ECE471/571 Lab Guide and Tips

Ming Li

For those of you who are not familiar with Linux/Ubuntu, please get familiar with the terminal, basic operations, and command line. Linux commands will be used quite often in the labs, and proper usage of them can make your tasks very efficient and convenient. To help you I attached some online resources:

The Linux command line for beginners (this includes basic operations, 20-30 minutes read)

https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

or: Basic Linux/Unix Commands with Examples (20-30 minutes read)

https://www.guru99.com/must-know-linux-commands.html

The Ultimate A To Z List of Linux Commands | Linux Command Line Reference (all the commands)

https://fossbytes.com/a-z-list-linux-command-line-reference/

The descriptions and usage of these Linux commands are based on their manual pages. To access the man page in the terminal:

```
man command_name
```

For example: `man cat`

In addition to the ones mentioned in the lab descriptions, I have provided a list of commonly used Linux commands or programs/tools in each lab in the following (will be updated for each lab):

Lab 1:

Task 1:

Note: No need to redo the encryption procedure, the 3-step code in the beginning is only for your reference to show you how the ciphertexts are obtained. Your task is to crack the ciphertext given in the txt file and obtain the plaintext and the key.

`cat`: Used to concatenate files and print them on the screen.

`tr`: replace certain characters in a file

`grep`: Searches input files for a given pattern and displays the relevant lines

`grep` can also be used to get frequency of letters or phrases, e.g:

```
grep -o xxx file.txt | uniq -ic
```

Tasks 2-6 (in addition to above):

`touch`: create a file.

`echo`: send input string(s) to standard output, either save to a file or display text on the screen. E.g.,
`echo -n "string" > file.txt`

`gedit`: a text editor to edit the files (can be easier to use than bless).

`xxd`: command line hex viewing tool, allows you to observe the contents of the file in hexadecimal format (as the data may not be directly printable). `xxd -p filename` allows you to view the file contents in plain hex values without the formatting.

`Xxd` can also be used to convert the ASCII representation of hex values to actual binary content, e.g.:
`xxd -r -p hexfile.txt > binaryfile.txt`

`ghex`: open a file using the hex editor to manually change bits in a file.

Online XOR calculator: www.xor.pw

Hints for task 6.3: This task asks you to determine Bob's plaintext (either Yes or No). What is known to the adversary (you) is the ciphertext for an arbitrary plaintext chosen by yourself, under an IV that is given/predictable. Based on the first ciphertext C1 and the first IV1 (both given), you need to construct another plaintext P2 whose corresponding C2 is also available to you (by querying the encryption oracle), as well as the knowledge of the next IV2 (predictable and is given by the program). With all this information (C1, P2, C2 and IVs) you need to find a way to determine what is P1. The key point here is how to construct a correct P2 from a guessed P1 (only two possibilities, "yes" or "no") and the IVs. Exploit the structure of CBC mode (specifically the XOR operation).

Note that, it is important to know what is the padding format added by the CBC algorithm in openssl, otherwise you will not obtain the correct result. The padding is NOT 00 but it is 0D repeated in hex (until reaching the block length). If you want to recover and confirm the padding you should try decryption with the -nopad option. You then take you guessed plaintext P2 and ask Bob to encrypt it with the knowledge of IV2 given in the lab, to see whether the ciphertext is the same as C1. Make sure to have -nosalt at the end of the encryption command. Make sure that the key, IVs, and the ciphertexts are all represented in hex or binary format. Plaintexts need to be converted from ASCII code into hex or binary first.

Lab 2:

In this lab you are expected to write some C code.

In Task 1, after you create and edit task1.c, in the command line, you can use gcc to compile, e.g.,
`gcc task1.c -o task1`

and then `./task1` to run.

In Task 2, you will need to brute force all the keys possible during the giving time range. To do this you can use the `date` command to get the time elapsed in seconds first, and then write a short C program to generate and store all the possible keys in a file. Then you can write a script to encrypt the known plaintext using all the possible keys and compare with the ciphertext. You can use either C or python to do this, or bash script.

Lab 3:

For the hex editor, gHex is a good alternative to Bless, if you encounter issues with Bless.

Frequently used commands: `md5collgen, md5sum, diff, more, cat`

In Task 2, to obtain the two files M and N with the same hash, you can use the same files that you have generated in Task 1. Or you may choose to generate two new files using the collision generation method in Task 1 again (with any prefix of your choice).

In Task 3, you need to write a simple C program to display contents of an array, then generate a prefix from the binary file of your C code, and then generate two binary files with the same hash using md5collgen.

In Task 4, you need to write two different C programs (each containing two strings), such that they have the same MD5 hash.

Lab 4:

Read and compile the sample C code first. Your code in each task will make use of the BIGNUM API, in a similar fashion to the sample code.

Task 1 tip: we can use `BN_mod_inverse( )` function. To initialize the input big numbers, use `BN_hex2bn()` function.

Tasks 2, 3 and 4, 5 are similar to each other.

Task 2, you need to encrypt the plaintext message, and then decrypt it to check whether it is the same.

You can use Python command to convert an ASCII string to a hex string:

```
Python –c 'print("the plaintext message in ASCII".encode("hex"))'
```

And to convert hex back to ASCII after decryption:

```
Python –c 'print("the plaintext message in hex".decode("hex"))'
```

In Task 6, you will download a certificate from a real web server first and manually verify the signature using your code in the previous task. Note in Step 4 you need to extract the body of the certificate, don't include the signature block itself.

Lab 5:

Before you do this lab, the first thing is to make sure you can setup multiple VMs on the same machine. Follow this manual to do so: https://seedsecuritylabs.org/Labs_16.04/Documents/SEEDVM_VirtualBoxManual.pdf , appendix A and B. Clone the VM to setup another VM, and make sure they are setup under the same network.

For task 1.1A, make sure the filter is set to ICMP and not TCP or IP. If you do not see anything captured, make sure you are receiving traffic by generating them.

While running the script, open another terminal and execute `ping github.com`. The ping program uses the Internet Control Message Protocol (ICMP), thus the traffic can be sniffed by Scapy with a filter specified as `icmp`. Don't forget to run the script with the root privilege, e.g., `sudo python sniffer.py`.

For task 1.1B, you will need to generate telnet traffic if you want to collect anything on port 23. Try telnet using the other VM's IP address.

You can also just use Scapy to generate random TCP packets on port 23 from whatever IP you want. Use the code in **Task 1.2: Spoofing ICMP Packet** as an example.

For task 1.2, you need to set the source address of the spoofed ping request packet to another VM's address, and destination address to any remote machine on the Internet which is alive, if you see the reply packet to that source address then it is successful. Make sure you can ping that remote machine first from your own VM first, then change the source address.

For task 2 (Task 2.1: ARP Cache Poisoning), you need to go into virtualbox and change the network setting for each VM to 'NAT Network'. Then run each VM (i.e. power them on, log in, etc.). Each should have an interface 'enp0s3' with an IP address 10.0.2.x (e.g. 10.0.2.34 in your screenshot) and should be able to connect to each other.

FAQ about the VM:

1. how do I copy the files into the VM from my host machine?

Answer: No need to do that. Just go online within your VM, go to D2L or Seed Labs website to download the required files for each task (e.g., ciphertexts). Do everything within your VM.

If you still want to copy files into your VM from your host or vice versa, you can try these methods:

3 Ways to Transfer Files between Windows and VirtualBox

https://www.isunshare.com/blog/3-ways-to-transfer-files-between-windows-and-virtualbox/

Copy and pasting between host and VM:

https://apple.stackexchange.com/questions/132233/copy-and-pasting-between-host-and-vm