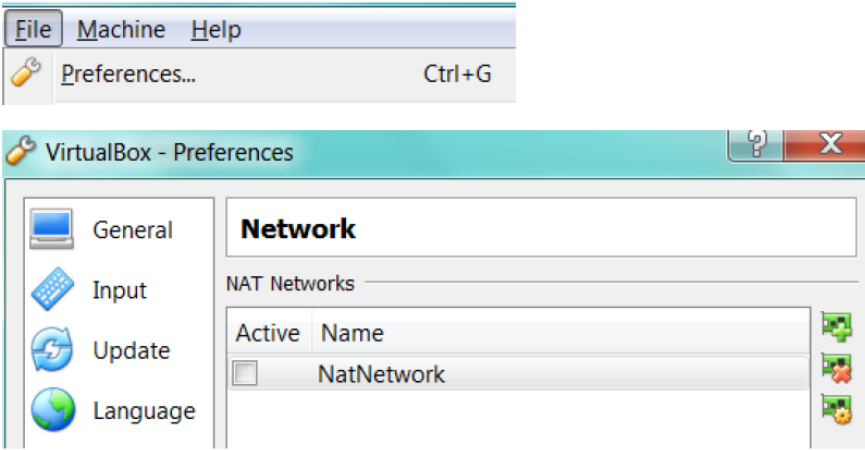

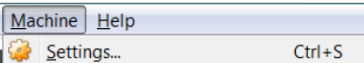
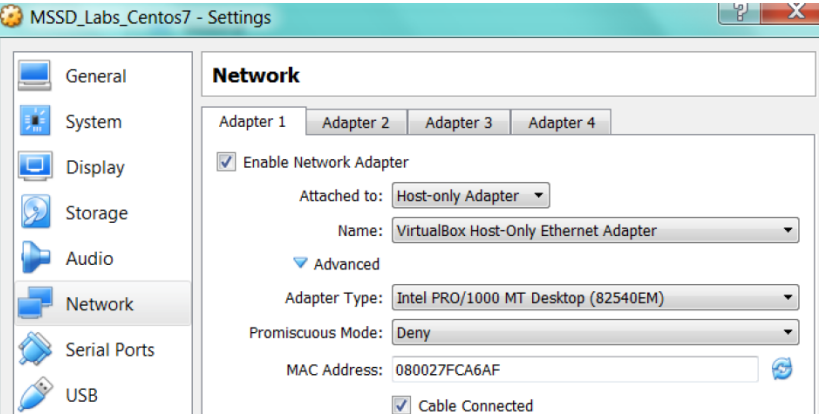


51.506 Security Tools Lab 1

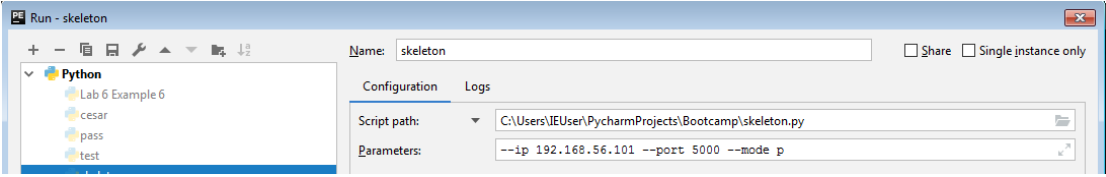
Assignment 2 – Breaking of simple ciphers.

Due Date: 6 June 2023

Please read the assignment fully before embarking on doing the assignment

1.	Objectives <ul style="list-style-type: none">• Use provided small HTTP client that will communicate with remote challenge server API• Break cipher challenges using brute force and frequency analysis• Manipulate OTP plaintext to a target string
2.	Setup <ul style="list-style-type: none">• We're using Python 3 in this exercise• Before starting the Centos VM, ensure you're not using NAT (uncheck if in use).• You will need to start the Kali VM to find the IP of the Centos VM.  <ul style="list-style-type: none">• Ensure you're using Host-only adapter   <p>(Adapter type & MAC address might be different)</p> 

