CS2107 Self-Exploration Activity 1

Notes:

This self-exploration activity is shared with you for your own exploration of the practical aspects of our module. This activity is ***ungraded***, and you ***do not*** need to submit any answers. The tasks and extra material here are ***not examinable*** either.

For Activity 1, you can perform the following:

1. To see how an attacker nowadays can generate **a malicious executable** that, when executed, takes a full control of a victim's Windows machine.
2. To try out some Python scripts and Linux command that allow you to see how the **substitution cipher** can be implemented as well as attacked.

Task 1: Looking at the "Difficulty" of Creating a Malicious Executable File

On their machine, some users simply like to run an executable file coming from an untrusted source. On example is an executable attached in an email which purportedly comes from someone a user knows. Another example is an executable file downloaded from an untrusted web server.

I've uploaded to LumiNUS a demo video file named "`CS2107-Msfvenom-meterpreter-demo.mp4`", which was previously shown in our first lecture. You can observe the video to see how an attacker can use **MSFvenom** (https://www.offensive-security.com/metasploit-unleashed/msfvenom/) to easily generate a malicious executable that targets a Windows machine.

When the executable is executed, its payload *calls back/home* the attacker, so that the attacker can remotely "own" the victim's machine.

Note that the malicious executable shown in the demo is rather basic in that:

- It does not trojanize a useful executable (e.g. a game or tool): this can make the victim user suspicious;
- It does not employ any encoding technique: this can cause anti-virus software running on the target machine block the executable's execution.

Nevertheless, the demo should be able to clearly illustrate how the attacker can *easily* generate such a powerful malicious executable. In fact, advanced features of the Metasploit tool can be employed to address the two above-mentioned issues.

Task 2: Running Python Scripts that Implement and Crack Substitution Cipher

There is a nice book that shares how you can implement several ciphers and also hack these ciphers in Python: Al Sweigart, "*Cracking Codes with Python: An Introduction to Building and Breaking Ciphers*", No Starch Press, 2018. You can read the book online for free at: https://inventwithpython.com/cracking/, and you can also download the source code used in the book from: https://inventwithpython.com/CrackingCodesFiles.zip.

If you are interested, you can check the following chapters about implementing and hacking the **substitution cipher**, and run the provided Python scripts:

- Chapter 16 - **Programming** the Simple Substitution Cipher
- Chapter 17 - **Hacking** the Simple Substitution Cipher

Task 3: Linux Command and Tools for Cracking Substitution Cipher

Alternatively, if you have a **Linux machine** and can run shell commands, you can run **`tr` (translate) command** to encrypt, decrypt, and hack a substitution cipher. Its manual page can be seen at: http://linuxcommand.org/lc3_man_pages/tr1.html.

To **encrypt** a plaintext file and output the corresponding ciphertext file, you can use `tr`, for instance using a sample letter mapping below, as follows:

```
$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalh
yuojzc' < plaintext.txt > ciphertext.txt
```

To **decrypt** a ciphertext file using the same letter mapping, you can run:

```
$ tr 'sxtrwinqbedpvgkfmalhyuojzc' 'abcdefghijklmnopqrst
uvwxyz' < ciphertext.txt > plaintext_recovered.txt
```

To **hack** a substitution cipher, first, you can check out sites that help you easily derive the statistics from a ciphertext, including the single-letter frequencies, bigram (2-letter sequence) frequencies, and trigram (3-letter sequence) frequencies, such as: http://www.richkni.co.uk/php/crypta/freq.php and https://www.dcode.fr/frequency-analysis.

For information about single-letter, *bigram* and *trigram* frequencies for a typical English plaintext, you can read:

- https://en.wikipedia.org/wiki/Frequency_analysis
- https://en.wikipedia.org/wiki/Bigram
- https://en.wikipedia.org/wiki/Trigram

Subsequently, you can use `tr` to **test**, for instance, replacing letters `y`, `b`, and `s` in `ciphertext.txt` with letters `T`, `H`, `E`, respectively as follows:

```
$ tr 'ybs' 'THE' < ciphertext.txt
```

In your hacking, you can then iteratively and adaptively continue adding your "guessed pairs of mapped letters" into `tr`'s two arguments until you find the right complete letter mapping.

Good luck and have fun with your hacking!