Exercise 9

Firewalls and Stealth communication

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

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

When you finish solving the assignment, submit your exercise with `infosec push ex-big-brother`.

Question 1 (50 pt)

Part A (10 pt) - Stealth SYN scan

A stealth SYN scan is a method to check whether some TCP ports are open, closed or filtered by a firewall, by sending them a SYN and receiving a SYN/ACK, a RST or nothing. Once the attacker knows which of the server ports are accessible to him, he can plan his attack accordingly.

`stealth_syn_scan(ip, ports, timeout)` in `q1/`**`q1a.py`** is a function performing a SYN scan. This function receives an IP, a list of ports, and a timeout (in case some of the ports are filtered and there's no response), and returns a dictionary from port number (as int) to the strings `'open'`, `'closed'`, or `'filtered'`. Implement the missing functions in `q1a.py` to make it work. Describe your solution in `q1/`**`q1a.txt`**, and explain why this is called a stealth SYN scan.

A note for general knowledge: A real stealth SYN scan, after receiving a SYN/ACK from an open port, would send a RST to close the connection; this isn't necessary in this exercise, but as you might notice during development - it may cause a DoS similar to a SYN flood by using up all the slots in the listening queue for the unresolved SYNs.

Implementation notes:

- We use the `sr` function since it sends all the packets at once, and waits together to the answer from all packets (instead of scanning sequentially)
- Since we use raw sockets, this script has to be run with **`sudo`**
- **IMPORTANT:** Don't add any calls of your own to send/receive packets

How to test this:

- For your convenience, `q1/q1a.py` is a script that receives an IP, a list of comma-delimited ports (for example, 21,22,23) and optionally a timeout (default is 5 seconds), and runs our code.
- To test your script, you can set up another VM as we did in exercise 8
    - Bind and listen on ports on the other VM using netcat (`nc`) or custom Python code
    - Create ports that don't answer at all (not even RST ACK) by dropping traffic using `iptables` (again, on the other VM)
- Scanning localhost is not recommended, as it behaves differently than "real" hosts

Part B (15 pt) - Firewall

To counter the SYN scan, we'll write a simple host-based firewall that records the number of SYNs received from each IP, and if the **number of SYNs in the last 60 seconds exceeds 15**, blocks that IP using `iptables`.

Implement `on_packet(packet)` in `q1/`**`q1b.py`**. This function receives a packet, and if it's a SYN, and there were more than 15 SYNs from that IP in the last 60 seconds, issues one `iptables` command to block that IP (by calling `block(ip)`). You will also need to implement `generate_block_command(ip)`. Describe your solution in `q1/`**`q1b.txt`**.

Notes:

- This is not a course in data structures, but your solution should be adequate; that is, it should be able to run on a server accessed by multiple IPs indefinitely, and **not run out of memory**.
- Also, notice that the number of SYNs per IP is **checked in a sliding window**, and not using time intervals
    - So, for example, 15 SYNs at 12:00:59 and another SYN at 12:01:01 should be blocked
- To test this, you can generate many SYNs from the other VM using netcat. For example, using `for i in `seq 30`; do echo "Run $i"; nc -v <IP> <SOME_PORT>; done`
    - After enough packets, your packets will be dropped and netcat will "hang" while waiting for response
    - You will need to reset your `iptables` between runs, to clear the state of the firewall

- Again, since we use raw sockets, this script has to be run with **sudo**
- You don't need to add a special treatment to avoid blocking ourselves

Part C (10 pt) - Find the problem

To counter the firewall, find a vulnerability in its design; namely, how is it susceptible to a DoS attack? Describe your solution in q1/**q1c.txt**.

Notes:

- We are looking for a DoS attack that exploits **a vulnerability in the design of this specific firewall**, so "use Smurf attack or DNS amplification to flood it" are not valid solutions (as everyone is susceptible to them).

Part D (15 pt) - Fix it

Seeing as our firewall is pretty bad, let's use a different method to counter the SYN scan: instead of blocking it, let's render it useless!

Write a Python script in q1/**q1d.py** that makes every port look open; that is, while it's running on some machine, running q1a.py on that machine will report every port open, which is kinda pointless. Describe your solution in q1/**q1d.txt**.

Notes:

- Again, since we use raw sockets, this script has to be run with **sudo**
- To test this, block all outgoing RST packets from your machine (using iptables), then run the script, and then connect to your VM from the other VM using nc -v <IP> <PORT>. Netcat should say a connection was established, even though it was not.

