Project 4: Network Security

This project is split into two parts, with the first checkpoint due on **Tuesday, April 22, 2025** at **6:00pm** and the second checkpoint due on **Thursday, May 1, 2025** at **6:00pm**. The first checkpoint is worth 20 points, and the second checkpoint is worth 80 points with 5 extra bonus points, applied based on the same policy as in MP1.

This is a personal project; you MUST work **on your own**. The code and other answers you submit MUST be entirely your own work. You MAY consult with other students about the conceptualization of the project and the meaning of the questions, but you MUST NOT look at any part of someone else's solution, including online ones (e.g., by past students on GitHub). You MAY consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Solutions MUST be submitted electronically in your Git repo, following the submission checklist given at the end of each checkpoint. You will lose points if in violation of submission requirements.

"The Internet was never designed to be secure."

– Dan Kaminsky

Introduction

This project will introduce you to common network protocols, the basics of interacting with the network from both offensive and defensive perspectives, and several local network attacks.

Objectives

After completing this MP, you will understand:

- How ARP binds IP addresses to link-layer addresses and how IP hijacking works.
- How DNS works and how to carry out a man-in-the-middle attack on DNS traffic.
- How TCP connections work and how to carry out a man-in-the-middle attack on TCP traffic.
- How an off-path TCP session spoofing attack works against a host with a predictable TCP sequence number generator.

Guidelines

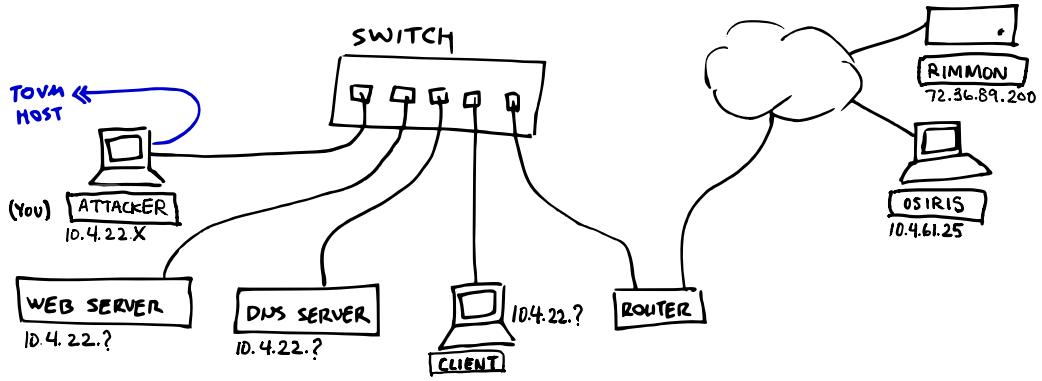
- You MUST work alone.
- In this handout, the term "`yournetid`" is a synonym to your actual NetID in lowercase (e.g. jdoe123). You MUST replace the term with your actual NetID when running commands and updating submission file content. Failing to do so may result in points deduction. This replacement does not apply to the term `netid.txt`.
- We have generated files for you to submit your answers in in your repository. You MUST submit your answers in the provided files; we will only grade what's there!
- You MUST push your code to the default branch. We will only grade the latest commit on the default branch before the deadline.
- Each submission file contains an example of the expected format. Failure to follow this format may result in 0 points for the section. You MAY delete the examples; they will be ignored (along with any other line starting with #) when grading.

Your VM

You are provided an Ubuntu VM (`https://uofi.box.com/s/rq9oft9ecwa3a8vpie8c8cu27xxpcoif`; this is the same VM as in the other MPs) which contains the network that you will be inspecting/attacking for this assignment. **Do not run attacks outside of `attacker` on the VM.** The VM has several hosts configured to provide you different vantage points on the internal network, which is detailed below. **Before you begin the assignment, turn on and initialize your VM by following the instructions provided in 4.1**.

