Project 3: Network Security

This project is due on **Friday, March 13, 2020** at **6 p.m.** and counts for 13% of your course grade. Late submissions will be penalized by 10% of the maximum attainable score, plus an additional 10% every 4 hours until received. Late work will not be accepted after the start of the next lab (of any section) following the day of the deadline. If you have a conflict due to travel, interviews, etc., please plan accordingly and turn in your project early.

The code and other answers you submit must be entirely your own work, and you are bound by the Honor Code. You may consult with other students about the conceptualization of the project and the meaning of the questions, but you may not look at any part of someone else's solution. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Starter code is provided on GitHub Classroom, which you will also use to submit your solutions. See "Submission Details" at the end of this spec for instructions about how to access GitHub Classroom.

Your code for this project must be written in the latest version of **Go** (1.13). Be sure you use the correct version, or your solution will not run on the autograder. You may only use standard libraries that ship with Go or modules that we either explicitly permit or include in the starter code. Completing the Tour of Go at https://tour.golang.org/ will provide the base knowledge necessary to complete the project. Please finish the entire Tour before coming to office hours with questions about the language.

Solutions must be submitted via Github and Gradescope, following the submission checklist below.

Introduction

This project will introduce you to network protocols, to network packet trace analysis, and to common network attacks.

Objectives:

- Gain exposure to core network protocols and concepts.
- Learn to apply manual and automated traffic analysis to detect security problems.
- Understand offensive techniques used to intercept and manipulate network traffic.
- Understand the mechanics of man-in-the-middle attacks.

Read this First

This project asks you to perform attacks, with our permission, against a target network that you are hosting for this purpose. Attempting the same kinds of attacks against other networks without authorization is prohibited by law and university policies and may result in *fines, expulsion, and jail time*. **You must not attack any network without authorization!** Per course policy, you are required to respect the privacy and property rights of others at all times, *or else you will fail the course.* See "Ethics, Law, and University Policies" on the course website.

Part 1. Exploring Network Traces

Security analysts and attackers both frequently study network traffic to search for vulnerabilities and to characterize network behavior. In this section, you will examine a network packet trace (commonly called a "pcap") that we recorded on a sample network we set up for this assignment. You will search for specific vulnerable behaviors and extract relevant details using the Wireshark network analyzer, which is available at https://www.wireshark.org.

To get started, download your unique pcap file from **https://project3.eecs388.org**, and open it using Wireshark. Familiarize yourself with Wireshark's features, and try exploring the various options for filtering and for reconstructing data streams.

Concisely answer the questions below. Except where noted, each response should require at most 2–3 sentences.

1. Multiple devices are connected to the local network. What are their MAC and IP addresses?

2. What type of network does this appear to be? List the letter of your answer (from among the choices below), and point to evidence from the pcap that supports this.

   A. Personal Area Network B. Home Network C. Wireless Local Area Network
   D. Campus Area Network E. Metropolitan Area Network F. Wide Area Network
   G. Storage-Area Network H. Enterprise Private Network

3. One of the clients connects to an FTP server during the trace.

   (a) What is the DNS hostname of the server it connects to?

   (b) Is the connection using Active or Passive FTP? Write "Active" or "Passive".

   (c) Based on the packet capture, what's one major vulnerability of the FTP protocol? Simply give the letter corresponding to your choice.

   A. Padding Oracle B. Doesn't use encryption C. Doesn't use authentication
   D. I don't know!

   (d) Name at least two network protocols that can be used in place of FTP to provide secure file transfer. Use the associated acronyms (e.g., HTTP should be your answer for Hyper Text Transfer Protocol).

4. The trace shows that at least one of the clients makes HTTPS connections to sites other than Facebook. Pick one of these connections and answer the following:

(a) What is the domain name of the site the client is connecting to?

(b) Is there any way the HTTPS server can protect against the leak of information in (a)? Answer "Yes" or "No", and explain why.

(c) During the TLS handshake, the client provides a list of supported cipher suites. Are any of these cipher suites worrisome from a security or privacy perspective? Why? List the cipher suite or suites that cause the concern.

(d) What cipher suite does the server choose for the connection?

5. One of the clients makes a number of requests to Facebook.

(a) Even though logins are processed over HTTPS, what is insecure about the way the browser is authenticated to Facebook? List only the letter of your choice, from among the options below:

A. Credentials sent in plaintext
B. Passwords are hashed but still visible, leaving them vulnerable to length extension
C. Session cookies are sent in plaintext
D. Ciphersuite chosen is vulnerable to padding oracle
E. The API token is sent in plaintext
F. Nothing at all.

