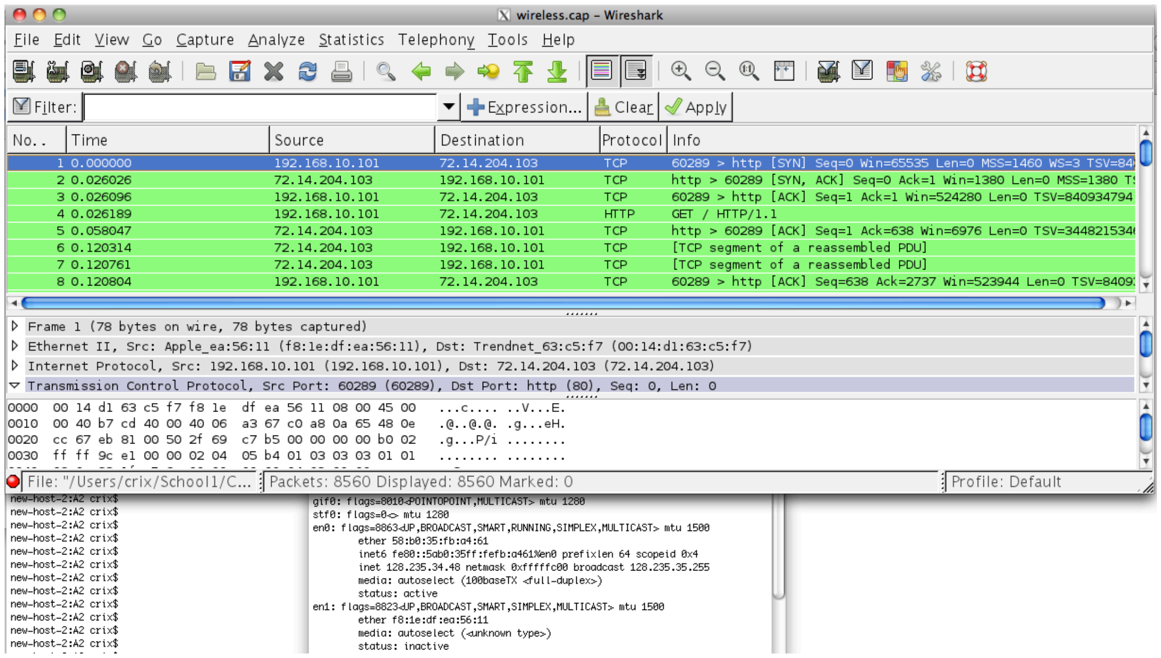
- Explain how to find the packets that contain the relevant cookies (this includes describing what display filters you used)
- Provide screenshots with Wireshark in which you show the packets that contain the cookie(s), together with the relevant http header.
- Explain how you extracted the cookie(s) from Wireshark

- Display in your submission (as text) all the **<u>relevant</u>** cookie(s) (relevant cookies are those cookies required for authentication; there may be additional cookies that are not relevant for authentication purposes)
- Explain how you inserted the cookie(s) in your browser
- Provide screenshots of www.stumbleupon.com that show your added page and your comment (which contains your name).

Try to identify the minimum number of cookies relevant for authentication.

IMPORTANT NOTE: all the captured screens are required to be taken as following:
- Open a command prompt window and type ipconfig /all (under Windows) or ifconfig (under Linux/UNIX). This will show the mac address of your machine.
- Make sure the window showing the mac address of your machine is included in all screenshots, together with the relevant information you're trying to capture in the screenshot. This has to be done **for all included screenshots**. As an example, see the figure below.

Hints:
- Use the tools in the Wireshark "Analyze" menu to help you
- Use a browser add-on for editing cookies, which will simplify cookie management.
- You can use whatever browser and add-on to manage cookies, as long as you explain what you used.


wireless.cap – Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression...
Clear
Apply

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.10.101	72.14.204.103	TCP	60289 > http [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=3 TSV=84
2	0.026026	72.14.204.103	192.168.10.101	TCP	http > 60289 [SYN, ACK] Seq=0 Ack=1 Win=1380 Len=0 MSS=1380 T
3	0.026096	192.168.10.101	72.14.204.103	TCP	60289 > http [ACK] Seq=1 Ack=1 Win=524280 Len=0 TSV=840934794
4	0.026189	192.168.10.101	72.14.204.103	HTTP	GET / HTTP/1.1
5	0.058047	72.14.204.103	192.168.10.101	TCP	http > 60289 [ACK] Seq=1 Ack=638 Win=6976 Len=0 TSV=344821534
6	0.120314	72.14.204.103	192.168.10.101	TCP	[TCP segment of a reassembled PDU]
7	0.120761	72.14.204.103	192.168.10.101	TCP	[TCP segment of a reassembled PDU]
8	0.120804	192.168.10.101	72.14.204.103	TCP	60289 > http [ACK] Seq=638 Ack=2737 Win=523944 Len=0 TSV=8409

```
▷ Frame 1 (78 bytes on wire, 78 bytes captured)
▷ Ethernet II, Src: Apple_ea:56:11 (f8:1e:df:ea:56:11), Dst: Trendnet_63:c5:f7 (00:14:d1:63:c5:f7)
▷ Internet Protocol, Src: 192.168.10.101 (192.168.10.101), Dst: 72.14.204.103 (72.14.204.103)
▽ Transmission Control Protocol, Src Port: 60289 (60289), Dst Port: http (80), Seq: 0, Len: 0
```

```
0000  00 14 d1 63 c5 f7 f8 1e  df ea 56 11 08 00 45 00   ...c.... ..V...E.
0010  00 40 b7 cd 40 00 40 06  a3 67 c0 a8 0a 65 48 0e   .@..@.@. .g...eH.
0020  cc 67 eb 81 00 50 2f 69  c7 b5 00 00 00 00 b0 02   .g...P/i ........
0030  ff ff 9c e1 00 00 02 04  05 b4 01 03 03 03 01 01   ........ ........
```

```
File: "/Users/crix/School1/C...   Packets: 8560 Displayed: 8560 Marked: 0   Profile: Default
```

```
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
new-host-2:A2 crix$
```

```
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        ether 58:b0:35:fb:a4:61
        inet6 fe80::5ab0:35ff:fefb:a461%en0 prefixlen 64 scopeid 0x4
        inet 128.235.34.48 netmask 0xfffffc00 broadcast 128.235.35.255
        media: autoselect (100baseTX <full-duplex>)
        status: active
en1: flags=8823<UP,BROADCAST,SMART,SIMPLEX,MULTICAST> mtu 1500
        ether f8:1e:df:ea:56:11
        media: autoselect (<unknown type>)
        status: inactive
```