**CS155: Computer Security** **Spring 2018**

# Project #3

Due: Wednesday, June 6 at 11:59 pm (at most one late day allowed)

## Introduction

This project is all about network security. Specifically, you will both use existing software to examine remote machines and local traffic as well as play the part of a powerful network attacker. Parts one and two should show you how a simple port scan can reveal a large amount of information about a remote server, as well as teach you how to use Wireshark to closely monitor and understand network traffic observable by your machine. Part three will focus on a dump of network traffic in your mobile network, and will teach you how to identify different types of anomalies. Finally, in part four, you will get to implement a DNS spoofer that hijacks a HTTP connection. In doing so, you will learn how to use Python packet manipulation libraries and TCP sockets, and solidify your understanding of the mechanics and shortfalls of the DNS and HTTP protocols.

## Part 1: Port scanning

Port scanning is a method that can be used by an attacker to probe which ports are open on a given host, learning details about which software the server is running on publically-addressable interfaces. With this information, an attacker gains a better understanding of where and how to attack the victim server. Port scanning takes advantage of conventions in TCP and ICMP that seek to provide a sender with (perhaps too much!) information on why their connection failed.

In this part, you will use the nmap tool (https://nmap.org; https://en.wikipedia.org/wiki/Nmap) to scan the server **scanme.nmap.org**. By doing so, you should be able to see the powerful information that a simple scan can reveal. In your scan, make sure to:

1. Only scan **scanme.nmap.org!** Do not scan any other servers. You should only scan a server if you have explicit permission from the server operator to do so.
2. Record the traffic with Wireshark (see part 2)
3. Use a TCP SYN scan. (Hint: Read the nmap man pages to find the appropriate flag to use.)
4. Enable OS detection, version detection, script scanning, and traceroute. (Hint: This is a single flag.)
5. Do a quick scan (-T4).
6. Scan all ports.

The Stanford network rate limits nmap scans, so if you are running the scan from within the Stanford network, it may take about 30 minutes to complete. Off the Stanford network, it should take between 1 and 5 minutes to complete. When you get the result, report the following about the target server (scanme.nmap.org) based on the results of the scan:

1. What is the full command you used to run the port scan (including arguments)?
2. What is the IP address of scanme.nmap.org?

3. What ports are open on the target server? What applications are running on those ports? (For this part, you only need to report the service name printed by nmap.)

4. The target machine is also running a webserver. What webserver software and version is being used? What ports does it run on?

Note: running nmap from the VM takes a few hours for some people. You can install nmap/Wireshark locally (which may be faster) or just leave it running for a while in the VM. To prevent your computer from sleeping during the scan you can run caffeinate from the terminal on macOS or disable sleep in settings for Windows/Linux.

### Part 1 Deliverables

**PortScanAnswers.txt**: A text file containing your answers to questions 1–4 above. Make sure you follow the answer format in the file, since your responses are autograded. If you don't follow the format, you will lose points.

## Part 2: Wireshark packet sniffing

Wireshark is a tool to monitor local network traffic. Wireshark has access to complete header information of all packets on a monitored interface and presents a helpful GUI for understanding the structure of different protocols. Because of this it can be a valuable debugging tool for networking projects, as you will see in part 4.

Use the Wireshark packet analyzer (https://www.wireshark.org/; https://en.wikipedia.org/wiki/Wireshark) to examine the traffic generated by nmap during the scan in Part 1. You will need to start Wireshark and record traffic on the interface nmap will use to scan before actually running the scan. The VM we provide has wireshark installed, just run **wireshark**. You can also install and run wireshark locally.

When you get the result, take a look at the Wireshark capture. Use Wireshark's filtering functionality to look at how nmap scans a single port. Report the following about the target server based on the results of the scan:

1. What does it mean for a port on scanme.nmap.org to be "closed?" More specifically, what is the TCP packet type, if any, the server gives in response to a SYN packet sent to port that is "closed?"

2. What does it mean for a port on scanme.nmap.org to be "filtered?" More specifically, what is the TCP packet type, if any, the server gives in response to a SYN packet sent to port that is "filtered?"

3. In addition to performing an HTTP GET request to the webserver, what other http request types does nmap send?

Once again, please answer all questions briefly; no response should take more than three sentences.