Your Network

Your VM is configured with a virtual network consisting of several communicating virtual hosts:



- **webserver:** A web server serving HTML content over HTTP port 80.
- **dnsserver:** An authoritative DNS server for certain local domains over port 53.
- **attacker:** A host that you control that will perform the attacks for this assignment. You will be provided SSH access to `attacker`.
- **client:** A host that interacts with `dnsserver` and `webserver`. You will be provided SSH access to `client` to verify that your attacks are working for Checkpoint 4.2.
- **osiris:** A NetBSD 1.0 host on the external network 10.4.61.0/24 that is the target of the Mitnick attack.
- **rimmon:** An offline external network host at 72.36.89.200 whose IP is trusted by `osiris` and can connect to `osiris` via RSH.

Your VM is running Ubuntu 18.04 Bionic Beaver. The VM user `student` has access to the switch interface, which you can use to monitor all traffic through the virtual switch as you develop your solution.
The `webserver`, `dnsserver`, `attacker`, and `client` virtual hosts are Docker containers running on the VM. `attacker` and `client` can only be accessed through SSH from within the VM. While you are working on your solution, you will have access to the VM (including switch interface), `attacker`, and `client`. All code you turn in will be run from `attacker` and will not have access to the switch interface nor any of the other virtual hosts.

Your Tools

Your VM comes with the following network analysis tools already installed:

- **Wireshark/TShark 2.6.10:** Wireshark is a graphical tool for examining live network traffic and recorded network traces. TShark is a non-graphical version of Wireshark. Your TA(s) will demonstrate basic Wireshark usage in Discussion section. You will need to use Wireshark or a similar tool to complete parts of Checkpoint 1 and to debug your solutions for Checkpoint 2. You can find additional documentation at `https://wireshark.org`.
- **tcpdump 4.9.3:** `tcpdump` is a command line tool for recording and analyzing network traffic. This is a command line alternative to Wireshark and can be useful for debugging your solutions for Checkpoint 2.
- **dig 9.11.3:** `dig` is a command-line tool for querying DNS servers. Your TA(s) will demonstrate basic dig usage in Discussion section. You will need to use dig or a similar tool to complete parts of Checkpoint 1. You can view the manual for dig by typing `man dig` in your VM shell, or at `https://linux.die.net/man/1/dig`.
- **curl 7.58.0:** `curl` is a command-line tool for interacting with an HTTP server. Your TA(s) will demonstrate basic curl usage in Discussion section. You will need to use curl or a similar tool to complete parts of Checkpoint 1. You can view the manual for curl by typing `man curl` in your VM shell, or at `https://linux.die.net/man/1/curl`.
- **Nmap 7.60:** Nmap is a command-line tool for scanning hosts on a network. Your TA(s) will demonstrate basic Nmap usage in Discussion section. You can find additional documentation at `https://nmap.org`. Note Nmap is only installed in `attacker`. You need to SSH into `attacker` to run the tool.
- **Scapy 2.4.3:** Scapy is a packet manipulation library for Python. Your TA(s) will demonstrate basic Scapy usage in Discussion section. You will use Scapy to complete Checkpoint 1 *and all parts of Checkpoint 2.* You can find documentation at:
  - `https://scapy.readthedocs.io/en/latest/`: official Scapy documentation
  - `https://scapy.net/talks/scapy_pacsec05.pdf`: Scapy release presentation
  - `https://raw.githubusercontent.com/catalyst256/scapy-guide/master/ScapyGuide.pdf`: unofficial Scapy dummies guide
  - `https://github.com/secdev/scapy`: Scapy source code

Read this first

This project asks you to perform attacks, with our permission, against a target network that we are providing 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!** There are also severe legal consequences for unauthorized interception of network data under the Electronic Communications Privacy Act and other statutes. Per the course ethics policy, you are required to respect the privacy and property rights of others at all times, *or else you will fail the course.* Talk to the instructor for more information on the course ethics policy, law, and university policies.

4.1 Checkpoint 1 (20 points)

In this checkpoint, you will familiarize yourself with network analysis tools and implement a simple TCP SYN scanner using the Scapy framework.