3.	<p>HTTP Client</p> <ul style="list-style-type: none"> • In this exercise, a remote HTTP API will provide a set of challenges for you. You are supposed to break the encryption and recover the plaintexts. • The server is running on <a href="http://<VM_IP>:<Port>">http://<VM_IP>:<Port> • A very simple plaintext API is running on <a href="http://<VM_IP>:<Port>/challenges/plain">http://<VM_IP>:<Port>/challenges/plain • The API will always provide JSON data or a plain ASCII error/success message. • Any binary data is encoded as hex string. For example, 'x41' is encoded as '41'. 'a' is encoded as '61', and so on. • The API provides a cookie. You must send that cookie together with your solution. • Based on the provided python request skeleton code, connect to the API, download the plaintext, and resubmit it as a "solution" field to <a href="http://<VM_IP>:5000/solutions/plain">http://<VM_IP>:5000/solutions/plain. You should get a positive feedback message when solution is correct. • Consider the full range of 256 ASCII values in your encryption and decryption implementation. Your decrypted plaintext should consist only of printable characters ([a-zA-Z0-9{}!])etc).
4.	<p>Trial Run</p> <ul style="list-style-type: none"> • You don't have access to the VM – it is a black box for you. You will just query it on port 5000. • Find the IP address using NMAP tool from the Kali VM <pre># nmap -sP 192.168.56.0/24</pre> <p>One of the IPs will be for Kali (use ifconfig to find out) while the other one with '(Oracle VirtualBox virtual NIC)' label will be for the HTTP Client.</p> <ul style="list-style-type: none"> • The python file skeleton.py is provided as a guideline. In the original file there's a simple plaintext challenge. The received challenge is in HEX and you're just converting it to plain ASCII using the string method decode("hex") and sending it to /solutions/plain so as to get the response whether your solution is correct. Note down that IP. • Run your scripts by setting parameters to <pre>--port <port> --ip <ip> --mode = ['p', 'c', 's', 'o']</pre> <p>You can run it from the command prompt.</p> <ul style="list-style-type: none"> • <pre>python skeleton.py --ip 192.168.56.101 --port 5000 --mode p</pre> <p>Or if you're using Pycharm you can edit configurations by pressing Alt+Shift+F10 then pressing 0 to see the 'Configurations' menu.</p> <p>Example (for plaintext):</p>

	<pre>--port 5000 --ip 192.168.56.101 --mode p</pre>  <pre>[DEBUG] Obtained challenge ciphertext: 0x666C61677B746869735F69735F615F746573747D with len 42 [DEBUG] Submitted solution is: { "cookie": "0x57029cb50bb3da652a1c7c9d5e6a5a66", "solution": "flag{this_is_a_test}" } [DEBUG] Obtained response: Your answer is correct... of course!</pre>
5.	<p>Caesar's cipher</p> <ul style="list-style-type: none"> • Extend the python script by option -c to call /challenges/caesar. Can you guess the key by brute force attack? • Send the solution plaintext to /solutions/caesar to verify whether it is correct • A shift of the Caesar cipher is operating on the ASCII value of characters ('a' shifted by 4: $0x61 + 0x4 = 0x65$) • Hint: You need to find a good way to identify the correct key from the decrypted plaintext. For example, you could decrypt the first 50 characters using all the possible keys and visually inspect them to find the correct key. Note that each challenge call will generate a new Caesar cipher with a new shift, so keep the challenge text and cookie for reference to the same challenge.
6.	<p>Frequency Analysis of Substitution Cipher</p> <ul style="list-style-type: none"> • Extend the python script by option -s to call /challenges/substitution. Can you find the substitution key by brute force attack? How many different combinations would you have to try? • Send the solution plaintext to /solutions/substitution to verify whether it is correct • The substitution operates on all ASCII values (full permutation of all 256 values) • To simplify the attack, the plaintext is in lower case characters only. • Hint: The key will change for each /challenges/substitution API call, be sure to record the cookie along with the ciphertext before analysing. Brute force search is likely infeasible (maybe consider using a dictionary?). Try frequency analysis.
7.	<p>OTP messages Integrity</p> <ul style="list-style-type: none"> • Extend the python script by option -o to call /challenges/otp. You will obtain a ciphertext encrypted with the One Time Pad (OTP) cipher, and your task is to modify pertaining ciphertext as described below.

	<ul style="list-style-type: none">• Try simply submitting the ciphertext to the API (/solutions/otp), you should see corresponding plaintext message.• Implement a solution that manipulates the ciphertext in order to change the plaintext message returned in the response of the API call. In particular, you must change the student ID to your own student ID, and the points wished to 6.
8.	<p>Hand-in</p> <ul style="list-style-type: none">• Submit one script that supports all four API calls (parameter --mode = ['p', 'c', 's', 'o']), computes the solutions, and submits them by pertaining /solutions/ API calls in the same vein as was demonstrated by the /plain/ example. You may hard-code parameters for the cipher analysis if required.• You can make your debug prints, however remove/comment out them in the final version. The final script must display only the request submitted to the server in JSON format (containing cookie and solutions fields). For example, in a case of plain challenge it must be in format: <pre>{ "cookie": "0x57029cb50bb3da652a1c7c9d5e6a5a66", "solution": "test" }</pre>• Make sure to put your username into the header of the submitted file.• Implement mandatory parameters --port and --ip using argparse library.• Make sure to do proper input validation as it'll be checked thoroughly, for example for invalid or inaccessible IPs/Ports.