Project 2: Network Security

This project is due on **Tuesday, March 15** at **11:59pm**. This is a group project: you can work in teams of two and submit one writeup per team. If you are having trouble forming a teammate, please post to the #partner-search channel on Slack.

Your answers here must be entirely your own work and that of your partner. You may discuss the problems with other students outside your group, but you may not directly copy answers, or parts of answers.

Part 1. Man-in-the-Middle (10 points)

One major concern of any networked application is the possibility of a Man-in-the-Middle attack. These attacks leak confidential information, modify the contents of the messages, and even inject malicious data! In this section, we will write a malicious web proxy that modifies unencrypted webpages by injecting a JavaScript payload.

Part 1.1: Simple Web Servers in Python

We will write the malicious web proxy in Python3. Python provides a simple web server as part of the standard library, which we can extend for our purposes. Below is an example of this web server, http.server.SimpleHTTPRequestHandler. It listens on port 8080, and responds to HTTP GET requests with a simple string. Make sure you can run this web server, and connect to it through your web browser. You will need to use a special IP address to browse to your local computer, called a loopback address: *127.0.0.1:8080*.

```
import http.server
import socketserver

class Server(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        self.wfile.write(b'Sample response')

port = 8080
server = socketserver.ThreadingTCPServer(('', port), Server)
print("[*] Serving on port {}".format(port))
server.serve_forever()
```

You do not need to turn anything in for this part, but make sure you can run the web server.

Part 1.2: Building a Malicious Web Proxy

After verifying that you can run and interact with your web server, we need to insert it in the middle of a web conversation. An actual attacker would need to somehow compromise a victim's router or WiFi network to get between them and the website they are visiting, but because we are testing this locally we can simply ask the browser to route all requests through the proxy server.

In your browser settings, set the HTTP proxy to be the address of your local web server (the exact steps will vary between browsers). This will route all web traffic to your web server, rather than their regular destinations. Keep in mind that this will disrupt your regular web browsing, so you will have to turn it off when you are finished testing! Alternatively, you can use a browser extension such as FoxyProxy to specify individual sites which should and should not be proxied. After setting the proxy address, browse to any unencrypted HTTP webpage such as http://neverssl.com or http://insecure.csci3403.com. You should see the sample response of your local web server, rather than the original content of the page.

Now we can work on turning this web server into a web proxy. Your proxy should take two command-line arguments: the port to run on, and a file containing a malicious JavaScript payload to inject into websites. Rather than returning the same string each time like you did in the server example, your proxy should request whichever web page the user originally browsed to, then return it after appending the malicious piece of JavaScript code.

Injecting JavaScript could be used to log keystrokes, steal cookies, or any number of nasty things, although for testing you can show a proof of concept by using this payload:

```
<script>alert("Payload injected successfully!")</script>
```

This will cause an alert dialog box to pop up, showing that you were able to execute JavaScript on the site.

If your proxy is working, you should be able to browse to any unencrypted HTTP URL and see the original site, but they should all run your specified JavaScript payload when you visit them. You can use the code below as a starting point:

```
import http.server
import urllib.request
import socketserver
import sys
```

```
if len(sys.argv) < 3:
    print('Usage: python3 mitm.py <port> <payload file>')
    exit(1)

class MaliciousProxy(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        print('[*] Got request for: {}'.format(self.path))
        # Your code will go here

port = sys.argv[1]
server = socketserver.ThreadingTCPServer(('', port), MaliciousProxy)
print("[*] Serving on port {}".format(port))
server.serve_forever()
```

Specifications:

- Your proxy server needs to be a Python 3 file which takes the port and payload file as command-line arguments.
- Your proxy server only needs to handle HTTP GET requests, and does not need to handle other HTTP methods or network protocols.
- Your payload should be an HTML <script> tag with the malicious JavaScript inside. We may use different payloads for grading.

Notes and hints:

- It is recommended that you write and test your code locally, to avoid dealing with Firewalls or other issues developing code in a remote environment.
- You may want to use a second browser to test your proxy, as proxying all your web traffic will flood your proxy with requests and prevent those web pages from functioning.

What to turn in: A file called **mitm_proxy.py** with the malicious proxy server.

Part 2. Packet Capture Analysis (15 points)

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”) from a real network. You will search for specific vulnerable behaviors and extract relevant details using the Wireshark network analyzer, which is available at https://www.wireshark.org.

Download the pcap from part1.pcap file from canvas, and examine 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. Each response should require at most 2–3 sentences.

Part 2 Questions

Concisely answer the questions below. Each response should require at most 2–3 sentences.

1. Multiple devices are connected to the local network. For each one, list their MAC address, IP address, and general purpose (personal computer, router, server, etc).
2. One of the clients connects to an FTP server during the trace.
    a. What is the IP address and DNS hostname of the server it connects to?
    b. Based on the packet capture, what's one major vulnerability of the FTP protocol?
    c. Name at least two other common network protocols that can be used in place of FTP to provide secure file transfer.
3. 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 domain name information from part (a)?
    c. At the very beginning of an HTTPS, the client and server agree on a cipher suite to use and exchange encryption keys. How many cipher suites does the client support?
    d. What cipher suite does the server choose for the connection?
    e. Who verified the server's identity by signing their certificate?
4. One of the clients makes a number of requests to Facebook.
    a. Facebook processes logins over HTTPS, so that user's passwords are encrypted. Despite this, what is insecure about the way that Facebook authenticates the user?
    b. How could this allow an attacker to impersonate a user on Facebook?
    c. How can this type of attack be prevented?
    d. What did the user do on Facebook?

What to turn in: A file called **part_2.txt** with your answers.

Note: This pcap dump was created many years ago. Any security vulnerabilities discovered here were patched long ago, and are no longer exploitable.

Submission Instructions

Please upload the following answers to Canvas in a single .zip file. The name of the zip file should include your identikey and your partner's identikey (if you have a partner). Only one of

you needs to upload the file, although it is recommended that you both upload it just in case. Your submission should contain these files:

Part 1: **mitm_proxy.py**
Part 2: **pcap_answers.txt**