BEFORE YOU BEGIN, make sure you initialize your VM. If you do not initialize your VM, then your answers for all sections of the assignment may be incorrect. We *will not* provide credit for errors of this nature. Please note: once you have initialized your VM, it will no longer have access to the public Internet, except `https://github.com/illinois-cs-461` and SSH communication. To initialize your VM please follow these steps

1. Start your VM and log in as user *student*. This can be the same course VM you used in the other MPs.

2. Open the Terminal application (control+alt+T), and then git clone your repository into the `~/Desktop/netsec/shared-with-attacker` directory by running:

   ```
   git clone https://github.com/illinois-cs-461/sp25_cs461_yournetid \
   ~/Desktop/netsec/shared-with-attacker/yournetid # accessing via HTTPS
   ```

   or

   make a folder mkdir suyashn2

   ```
   git clone git@github.com:illinois-cs-461/sp25_cs461_yournetid \
   ~/Desktop/netsec/shared-with-attacker/yournetid # accessing via SSH
   ```

   This directory is shared between the VM *student* user and `attacker` at `/root/shared-with-host`. It will be useful for synchronizing your code between `attacker`, which does not have a public Internet connection, and the VM *student* user, which can connect to `https://github.com/illinois-cs-461` so that you can upload your solution.

3. Merge the `NetSec` branch that contains the template submission files for the assignment into your default branch by running the following commands under your (just-cloned) git repository:

```
git pull
git merge origin/NetSec
git push origin
```

4. Run

   /suyashn2/sp25_cs_461_suyashn2/NetSec

   ```
   sudo ~/Desktop/netsec/shared-with-attacker/yournetid/NetSec/update_env.sh
   ```

   Note this command relies on content in the `update_files` directory. Unless requested by course staff, you MUST NOT modify any of the files in the directory as well as the `update_env.sh` script. or : sudo ./update_env.sh

5. Open `netid.txt` in the `NetSec` directory of your git repository and add your NetID with a newline character at the end to the file (i.e. press enter after typing your NetID). The content of the file should look like:

   ```
   yournetid
   ```

   Make sure to put your NetID correctly (all lowercase) as your network environment will be set up with randomized values based on the content of `netid.txt`. The autograder will assume your answers are based on the environment set up using the submitted `netid.txt`.

6. Run

   ```
   sudo netsec-setup ~/Desktop/netsec/shared-with-attacker/yournetid/NetSec/netid.txt
   ```

   This will initialize your network environment and will take a few minutes to run. *This will also disable all network traffic to the public Internet to prevent accidental network attacks on Internet hosts.* Unless requested by course staff, you MUST NOT further modify the VM firewall settings.

   Once it has completed, it will display information for connecting to the attacker host, which you will use for 4.1.3 and all of 4.2. If you forget the attacker IP address, you can always run `sudo netsec-info` to print out the IP address.

7. **For local VM setup**: Shut down the VM completely and create a snapshot, following the guide listed below. If you break the VM or the network setup, you can restore the VM to this snapshot. Note: restoring a snapshot reverses any change made to the VM after the snapshot was taken. Before you restore a snapshot, make sure you push your work to your git repository (or make backups through other means).

   - VirtualBox - `https://www.virtualbox.org/manual/ch01.html#snapshots`

   For campus VM farm VMs: Snapshotting is not available. Please contact the course staff ASAP if you break your campus VM farm VM. Despite not creating a snapshot, **you SHOULD still reboot your campus VM farm VM for the network changes to take effect.**

8. Start the VM if not started, and log in as user *student*.

After initializing the VM, you may begin the assignment.

4.1.1 Exploring Network Traffic (8 points)

As the *student* user in the VM, use Wireshark to observe traffic on the *switch* for at least 30 seconds. Recall that the *switch* interface shows you all traffic passing through the switch of your virtual network.

1. Identify all the active hosts on the local network, 10.4.22.0/24. (2 points)
   a) What are their IP addresses?
   b) What are their MAC addresses?

   What to submit: Submit a) `cp1.1.ip.txt` that contains the IP addresses and b) `cp1.1.mac.txt` that contains the MAC addresses. Write one address per line (address order between files does not matter). You MAY exclude the router in your answers. We will not check that.