(b) Copy the data from the pcap that proves your answer above.

(c) The user sent a message while on Facebook. What was the message?

What to submit Submit your answers to the online assignment in Gradescope.

Part 2. Anomaly Detection

In Part 1, you manually explored a network trace. Now, you will programmatically analyze a pcap file to detect suspicious behavior. Specifically, you will be attempting to identify port scanning and ARP spoofing.

Port scanning is a technique used to find network hosts that have services listening on one or more target ports. It can be used offensively to locate vulnerable systems in preparation for an attack, or defensively for research or network administration. In one kind of port scan technique, known as a `SYN` scan, the scanner sends TCP `SYN` packets (the first packet in the TCP handshake) and watches for hosts that respond with `SYN+ACK` packets (the second handshake step).

Since most hosts are not prepared to receive connections on any given port, typically, during a port scan, a much smaller number of hosts will respond with `SYN+ACK` packets than originally received `SYN` packets. By observing this effect in a packet trace, you can identify source addresses that may be attempting a port scan.

ARP spoofing is an attack that exploits the Address Resolution Protocol (ARP), the scheme used to discover the MAC address associated with a given IP address within a network. When Device A needs to send a packet to Device B on the network, it initially only knows the IP address of Device B and needs to determine Device B's MAC address. If Device A does not have this information already, it broadcasts an ARP packet to all computers on the local network, asking which device is associated with the IP address for Device B. Normally, Device B would respond with an ARP reply message containing its MAC and IP addresses. Device A then caches this information before sending the packet.

Because ARP packets are not authenticated, any device can claim to have any IP address. Furthermore, most network devices automatically cache any ARP replies they receive, regardless of whether they were ever requested in the first place. In an ARP spoofing attack, the attacker repeatedly sends unsolicited replies claiming to control a certain address with the aim of intercepting data bound for another system, thereby performing a man-in-the-middle or denial-of-service attack on other users on the network.

Your task is to develop a Go program that analyzes a pcap file in order to detect possible `SYN` scans and ARP spoofing attacks. To do this, you will use `gopacket`, a library for packet manipulation and dissection. It is available in most package repositories. You can find more information about `gopacket` at https://godoc.org/github.com/google/gopacket.

Your program will take the path of the pcap file to be analyzed as a command-line argument, e.g.:

```
go run detector.go capture.pcap
```

The printed output should begin with the line `Unauthorized SYN scanners:`, followed by the set of IP addresses (one per line) that sent more than 3 times as many `SYN` packets as the number of `SYN+ACK` packets they received. In calculating this, your program should silently ignore packets that are malformed or that are not using Ethernet, IP, and TCP. The line immediately following these IP addresses should print the line `Unauthorized ARP spoofers:`, followed by the set of MAC addresses (one per line) that send more than 5 unsolicited ARP replies. Unsolicited replies are those which contain a source IP and target MAC address combination that does not correspond to a previous request (each request should correspond to at most one reply).

A large sample pcap file captured from a real network can be downloaded at https://files.eecs388.org/sample.pcap. (You can examine the packets manually by opening this file in Wireshark.) For this input, your program's output should match the following, with the addresses in each section in any order:

```
Unauthorized SYN scanners:
128.3.23.2
128.3.23.5
128.3.23.117
128.3.23.150
128.3.23.158
128.3.164.248
Unauthorized ARP spoofers:
7c:d1:c3:94:9e:b8
14:4f:8a:ed:c2:5e
```

What to submit Submit a Golang program that accomplishes the task specified above as a file named `detector.go`. You should assume that `gopacket` is available, and you may use standard Go system libraries, but your program should otherwise be self-contained. We will grade your detector using a variety of different pcap files.

Part 3. DNS Spoofing

Some of your friends at the Ross School of Business have decided to open up a bank. However, in order to cut costs they have decided to employ students who have not taken 388 to build their site. Not knowing about the benefits of encryption, the students decide to serve it using plaintext HTTP.

After a few weeks, your friends notice that, for some reason, clients are seeing money being transferred to the wrong account. Coming to their senses, they decide to hire you, a clever 388 student, to diagnose the problem. Appalled at the use of HTTP in a banking site, you suggest that someone is man-in-the-middling the users' connections. Your Ross friends scoff at you, declaring this **IMPOSSIBLE**. Your goal is to build a program that demonstrates a man-in-the-middle attack against the banking site, to prove that this is a real security problem.

Your friends have provided a sample network for you to attack, which simulates the average user of the banking site. It consists of a DNS server, a client, a web server that hosts their site, and the position of the attacker. The machines are described in more detail below.