### Part 2 Deliverables

**WiresharkAnswers.txt**: A text file containing your answers to questions 1–3 above.

## Part 3: Packet log processing

You have been given a dump of packet header information for the network that your mobile device is on (trace.txt). Your device's IP address is 10.30.22.101. Lately, there have been a number of sketchy occurrences on this network. In order to learn more about network security, your job is to sift through the packet header information and detect some anomalous behavior on the network. Since the volume of packets is large, it is expected that you will write scripts in the language(s) of your choice to do the following:

1. Your mobile device has accessed a number of websites. List 5 IP addresses of websites that have been accessed by your mobile device. (Hint: What port is typically used for webservers? Your mobile device's IP address is 10.30.22.101. There may be more than 5 valid answers to this question, but you should only list 5.)

2. Someone sketchy has been looking for attack vectors into hosts on this network by scanning for open ports on a machine. (Hint: What kind of pattern would you expect from a host that is port scanning other hosts? How might you isolate this pattern in the packet dump?)

   (a) What is the IP address of the origin of the port scan?

   (b) What is the IP address of the host that was scanned?

   (c) What range of ports was scanned?

3. There has been an unusually high volume of SYN packets going from one host on this network to another host on this network. This can be indicative of a type of Denial of Service (DoS) attack called a SYN flood.

   (a) What is the IP address of the SYN-flooding sender?

   (b) What is the IP address of the SYN-flooded receiver?

   (c) How many SYN packets were sent from this sender to this receiver?

4. Some malware on your phone has caused it to behave improperly and inject a malicious packet in the network. More specifically, your phone has sent a packet during a TCP connection that it should not have sent. Doing so can sometimes cause problems on the network and crash hosts that are not implemented correctly. What is the checksum value of the injected packet? (Hint: Under what conditions will a client send a packet with a sequence number that has already been sent and acknowledged? Should these packets always have identical content? What does the checksum of a packet indicate?)

### Part 3 Deliverables

1. **PacketSniffingAnswers.txt**: A text file containing your answers to questions 1–4 above.

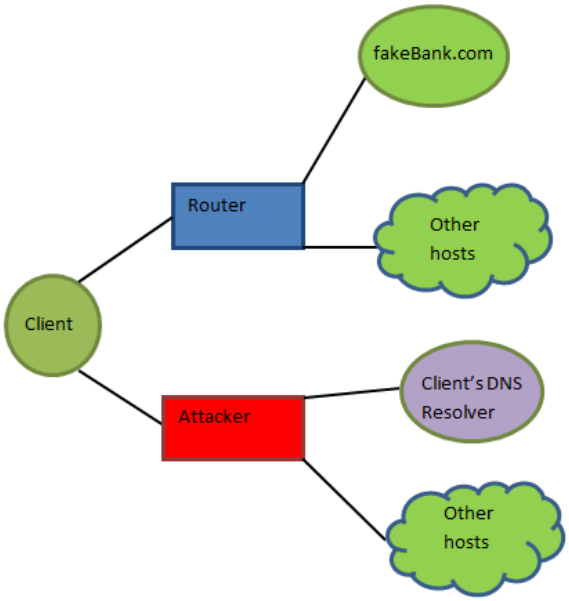
2. Any scripts you wrote to answer the above questions.

## Part 4: DNS Spoofing and Man in the Middle

Now that you've learned how to use existing programs to both analyze network traffic locally and discover what software is running on remote machines, it's time to write an exploit of your own. By doing so, you will reveal security issues associated with the lack of authentication in the DNS

and HTTP protocols and learn how to use powerful packet-manipulation libraries. In this task, you will play the part of a network attacker who tricks a victim web-browser to connect to the attacker's server instead of a site the victim wanted to visit. By doing so, the attacker can perform a man-in-the-middle attack to forward the victim's requests to and from the site without being noticed while stealing confidential information along the way. To do so, you will have to spoof a DNS response to trick the user into associating the hostname "fakeBank.com" with the attacker's IP address instead of its real address. You will use the "scapy" Python library to manipulate packet headers in a quick and reliable manner. Once the DNS response has been successfully spoofed, you will accept a connection from the victim and forward all requests to and from the actual webserver for "fakeBank.com".

The topology of the system is shown below. For this assignment, we will act as if the attacker has infected a router between the victim and the victim's DNS resolver and can thus sniff packets to the resolver and prevent packets from being forwarded from the resolver to external clients.