2. What is the IP address of the gateway? (2 points)

   What to submit: Submit `cp1.1.gw.txt` that contains the IP address of the gateway.

3. One of the hosts performed port scanning, a technique used to find network hosts that have services listening on targeted ports.
   What is the IP address of the port scanner? (1 points)

   What to submit: Submit `cp1.1.portscan.txt` that contains the IP address of the port scanner.

4. The client performed DNS lookup, a technique used to retrieve information associated with domain names.
   What is the fully-qualified domain name (FQDN) of the DNS query? What IP address does the name resolve to? (2 points)

   What to submit: Submit `cp1.1.dns.txt` that contains the FQDN of the DNS query and the IP address that it resolves to, each on a separate line (order does not matter).

5. The client performed an HTTP request, a technique commonly used for retrieving HTML documents from Internet hosts.
   What URL is the client retrieving from the Web server? (1 points)

   What to submit: Submit `cp1.1.http.txt` that contains the full URL retrieved by the client.

4.1.2 Interacting with Network Hosts (4 points)

For this problem, you will log into `client` by running: `ssh root@<client-ip-from-part-1>` with the password *securepassword*. From the client, you will perform a basic DNS query and HTTP request. These commands will be helpful for Checkpoint 4.2.1 to verify the success of your attacks.

1. Use `dig` to query the DNS server you found in 4.1.1. What does `smtp.bankofbailey.com` resolve to? (2 points)

   What to submit: Submit `cp1.2.dig.txt` that contains the IP address included in the A record response.

2. Use `curl` to retrieve `http://www.bankofbailey.com` from the Web server you found in 4.1.1. Make sure you submit the entire *unmodified* HTTP response, not just the HTML portion. (2 points)

   What to submit: Submit `cp1.2.curl.txt` that contains the full unmodified HTTP response for `http://www.bankofbailey.com` as your solution.

4.1.3 Scanning Network Hosts (8 points)

For this problem, you will log into `attacker` by running: `ssh root@attacker` with the password *securepassword*. If you have forgotten the attacker IP address, you can run `sudo netsec-info`. From `attacker`, you will first use Nmap to identify open ports on the local network. You will then use Scapy to write your own Python program to scan individual IPs and report which ports are open.

1. Use Nmap (**only available on `attacker`**) to determine which hosts on the local network, 10.4.22.0/24, have an open TCP port between 1–1024 (inclusive), as determined by SYN scan (2 points). Note the identified hosts having open TCP ports can be different from problem 4.1.1.1.

   What to submit: Submit `cp1.3.nmap.txt` that contains `IPaddress,port` pairs, one per line, for open TCP ports.

2. Write a SYN scanner using Scapy. You will turn in your program, rather than the results of the scan. Your solution will be graded by running it on a slightly different network. You can test your solution by scanning some of the hosts identified in earlier parts of Checkpoint 1 (6 points). Note: due to Scapy quirks, your scanner does not need to be able to scan the attacker host itself.

   Specifications

   - Your program MUST use python3 and take two arguments: the interface to scan on, and the IP address to be scanned, (e.g. `python3 cp1.3.synscan.py eth0 10.4.61.200`).
   - Your program MUST use SYN scanning to detect open TCP ports between 1–1024, inclusive. It MUST send a SYN packet and record the port as open if a SYN+ACK is received; otherwise, the port is closed. If a SYN+ACK is received, the scanner MUST respond with a TCP RST packet in order to free up connection resources on the machine being scanned.
   - Your program MUST print out pairs of `IPaddress,port`, one per line for open TCP ports that are detected by SYN scanning. The order of `IPaddress,port` pairs does not matter, but each pair MUST be printed only once.

- Your program MUST finish scanning a single online IP within 5 minutes. You do not need to fulfill this time requirement for offline hosts. Note the default timeout for Scapy's `sr1` function should work fine.
- Your program MUST NOT hard-code the `attacker` source IP address.
- Your program MAY print comment lines as long as they are prefixed with "#" (the pound character, ASCII value 35). These lines will be ignored by the autograder.