Question 2 (50 pt)

In this question, we'll be implementing some technology from [1984](). If you haven't read this book, you really, really should.

Winston wants to send Julia his love: from within `q2/a` run `python3 julia.py`, then `python3 winston.py`, and see their passion over TCP. However, Big Brother is listening, and love is forbidden in Oceania.

Part A (5 pt) - Simple spying

Implement `spy(packet)` in `q2/a/`**`bigbrother.py`** so that it tracks TCP packets containing the word `love`, and adds the sender's IP to the `unpersons` set, so the Thought Police can handle him later. Describe your solution in `q2/a/`**`q2a.txt`**.

Part B (15 pt) - Encryption

Winston and Julia know that Big Brother is listening, so they decide to encrypt their data; this way, no one will be able to tell whether their message contains the word `love` or not. Reimplement `send_message(ip, port)` in `q2/b/`**`winston.py`** and `receive_message(port)` in `q2/b/`**`julia.py`** so Winston still sends `"I love you"`, but encrypted with AES (you can chose the key); and Julia receives and decrypts it (she knows the key, too; they've exchanged it beforehand).

1. Make sure `q2/a/bigbrother.py` doesn't suspect this traffic.
2. Describe your solution in `q2/b/`**`q2b.txt`**.

Notes:

- Use PyCrypto (`import Crypto`); it's already installed.
- The key length, mode, IV and other parameters aren't crucial, but choose something reasonable.
- Also, mind the padding: AES data size must be a multiple of 16, and Julia should be able to receive and decrypt any message from Winston, so she can't assume its size. Personally, I like PKCS7 padding.
    - If you're not sure what PKCS7 padding is, read [here]() for an explanation (and [here]() for some more hints)
    - You can also use other padding schemes
    - Your scheme should work for any message - padding with \0 at the end and then removing all \0's is bad, as we may send a message with \0
- It's OK to assume no message is longer than 15 bytes

Part C (5 pt) - Detecting encryption

Big Brother noticed an increase in encrypted traffic, and decided it's forbidden as well. To detect whether the data is encrypted or not, he decided to measure the [Shannon Entropy]() of the TCP payload, as encrypted data has high entropy (> 3.0, meaning it looks like random), whereas normal text has lower entropy (~ 2.0, meaning it has a predictable and consistent distribution).

Reimplement `spy(packet)` in `q2/c/bigbrother.py` so that it also adds to the `unpersons` set the sender's IP of any TCP packets with data whose Shannon Entropy is higher than 3. Describe your solution in `q2/c/q2c.txt`.

Notes:

- The script should also detect packets with `love`, as in part A
- Shannon Entropy can be computed like so:
    - Given a string, get its characters distribution by computing for each character, how many times it appears divided by the string length.
    - Then, sum the product of these values multiplied by their log2 and negate it.
    - In code, this would be:

```
distribution = [float(string.count(c)) / len(string) for c in set(string)]
entropy = -sum(p * math.log(p) / math.log(2.0) for p in distribution)
```

Part D (25 pt) - Stealth communication

This is where things get interesting: Winston and Julia decided to hide their data in the TCP metadata.

Winston would send Julia SYN/ACKs, and put his message in the **reserved bits**. There are only 3 bits there, so he would break his message into triplets of bits, and send them over several packets.

These packets would be from source port 65000, with a serial SEQ number (i.e., 0 for the first triplet, 1 for the second, 2 for the third and so on); and in the ACK number, he'll put the total number of triplets, so Julia knows when she has received them all.

Reimplement `send_message(ip, port)` in `q2/d/winston.py` and `receive_message(port)` in `q2/d/julia.py`, so Winston still sends "I love you", but via the metadata of SYN/ACKs; and Julia receives and reconstructs it. Make sure `q2/c/bigbrother.py` doesn't suspect this traffic. Describe your solution in `q2/d/q2d.txt`.

Notes:

- Some packets may be received more than once when you use Scapy to sniff packets on the local machine (you may get the packet both when it's "sent" and when it's received)

- Your code should work regardless of whether packets are received once or twice (hint - it doesn’t matter if your code is written correctly)
- This means you should also stop the capture when a condition is met, and not by count of packets

Final notes:

- Document your code
- Don’t install/use any third party libraries that the grader won’t have
- If your answers take an entire page, you probably misunderstood the question