Exercise 9

Common network vulnerabilities

Log into your VM (`user` / `1234`), open a terminal and type in `infosec pull 9`.

- When prompted, enter your username and password
- Once the command completes, your exercise should be ready at `/home/user/9/`

When you finish solving the assignment, submit your exercise with `infosec push 9`.

Setup

In this exercise, you will learn about the risks imposed on everyday computer networks, due to the existing usage of insecure networking protocols. To do that, we'll set up a victim machine talking to the internet and use our existing machine to attack its communication.

Notes:

1. The setup instructions seem long (2.5 pages), but in fact they are really short. It's just super detailed, to solve most possible questions :)
2. This is a networking exercise. People send some really random stuff over the internet and network, and strange inputs may crash code that is not well-written. Double check your code - **we will not accept any appeal over lost grade due to a crash with strange packets**.
    a. If the exercise says you can assume a packet satisfies some condition, it's OK if your code ignores packets that don't match. However, your code must NOT throw an exception

Step #1 - Add another VM

For this exercise, you'll need another VM to server as the victim. To make it easier for you to run two VMs without exhausting your system resources, we created a new tiny VM - (**fast** download link, **slow** (backup) download link). It requires only 256Mb of RAM, and has only a text (console-like) interface, which should be super-light-weight.

After downloading this VM and importing it in Virtualbox (File → Import Appliance...) we'll need to set up both VMs to be on the same local network, which uses NAT to access the internet.

Step #2 - Creating a shared network[1] for the VMs

1. Open the networks configuration tab for Virtualbox - Go to File → Preferences and select the Networks section
2. Inside NAT Networks tab, the Use the '+' button to create a new network
3. Edit the settings of the network (use the button that looks like a screwdriver) to have the following settings:
    a. Enable Network: True
    b. Network Name: infosec
    c. Network CIDR: 10.0.2.0/24 (should be the default, fix it if not)
    d. Support DHCP: Enabled
    e. Support IPv6: **Disabled**
4. Close all the dialogs with OK to save our changes

Step #3 - Putting both VMs on the network

For each of the two VMs (the regular and the victim one) do the following:

1. Make sure the VM is turned off
2. Enter the VM Settings (via the gears button in the main window) and select the Network section
3. In the tab of the default network adapter (Adapter 1), change the Attached to setting to NAT Network and from the name dropdown chose our network - infosec
4. Expand the Advanced section and set Promiscuous Mode[2] to Allow VMs
5. Close all the dialogs with OK to save our changes

Step #4 - Verify we have internet access

Verify your setup by attempting to access the internet on both machines:

1. Turn on the VM and login (the new VM has the same username/password)
    a. Starting the machine may take minute longer than usual as network configurations are updated for the first time. That is OK
2. Try opening a command line and typing `nslookup www.google.com`
    a. If successful, you'll get the IP of google.com
3. If you don't have internet access, this may be a bug in your version of Virtualbox (affecting versions of Virtualbox before 5.1.4)
    a. Install a later (>= 5.1.4) version from the Virtualbox website

---

[1] We'll set up a NAT network - meaning both VMs will use the same IP Subnet, and will communicate with the "outer world" using NAT which will be performed by the Gateway (Virtualbox)
[2] Promiscuous = The VM will listen to all traffic on the network, even if not directly addressed to it. This is indeed the state in many local networks

b. If you are using Linux and need help checking for the bug and/or upgrading, see here for more details
    i. Make sure you also remove the `virtualbox-dkms` package before installing the latest Virtualbox version

Step #5 - Verify the VMs can see each others' traffic

- On the main VM (not the mini one), from within the exercise directory, run `sudo python sniffer.py --all` to run the packet sniffer we wrote for you
- Open the victim VM and run `elinks https://infosec.cs.tau.ac.il` – this will open a command-line browser and point it to the course website. If your network setup was correct, you should now see details of packets printed in the main VM :)
    - Note: You'll need to use the `elinks` browser in a few more questions. For your convenience, there's a cheat-sheet on how to use it, at the end of this file.

Question 1 (30 pt)

In this question we'll see how dangerous is using forms over HTTP without encryption (not HTTPS) to perform login. To do this, you'll write a sniffer that extracts the username and password of every person (on your local network) logging in to the course website over HTTP.

Before we start, note that for your privacy, the course website redirects all HTTP traffic to HTTPS, unless it's performed by elinks on our dedicated VM. So, use elinks if you want to test this exercise.

1. (5 pt) Use wireshark to record a login attempt, and save the result to `q1.pcap`.
    a. The login page for the course website is http://infosec.cs.tau.ac.il/2019/login
    b. You'll need to run wireshark with sudo (from the command-line)
    c. Note the HTTP and not HTTPS so that we can sniff insecure traffic
    d. Start wireshark and filter the display to contain only the HTTP traffic
    e. Stop the recording once you can see the relevant packets
    f. Filter the packets so that only the login attempt itself is visible (i.e. just the login request with the user/password, not the loading of the page)
    g. Export only the displayed packets to `q1.pcap`
        i. That is, the single packet of HTTP login request (without other packets of the traffic)