What to submit: Submit `cp1.3.synscan.py` that prints out `IPaddress,port` pairs, one per line, for open TCP ports.

Checkpoint 1: Submission Checklist

Inside your repository, you will have auto-generated files named as below.
Make sure that your answers for all tasks up to this point are submitted in the following files by **Tuesday, April 22, 2025** at **6:00pm**.

NetID Information

`netid.txt`: a text file containing your NetID.

Example content format of `netid.txt`

```
yournetid
```

Solution Format

Example content format of `cp1.1.ip.txt`

```
1.2.3.4
127.0.0.1
```

Example content format of `cp1.1.mac.txt`

```
0f:0f:0f:0f:0f:0f
1e:1e:1e:1e:1e:1e
```

Example content format of `cp1.1.gw.txt`

```
8.8.8.8
```

Example content format of `cp1.1.portscan.txt`

```
192.168.1.254
```

Example content format of `cp1.1.dns.txt`

```
example.com.
1.2.3.4
```

Example content format of `cp1.1.http.txt`

```
http://example.com/path/
```

Example content format of `cp1.2.dig.txt`

```
1.2.3.4
```

Example content format of `cp1.2.curl.txt` (there may be more/fewer/different headers)

```
HTTP/1.1 200 OK
Content-Length: 45
Content-Type: text/html
Date: Mon, 25 Feb 2019 17:21:24 GMT
Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT

<html><body>Foo</body></html>
```

Example content format of `cp1.3.nmap.txt`

```
10.4.22.1,22
10.4.22.1,25
10.4.22.4,80
```

Example content format of `cp1.3.synscan.py`

```
from scapy.all import *

if __name__ == "__main__":
    print("10.4.61.4,994")
    print("#ignored comment line")
    print("10.4.61.4,25")
```

Example output format of `cp1.3.synscan.py`

```
10.4.61.4,994
#ignored comment line
10.4.61.4,25
```

List of files that must be submitted for Checkpoint 1

- `netid.txt`
- `cp1.1.mac.txt`
- `cp1.1.ip.txt`
- `cp1.1.gw.txt`
- `cp1.1.portscan.txt`
- `cp1.1.dns.txt`
- `cp1.1.http.txt`
- `cp1.2.dig.txt`
- `cp1.2.curl.txt`
- `cp1.3.nmap.txt`
- `cp1.3.synscan.py`

4.2 Checkpoint 2 (80 points + 5 bonus points)

In this checkpoint, you will implement the following network attacks:

- ARP spoofing
- HTTP man-in-the-middle injection
- Off-path TCP session spoofing

These attacks MUST be carried out from `attacker`. You will have access to the switch in order to observe the normal interaction between hosts and to observe your attempted attacks. You will also have access to `client` to verify your attacks end-to-end. However, the autograder will run your program from `attacker`, and your program will not have access to other hosts.

4.2.1 MITM attacks (55 points + 5 bonus points)

4.2.1.1 ARP Spoofing (25 points) (*Difficulty: Medium*)

For this problem, you will implement an ARP spoofing attack that will allow you to impersonate `webserver`, `dnsserver`, and `client` on the network. You will use this attack to cause all traffic between these hosts to go through your host, `attacker`. You will need to pass the traffic through to the intended host, in order to observe the rest of the conversation.

While you are developing your solution, you will have access to the switch, so you will be able to observe the normal interaction between the hosts. Your attack must make it appear, to each of the hosts, that they are communicating with their intended peer, as if the packets were routed by the peer directly. However, *your solution must work from the attacker's host,* by performing ARP spoofing and packet forwarding correctly.

Once you have traffic flowing between the Web server, DNS server, and client correctly through your host (attacker), you will need to answer the following questions:

- What host name is the client trying to resolve?
- What is the IPv4 address corresponding to that name, returned by the DNS server in response to the query?
- What is the value of the `session` cookie sent by the Web server to the client?
- What is the value of the Basic Auth password sent to the Web server?

