Lab 6: Wireshark & Scapy

Wireshark is an open-source packet analyzer that captures and visually presents network traffic data, allowing users to inspect network protocols and troubleshoot network issues in real time. It is commonly used by network administrators to diagnose problems such as slow networks, dropped packets, or unusual traffic. Its wide range of protocol support and its ability to decrypt traffic when provided the right encryption keys make Wireshark a go-to tool for both real-time and post-event network analysis. In cybersecurity, Wireshark is invaluable for detecting malicious activity, such as packet injection attacks, man-in-the-middle attacks, and other types of network intrusion. For instance, in 2013, Wireshark played a key role in analysing the infamous Edward Snowden leaks, allowing network analysts to monitor the data flow of classified information on public networks.

Scapy is a versatile Python-based packet manipulation tool that allows users to craft, send, sniff, and analyse network packets. While Wireshark focuses on visualising and analysing captured traffic, Scapy goes a step further by enabling users to generate custom network packets. This makes it extremely useful in both offensive and defensive security contexts. Scapy is widely used in penetration testing, where security researchers simulate attacks by sending malformed or malicious packets to identify vulnerabilities in network devices or applications.

Part 1: Investigating HTTP Traffic with Wireshark

In this part, you will start to explore Wireshark. As mentioned in the introduction, Wireshark is a software tool used to capture packets and analyse them, and is an extremely powerful tool used by network engineers and blue-teamers (and often even attackers) to analyse network traffic. The standard file format for storing such packet captures is the ".pcap" format which you can read more about (if you're interested) here.

- Open the provided `http_traffic.pcap` file using Wireshark. This is the traffic generated by a client retrieving a web page from a remote server.
- Filter the captured traffic to show only HTTP packets (enter `http` on the display filter bar).
- Create a report with the filename `report.pdf`. Investigate the packets corresponding to the first pair of messages to and from the server (i.e., not the */favicon.ico* request/response). Answer the following questions (with respect to those two packets) in your report:
    a. What version of HTTP is the client's browser running (1.1 or 1.0)? What version of HTTP is the server's browser running?
    b. What is the IP address of the client's machine? What is the IP address of the server's machine?
    c. What is the status code returned from the server to the client's browser?
    d. When was the HTML file which the client is retrieving last modified at the server?

e. How many bytes of content are being returned to the client's browser?

Part 2: Basic Authentication over HTTP

Some websites used to use Basic authentication to authenticate users. Unfortunately, some insecure websites still do the same. You might have observed that HTTP traffic is unencrypted. In particular, if you send passwords over HTTP, they will be clearly visible to anyone who has access to the network packets.

- Read about Basic authentication from here. You only need the information in the first paragraph.
- Open the provided `http_basic_auth.pcap` file using Wireshark and inspect the packets.
- Answer the following in your report:
    a. What is the response of the server (status code and phrase) to the first GET request sent by the client? Why does this happen?
    b. Note that the second GET request succeeds. Why? What new field is added in the request to the server?
    c. What is the username and password of the users?

Part 3: Analysing and Decrypting TLS Traffic

To avoid situations where the password is sent in the clear, the contents of the packets are often encrypted and sent. Transport Layer Security (TLS) is one of the protocols used to send data over a public network while ensuring confidentiality of the packets' contents.

- The first challenge which arises is how can two machines across the Internet which have never spoken before be able to establish a method of communication amidst themselves such that no eavesdropper could understand their communication. Note that any and every communication between them can be read by the eavesdropper, and there's no way for them to agree on anything specifically secret to them beforehand. Public-key cryptography aids in this process. Read about it (at a high-level, not at a technical level) and explain in a few sentences how it achieves secure communication between two parties across the public Internet. This needs to only be an informal explanation, you don't need to delve into any mathematics for deep technical details for this. Write your explanation in the report.
- Read more about TLS from here.
- Open the `tls_traffic.pcapng` file.
- You are provided a pre-master key in the file `premaster.txt`. Load the pre-master key in Wireshark (Edit -> Preferences -> Protocols -> TLS -> (Pre)-Master-Secret log filename).

- Decrypt the traffic, inspect the TLS stream embedded in TCP stream 0 (after decrypting, inspect the first TLS packet and follow the TLS stream), and answer the following questions in your report:
    a. Which cipher is common between both SSL end points?
    b. Which protocol was used for communication?
    c. What is the master key used for the session (in hex)?
- If you're interested, you can find a detailed step-by-step description along with an example of the TLS 1.3 protocol here.

Part 4: Capturing Your Own Traffic

You don't just need to rely on other packet capture files, you can create your own! Wireshark allows you to capture packets on your network, either on the loopback interface or on any of the wired/wireless interfaces. In fact, it also allows you to capture USB traffic! For those of you into electronics, Wireshark also allows you to capture data sent by your machine to a microcontroller!

- Start capturing your network traffic and navigate to http://astoundinggoodcoolmelody.neverssl.com/online/. Once the page loads, stop the packet capture. Inspect the packets in Wireshark. What is the last time the page was modified? Put your answer in your report along with a screenshot of Wireshark with the appropriate packet selected and state (with respect to the screenshot) how you inferred the modification time.
- Save your packet capture as `my_http_capture.pcap`.
- Use your socket programs from the last lab and capture packets on your loopback interface using Wireshark. Identify which packet(s) contain the data sent by sender.c. Follow the respective UDP stream (Right click on packet -> Follow -> UDP stream) and see that you can fully see the conversation. Attach a screenshot of the UDP stream showing the data sent by your sender.c.
- Save your packet capture as `my_udp_capture.pcap`.

Part 5: Using Scapy for Packet Manipulation

Scapy, as mentioned in the introduction, is a python module which allows you to craft your own packets and explore packet capture files programmatically. Their website (https://scapy.net/) is an excellent resource for tutorials for learning scapy. You could follow any of the links under the "Good Introductions to Scapy" section on their website, or directly go for the documentation which has an excellent interactive tutorial! Use Scapy to accomplish the following tasks:

- Craft an ICMP Echo Request packet and send it to any IP address of. Capture the ICMP Echo Reply response. Submit the script in a file named `scapy_icmp.py`.

- Modify a default DNS query packet to request a non-standard DNS record type (e.g., **ANY**) and send it to a DNS server. Submit the script in a file named `scapy_dns_ns.py`.
- Create a TCP SYN packet with a spoofed source IP address and send it to any target IP. Note that you are able to send packets with a source IP which is not your own. Think about how you might use this as an attacker! Submit the script in a file named `scapy_ip_spoof.py`.
- For each of the above exercises, make sure to capture the packet you send out using Wireshark and attach screenshots of each of them with an appropriate packet highlighted and a brief explanation of why that packet is indeed the packet you were asked to craft.

Submission Instructions:

Create a folder called **<roll_no>_lab6_submission/** (your roll number should be in lower case; e.g., if your roll number is 22B56789, your folder should be named *22b56789_lab6_submission/*). The folder should contain the following files in a flat hierarchy (that is, there should be no other folders inside your submission folder):

1. `report.pdf`
2. `my_http_capture.pcap`
3. `my_udp_capture.pcap`
4. `scapy_icmp.py`
5. `scapy_dns_ns.py`
6. `scapy_ip_spoof.py`

Use **tar -cvzf** to compress your folder into **<roll_no>_lab6_submission.tar.gz**. Your roll number should be in lowercase here too.

Please **strictly** follow the file naming format mentioned above. Any deviation from the same will attract a penalty.