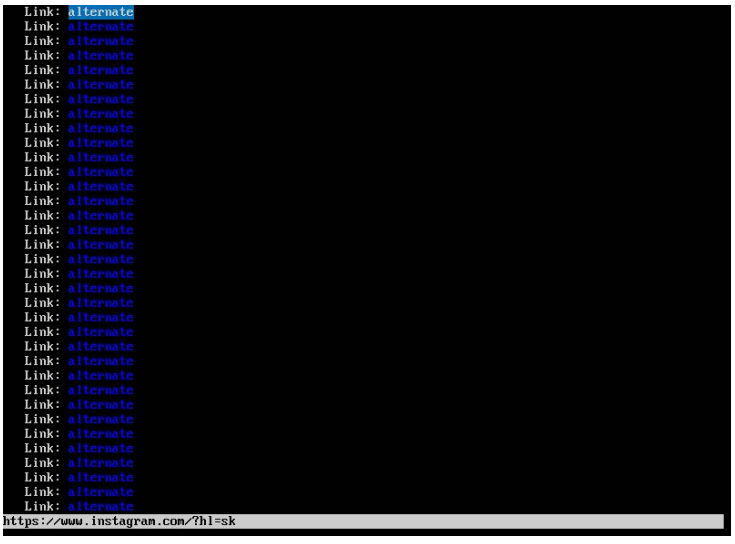
2. (10 pt) Inside the `q1.py` script, implement the `packet_filter` function – this function will filter away (return False for) any traffic which is not on the HTTP port coming **from any HTTP client to any HTTP server**.
3. (15 pt) Inside the `q1.py` script, implement the `parse_packet` function – this function will extract the username and password of any user trying to log-in to the course website. The exact details are in the documentation in the code.
    a. You will want to use the urlparse module to assist you in parsing certain parts of the packet
    b. You can assume that both the username and the password are **non-empty**

Notes:

1. Running the script has to be done with `sudo` (since only root may sniff packets). Once it's running, you can only stop it with Ctrl+C.
2. You can assume the entire HTTP request fits within one packet.
3. For debugging, you can run `q1.py` on the .pcap file (run `q1.py --help` for more details). Note that it should still also work on live traffic, so make sure to test that eventually.
4. Document your solution in `q1.txt`.

Question 2 (30 pt)

In this question we'll try a different attack – instead of stealing the username and password, we will prevent the user from working, by redirecting every attempt to access the course website, to somewhere else. We will eternally prevent students from accessing the course website, by redirecting them to Instagram:



(If you thought Instagram was useless, then taking away it's pictures just took it to a whole new level...)

Inside the `q2.py` script, implement the `get_tcp_injection_packet` function that returns a packet to inject into the TCP session, to redirect attempts to access the course website over HTTP to Instagram.

Notes:

1. Again, run the script with `sudo` and stop it with Ctrl+C
2. To test your code, use the victim machine to access the course website (over HTTP) and see that indeed every attempt is blocked and redirected
3. Make sure you don't hijack any other TCP/HTTP session – only attempts to access the course website.
4. Don't forget to set the flags properly on your TCP packet to **close the connection**.
5. Again, you can assume the entire HTTP request fits within one packet
6. If your injection doesn't work, try recording the connection with wireshark – for some mistakes, it will highlight the packet with an explanation of the error
7. Document your solution in `q2.txt`

Question 3 (40 pt)

In this question we're going to do something more interesting than just hijacking one session – we are going to redirect all of the traffic of the victim to the internet, through our own machine. This would allow us (for example) to hijack every connection, without worrying about winning the 'race' to inject packets before the original target does.

To run `q3.py` we need the following arguments:

`q3.py <our_ip> <our_mac> <gateway_ip> <gateway_mac>`

1. Find the network gateway IP using the `route` command
2. Find the network gateway MAC address using the `arp` command
3. Find our own IP and MAC address using the `ifconfig` command

Now, go over the code in `q3.py` and get the general sense of what's going on in order to implement the following functions:

1. (10 pt) Implement the `is_packet_to_original_gateway` to detect whether a packet was sent from a different machine (not ours!) to the network gateway
    a. Hint: The gateway receives all traffic to the internet, so the IP of the packet is usually not the IP of the gateway itself!
2. (20 pt) Implement the `create_poison` function to create an ARP packet to send to the victim machine, to poison it's ARP cache so that the victim will believe we are the gateway!

3. (10 pt) Implement the `is_stolen_packet` function to detect whether a packet was sent to us instead of the gateway.
    a. Hint: How do we distinguish these packets from packets that are really aimed at us?

Now, here's how you can test this:

1. In the victim machine, send an IP ping to some machine on the internet.
    a. For example, `ping 8.8.8.8`
2. See how responses are received on the victim machine, meaning the packet did reach the internet (through the gateway)
3. On the attacking machine, run `q3.py` (with `sudo` and the above args) and it should immediately find the victim traffic and poison it's ARP cache
4. Once the poisoning happens, the attacking machine should identify stolen packets being received and the victim machine should stop receiving answers to the ping requests
5. You can further verify the success of your attack by looking at the arp cache on the victim machine – do this by running `arp -n` and identifying the attacker MAC address for the gateway on the victim machine
6. When done, kill the script with Ctrl+C

Final notes:

- Document the reasoning and rationale for your code (in addition to documenting the code)
- Yes, you can import any python module already installed on your machine
- Don't use any additional third party libraries that aren't already installed on your machine (i.e. don't install anything)

Elinks Cheat Sheet

- `Esc` - Open the menu
- `G` - "Go to URL" - open the location bar to type an address to go to
- `H` - "History" - open a dialog with links you've visited
- ← / → - Navigate back/forward in pages you visited
- ↑ / ↓ - Navigate to the previous/next points of interests (links, forms, etc.) in the page
- `Enter` - For links/buttons - follow/click the link/button. For form fields, enter the form field to edit it (use ↑ / ↓ to navigate away to the previous/next field)