Your solution to this problem is a Python program `cp2.1.passive.py`, using Scapy, that prints the answers to these questions. Note that your program will be graded using a slightly different environment, that is, the answers to the above will be different in the testing environment than what you see in your VM. Your solutions should generalize to different answers for the testing environment. For example, do not expect the length of the host name, session cookie, or Basic Auth password to be the same in the testing environment. Do not assume that cookie or Basic Auth headers will be present in all HTTP responses.

Specifications

- Your program MUST use python3 and take four arguments: the interface to scan on, the client IP , the DNS server IP, and the HTTP server IP. The code for argument parsing is included in the `cp2.1.passive.py` template file. Ex:
  `python3 cp2.1.passive.py -i eth0 --clientIP 10.4.22.3 --dnsIP 10.4.22.4 --httpIP 10.4.22.5`
- Your program MUST perform a passive man-in-the-middle attack between the client and DNS and HTTP server through ARP spoofing.
- Your program MUST complete ARP spoofing within three seconds of program startup.
- Your program MUST forward all packets. The intercepted hosts should not observe any disruption in communications. Your program MUST NOT modify `/proc/sys/net/ipv4/ip_forward`, which will configure the kernel to automatically forward all packets.
- Your program MUST print the answer to each question on a new line beginning with "$*$" (the asterisk character, ASCII value 42) followed by "`hostname:`" and the host name the client is resolving; "`hostaddr:`" and the IP address returned by the DNS server for the host name above; "`basicauth:`" and the client's Basic Auth password (decoded); "`cookie:`" and the server's session cookie for the user. Your program MUST print at least one line per observed answer for all observed answers, even if they are duplicates (e.g. if your program sees two client DNS queries, print the hostname line for both). The order of the answers does not matter.
- Upon observing relevant traffic, your program MUST print out the answer within three seconds.
- Your program MAY generate additional lines of output (e.g., for debugging); however, all such lines must be prefixed with "#" (the pound character, ASCII value 35).
- Upon receiving `KeyboardInterrupt`, your program MUST restore the correct ARP table mappings for the client and servers within three seconds and then terminate. Your program MUST continue to run (and print answers) before receiving `KeyboardInterrupt`.

What to submit Submit `cp2.1.passive.py` that performs ARP spoofing to execute a passive MITM attack and prints out the relevant sensitive information.

4.2.1.2 Script Injection (30 points + 5 bonus points) (*Difficulty: Hard*)

For this problem, you will use your man-in-the-middle capability to *rewrite* HTTP request responses sent from the Web server to the client to insert JavaScript into the HTML file served to the client. We have provided the following test HTML files and test cases for you to consider, and we recommend completing them in the order presented:

- `http://www.bankofbailey.com/index.html`—basic single-packet HTTP response (22 points)

- `http://www.bankofbailey.com/long.html`—long multiple-packet HTTP response (4 points)
- `curl <url> <same-url>`—multiple HTTP request/response on a single TCP connection (4 points)
- `http://www.bankofbailey.com/hard.html`—script injection beyond packet boundary (bonus, 5 points)

For the basic single-packet response, long multiple-packet response, and multiple HTTP request/response on a single TCP connection cases, you MAY assume that the </body> tag is present as a whole in a single packet and there is enough space in the </body> packet for the injection string. You MUST NOT assume either for the script injection beyond packet boundary case.

Specifications

- Your program MUST use python3 and take four arguments: the interface to scan on, the client IP address, the server IP address, and the script to inject. The code for argument parsing is included in the `cp2.1.http.py` template file. Ex:
  ```
  python3 cp2.1.http.py -i eth0 --clientIP 10.4.22.3 --serverIP 10.4.22.5
  --script 'alert("hi")'
  ```
- Your program MUST perform an active MITM attack between the client and server through ARP spoofing. Specifically, your program must insert the following string immediately before the closing </body> HTML tag in any request for a Web page from `www.bankofbailey.com`. The string you MUST insert is:

  ```
  <script>SRC</script>
  ```

  where `SRC` is the `--script` command line argument provided to your script. You MUST NOT insert any other character, including newline character before or after the `<script>SRC</script>` string.