The client: The client is at address 127.0.0.1. It will be the victim in this attack. The client's IP address may change for autograding purposes.

The DNS server: The DNS server is at address 127.0.0.3:8090. The client queries this server to get the address of bank.com. The DNS server's IP address and port may change for autograding purposes.

The bank web server: The HTTP server for bank.com is at address 127.0.0.2:8088. You should always be forwarding requests to this address. For autograding, bank.com will **always** be at 127.0.0.2:8088.

The attacker: You will be at address 127.0.0.5. You should host your MITM proxy at this address on port 8088. This address and port will remain the same for autograding purposes.

In this network, you will only be in a passive listener role. You will only have the ability to spoof packets to the client. You will not have the ability to halt packets between the client and DNS server. This, in most networks, would cause a race in which you need to respond faster than the DNS server. However, to simplify the project, we made it so that the DNS server *never* responds. *Hint: You may want to change the DNS server to respond, in order to debug your spoofed packet.*

Vulnerabilities The vulnerabilities that you will be exploiting during this attack are the lack of authentication in DNS and the lack of encryption in plaintext HTTP. Since DNS is not authenticated, anyone is able to respond to a DNS request that they see going over the wire. This means an attacker positioned in the network can respond to a DNS request with a false address, and thereby trick clients into connecting to the wrong IP address for the domain. Since HTTP is unencrypted and unauthenticated, the man in the middle is able to steal information from the requests and even modify them. Your job is to create a program that exploits these vulnerabilities to meet the requirements below.

Requirements:

1. When the client queries a DNS A record for bank.com, your program should send a spoofed response containing the attacker's address, 127.0.0.5, in under 2 seconds. You should ignore DNS queries for other names or record types.

2. Your program should listen for HTTP requests at 127.0.0.5:8088. It should pass through all requests to the bank server at 127.0.0.2:8088 and return the server's responses to the client. All requests and responses (including the HTTP headers) should be passed unmodified, except as listed below.

3. Your program should support multiple concurrent DNS queries and HTTP requests.

4. Whenever a client makes a POST request to a URL containing "login", your program should print the credentials, which are the values of the "username" and "password" parameters.

5. Whenever the client makes a POST request with the parameter "to", your program should change the value to "Colin". Before forwarding the server's response to the client, your program should reverse the change, so that the "to" parameter in the response contains the value the client actually sent.

6. Steal each cookie the client sends or the server sets. That is, any time the client sends the "cookie" request header or the server response contains the "set-cookie" header, your program should print the value.

7. Your program should store a copy of any ".pdf", ".jpg", or ".csv" files that the client downloads from the server (based on the file extension in the request URL). Save them in a directory called "results/". Name the files according to the order in which they were requested with the appropriate file extension. For example, "1.jpg", "2.pdf", "3.pdf", "4.csv", etc. The directory will exist and be empty when we grade.

8. Your program should exit immediately when the client navigates to any URL on the bank site that ends with "/kill", by calling `os.Exit(1)`. You don't need to respond to the request.

Running Your Solution: For this project, we released an Ubuntu VM accessible at https://files.eecs388.org/388-proj3-vm.tar.gz that contains all of the necessary dependencies. We recommend that your develop using the VM, as we cannot help debug issues that arise from other system configurations. Always run the solution using the `./run.sh` script provided with the starter code. You can use the options `start`, `stop`, `restart`, and `status`. The starter code will not work until you modify it appropriately, which may require commenting out imports and/or functions. You must run your solution as root.

What to submit Submit a Golang program that accomplishes the task specified above, as a file named `mitm.go`. You should assume that all libraries included in the starter code are available, and you may use the Go standard libraries, but your program should otherwise be self-contained. We will grade your solution using a variety of different addresses for the client and DNS server.

Submission Details

1. Create a repo using this GitHub Classroom link.
2. Check that your repo contains the starter files. If you have any issues with GitHub Classroom, email `eecs388-staff@umich.edu` for assistance.
3. At each deadline, your repository will be cloned automatically for grading.

   If you need to submit late, you must report the hash of the commit you want graded using the late submission form. The late penalty will apply to the timestamp of the form submission, not the timestamp of the commit.

Submission Checklist

Make sure you have completed the following items by the deadline:

- ☐ Submit responses for Part 1 to the Gradescope assignment.
- ☐ Push the following files to GitHub:
  - ☐ `detector.go` Your Go source code for `SYN` scan and ARP spoof detection.
  - ☐ `mitm.go` Your Go source code for the MITM attack.