Your steps are the following. All code changes will be made to attacker/attacker.py.

Tasks:

1. Open a TCP socket on the attacker's web port to accept connections from victims.
2. Sniff for DNS lookups on the network. Use scapy's `sniff` function to trigger a callback when a DNS packet is detected. This callback should respond to DNS queries with a fake DNS response which directs the victim to the attacker's IP address.
3. Once the DNS response has been sent, wait for the client to connect to the attacker's web port and accept HTTP requests on that socket.
   (a) The attacker should read the contents of each request and check for confidential information to log to the attacker's filesystem. For this assignment, you will look for a POST request with parameters `username` and `password` and call the `log_credentials` function to log this data to the filesystem.

(b) You will then forward the exact contents of the HTTP requests to the actual host *fakeBank.com* (whose IP should be resolved by calling the provided **resolveHostname** function) and relay the host's response back to the victim. The downloaded contents in `client/lib/downloadedPage.txt` should match `httpServer/lib/fileToDownload.txt` if the forwarding is handled correctly.

4. Close the connection and exit the program once the server and client are done communicating. The client will signal the end of its session with a POST to `/post_logout`.

## Additional Information

- We will provide you with a VM to develop on, which already has all requirements installed. We highly recommend you develop on this VM. It should be much faster than the project 1 VM, but also note that you can increase the RAM of the VM to make it run faster depending on the amount of spare resources on your host machine.
- All hosts will have local IPs, communicating locally over the local loopback interface ('lo'). Running ./run.sh will create and run the topology, giving an IP address to the client, DNS server, web server and attacker and launching each process. Note: the client tries to resolve the server's hostname and login automatically (see client.py for details).
- When the network is started using run.sh, there are a series of logs generated in the `log` folder. The logs are generated by the client, web server and dns server running in the background. Use stop.sh to stop the network's background processes. Finally, use reset.sh to reset the client and attacker lib directories and clear the logs.
- For this assignment, assume the attacker controls the router between the victim and the resolver. Thus, there will be no race conditions between the attacker's response to the victim and that of the actual resolver. Note that this often would not be the case in reality.
- Use Wireshark to debug on the 'lo' local loopback interface. You gained some exposure to Wireshark in part 2, and being able to see exactly what packets are being sent across the network is a powerful debugging tool.
- Scapy's sniff function can trigger a callback whenever certain packets are detected on the network.
  - To pass in additional arguments to a callback, consider using a lambda function. For example:

    ```
    cb = lambda originalArg, args=(extraArg1,extraArg2):callback(originalArg,args)
    ```

    would invoke the callback with originalArg, extraArg1, and extraArg2.
- Scapy builds packets by constructing each individual layer and stacking them together. You pass in various header fields and scapy fills in default values for the rest. Scapy documentation can be found at https://scapy.readthedocs.io/en/latest/, but doesn't provide much information on individual parameters so below we provide some headers with some possible parameters. You don't necessarily need to use all of the headers/parameters listed below.

```
IP(src="sourceip",dst="dest_ip",proto=protocol, ttl=TTL)
TCP( sport=source_port, dport=dest_port, flags="TCP_flags" (A=ack, F=fin etc.,  ) )
```

```
UDP(sport= X, dport=X  )
DNS(id=id, ns=DNSRR(nameserver), ancount=X, nscount=X, an=DNSRR(answer), ...)
DNSRR(rdata="IP_address of host", rrname="host.com", ttl=TTL)
```

  - Additional flags for DNS follow DNS header convention (aa, rd, qr, ...). For more information on DNS flags see: http://www.networksorcery.com/enp/protocol/dns.htm

- Use Python's socket library to create both TCP client and server sockets. Refer to Python documentation: https://docs.python.org/2/library/socket.html
- Remember that HTTP 1.0 uses a unique TCP connection for each request (you can confirm this with Wireshark!).

### Part 4 Deliverables

**attacker.py**: Your completed attacker.py file.

## Submitting

1. Make sure the deliverables mentioned in this writeup for part[1,2,3,4] are in their respective directories; misplaced deliverables may not receive credit. **Many items will be auto-graded, so be careful to follow the given formats exactly.**
2. Run the following command from the project root directory to generate the submission tarball submission.tar.gz: `make submission`.
3. Upload the submission tarball to Gradescope.