  We recommend `alert("Successful Injection!");` as the `SRC` string. You can test your solution by running curl on `client`. Successfully intercepting the auto-generated traffic is a good start, but incomplete. HTTP clients (e.g. curl, Firefox) may not display signs of certain TCP errors, such as improper TCP connection termination, so pay close attention to Wireshark and compare unaltered TCP streams with your intercepted TCP streams.
- Your program MUST complete ARP spoofing within three seconds of program startup.
- Your program MUST allow any client `curl` command to the Web server to complete within three seconds.
- Your program MUST support multiple concurrent connections between the client and server.

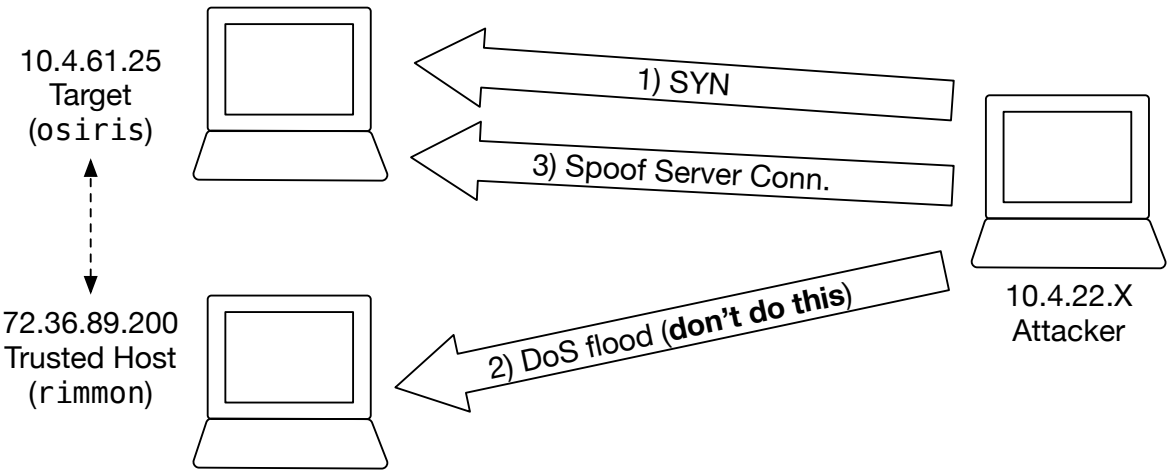
- Upon receiving `KeyboardInterrupt`, your program MUST restore the correct ARP table mappings for the client and server within three seconds and then terminate. Your program MUST continue to run (and inject the script element in HTTP responses) before receiving `KeyboardInterrupt`.

What to submit Submit `cp2.1.http.py` that performs ARP spoofing to execute a MITM attack that actively injects a `<script>` element into the HTTP response.

4.2.2 TCP Off-Path Session Spoofing (25 points) (*Difficulty: Hard*)

For this problem, you will implement part of Mitnick's[1] Christmas Day attack. You can learn more about this attack at:

- `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`
- `https://link.springer.com/content/pdf/10.1007%2F0-387-31163-7_8.pdf`
- `https://totseans.com/totse/en/hack/hack_attack/hacker03.html`



The part of the attack you will be implementing is the TCP session spoofing that is at the heart of the attack. In particular, you will be attacking a system using the same implementation of TCP that was used on the SPARCstation workstation attacked by Mitnick.[2] For this attack, you will need to execute a bash command *on the target machine* that adds the attacker IP address to the list of trusted hosts in `/root/.rhosts` on the target. To do so, you will need to spoof an RSH connection to the target machine 10.4.61.25, appearing to be from a trusted host 72.36.89.200. The target is configured to trust all connections from 72.36.89.200 (IP-based authentication), allowing anyone logged in to 72.36.89.200 to run a shell on the target without explicitly logging in with a user name and password.

[1] R.I.P. Kevin Mitnick (1963 – 2023).

[2] Tsutomu Shimomura's Sun workstation was running Solaris, which, at the time, used the TCP/IP stack derived from 4.2BSD. The system you will be attacking is running NetBSD 1.0, derived from the same source.

To successfully spoof a connection from the trusted host to the target, you will need to figure out how to predict the sequence number used for new connections to the server, so that when you initiate your connection, you will know what sequence number to use *without seeing the target's reply* (which is sent to the trusted host). For reference, the NetBSD 1.0 source code can be found here: `https://github.com/NetBSD/src/tree/netbsd_1_0/sys/netinet`.
While the actual attack carried out by Mitnick was more involved, for this assignment you:

1. SHOULD NOT carry out any reconnaissance on the target, because you already know the target trusts 72.36.89.200 and will accept a RSH connection without checking the user name and password;
2. SHOULD NOT perform a DoS attack to keep the trusted host 72.36.89.200 from responding with a TCP RST, because in your scenario, 72.36.89.200 will be offline; and
3. SHOULD NOT carry out the next stage of the attack, because your victory condition is adding the attacker's IP address to `/root/.rhosts` on the target.

Specifications

- Your program MUST use python3 and take three arguments: the interface to use, the target IP address, and the trusted host IP address.
  ex) `python3 cp2.2.mitnick.py eth0 10.4.61.25 72.36.89.200`
- You MUST execute the following command on the target as the root user:

  ```
  echo 'YOUR_ATTACKER_IP root' >> /root/.rhosts
  ```

  Note: you do not need to do a proper teardown of the RSH attack connection, you only need to be able to add your IP to the rhosts file.
- Your program MUST NOT hard-code the `attacker` source IP address.
- Your program does not need to work successfully 100% of the time. The autograder will re-run your attack for 5 times (max 10 seconds each) and check to see if it succeeds at least once.

The RSH protocol is detailed here (Section "Service Request Protocol"):

- `https://www.ibm.com/docs/en/aix/7.2?topic=r-rshd-daemon#rshd__title__5`

We HIGHLY RECOMMEND you to read this documentation. Based on our experiences, it will save you a couple of hours of debugging time at the cost of a mere 5-minute read. You can also attempt to RSH to the target from the netsec VM user (even though the host will receive a permission denied error) as a reference network trace for your attack. You can verify your attack's success by connecting to the target via RSH by running

```
rsh 10.4.61.25 uname -ns
```

which should print `NetBSD osiris.shimomura`.

After your attack has succeeded, and you wish to remove your IP address from `/root/.rhosts` to test your attack again, you can run:

```
rsh 10.4.61.25 "head -n2 /root/.rhosts > tmp && mv tmp /root/.rhosts"
```

If your program does not reset TCP connections/SYNs, after some time the target my stop responding to incoming TCP SYN packets. If you encounter this issue, run `sudo netsec-restart` to restart the NetSec environment. This will reset the target and resolve the issue in most cases.

It is also known that the MP network at times chokes and does not forward spoofed packets correctly, which would make it impossible to perform the attack. If you find yourself in this situation, first backup your files, then reboot your VM. This would resolve the problem in most cases. If the problem still persists, contact the course staff ASAP.

What to submit Submit `cp2.2.mitnick.py` that performs the Mitnick attack as specified.

Checkpoint 2: Submission Checklist

Make sure that your answers for all tasks up to this point are submitted in the following files by **Thursday, May 1, 2025** at **6:00pm**.

NetID Information

`netid.txt`: a text file containing your NetID.

Example content format of `netid.txt`

```
yournetid
```

Solution Format

Example output format of `cp2.1.passive.py`:

```
# this is a comment that will be ignored by the grader
*hostname:somehost.com.
*hostaddr:1.2.3.4
*basicauth:password
*cookie:Name=Value
```

Example content format of `cp2.1.http.py`:

```
from scapy.all import *

if __name__ == "__main__":
    # perform active HTTP attack
```

Example content format of `cp2.2.mitnick.py`

```
from scapy.all import *

if __name__ == "__main__":
    # perform mitnick attack
```

List of files that must be submitted for Checkpoint 2

- `netid.txt`
- `cp2.1.passive.py`
- `cp2.1.http.py`
- `cp2.2.mitnick.py`