Project 4: Network Security

This project is split into two parts, with the first checkpoint due on **Wednesday, November 2** at **6:00pm** and the second checkpoint due on **Friday, November 11** at **6:00pm**. The first checkpoint is worth 2% of your total grade, and the second checkpoint is worth 10%. We strongly recommend that you get started early. Each semester everyone will be given ONE late extension that allows you to turn in up to one checkpoint up to 24 hours after the due date. Extensions are not automatic. So, if you want to use your late extension, you MUST send an e-mail to **ece422-staff@illinois.edu** before the deadline. Late work will not be accepted after 24 hours past the due date.

This is a group project; you SHOULD work in **teams of two** and if you are in teams of two, you MUST submit one project per team. If two different sets of answers are submitted, the one with lower total grade will be accepted. Please find a partner as soon as possible. If you have trouble forming a team, post on Piazza's partner search forum. Note that Aircrack-ng Suite, one of the tools required for Checkpoint 2, may not be compatible with your hardware. Please read the **Checkpoint 2 Setup** in advance. Build your team such that at least one member of the team can run the required tools.

The code and other answers your group submits must be entirely your own work, and you are bound by the Student Code. 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 or collaborate with anyone outside your group. 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 any one of the group member's SVN repository, following the filename and solution formats specified under the submission checklist given at the end of each checkpoint.

"You can't defend. You can't prevent. The only thing you can do is detect and respond."

– Bruce Schneier

Introduction

This project will introduce you to common network protocols, the basics behind analyzing network traces from both offensive and defensive perspectives, and several local network attacks.

Objectives

- Gain exposure to core network protocols and concepts.
- Understand offensive techniques used to attack local network traffic.
- Learn to apply manual and automated traffic analysis to detect security problems.

Guidelines

- You SHOULD work in a group of 2.
- Your answers may or may not be the same as your classmates'.
- All the necessary files to start the project are given under a folder called "mp4" in your SVN repository: `https://subversion.ews.illinois.edu/svn/fa16-ece422/NETID/mp4`
- We generated files for you to submit your answers in. You MUST submit your answers in the provided files; we will only grade what's there!
- Each submission file contains an example of expected format. Failure to follow this format may result in 1 point deduction for the section. You MAY delete the examples; they will be ignored (along with any other line starting with #) when grading.

Required Tools

- Wireshark 32 bit*
- Aircrack-ng Suite
- nmap
- Python 2.7
- dpkt (Python package)

* We strongly recommend using 32 bit version of Wireshark for Checkpoint 2. Bugs within 64 bit version may prevent you from completing the task.

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.* See "Ethics, Law, and University Policies" on the course website.

4.1 Checkpoint 1 (20 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 trace from a sample network we set up for this assignment. You will search for specific details using Wireshark network packet analyzer (`https://www.wireshark.org`).

4.1.1 Exploring Network Traces (10 points)

Examine the first network trace, `4.1.1.pcap`, using Wireshark.

Provide concise answers to the following questions.

1. Identify all the hosts on the local network. (3 points)
   a) What are their IP addresses?
   b) What are their MAC addresses? Assume that each IP address is mapped to one MAC address.

   What to submit Submit a) `4.1.1.1_ip.txt` that contains the IP addresses and b) `4.1.1.1_mac.txt` that contains the MAC addresses. Write one address per line.

2. How many unique TCP conversations, also known as TCP sessions, are there? (1 points)

   What to submit Submit `4.1.1.2.txt` that contains the number of unique TCP conversations. Write the number only.

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

   What to submit Submit `4.1.1.3.txt` that contains the IP address of the gateway.

4. Retrieve and submit the content of the files downloaded from the FTP servers. (3 points)

   What to submit Submit a) `4.1.1.4_active.txt` that contains the content of the file downloaded from the *active* FTP server and b) `4.1.1.4_passive.txt` that contains the content of the file downloaded from the *passive* FTP server.

5. 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 `4.1.1.5.txt` that contains the IP address of the port scanner.

Check your answer formats with the examples provided under *Checkpoint 1: Submission Checklist*.

4.1.2 HTTPS Traffic (10 points)

Examine the second network trace, `4.1.2.pcap`, using Wireshark.

Provide concise answers to the following questions.

1. In what year was this traffic captured? (1 points)

   What to submit Submit `4.1.2.1.txt` that contains the year. Write the number only.

2. What is the hostname (or Fully Qualified Domain Name) of the server that the client connected to? (1 points)

   What to submit Submit `4.1.2.2.txt` that contains the hostname of the website.

3. During TLS handshakes, the client(s) provided a list of supported cipher suites.
   List the supported cipher suites from one of the clients. (1 points)

   What to submit Submit `4.1.2.3.txt` that contains the list of the supported cipher suites. Write one cipher suite per line. The name and hex Cipher ID of each cipher suite MUST be as shown in Wireshark.

4. Which cipher suite did the server choose for the connection? (1 points)

   What to submit Submit `4.1.2.4.txt` that contains the cipher suite chosen by the server. The name and hex Cipher ID of the cipher suite MUST be as shown in Wireshark.

5. A user of the client searched a person's name on the website.
   What is the first name of this person? (2 points)

   What to submit Submit `4.1.2.5.txt` that contains only the first name of the person searched on the website.

6. The user sent a message to the person found in 4.1.2.5.
   What is the *body* of the message? (3 points)

   What to submit Submit `4.1.2.6.txt` that contains the *body* of the message. The message should be 112 characters long.

7. Export and submit the user's cookie used in sending the message. (1 points)

   What to submit Submit `4.1.2.7.txt` that contains the user's cookie. Your answer should include only the content, not the "Cookie: " prefix.

Check your answer formats with the examples provided under *Checkpoint 1: Submission Checklist*.

Checkpoint 1: Submission Checklist

Inside your mp4 SVN 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 **Wednesday, November 2** at **6:00pm**.

SVN Repository

`https://subversion.ews.illinois.edu/svn/fa16-ece422/NETID/mp4`

Team Members

`partners.txt` : a text file containing NETIDs of both members, one NETID per line.
Place the NETID of the student making the submission in the first line.

example content of `partners.txt`

```
your_netid
partner_netid
```

Solution Format

example content of `4.1.1.1_ip.txt`

```
1.2.3.4
127.0.0.1
```

example content of `4.1.1.1_mac.txt`

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

example content of `4.1.1.2.txt`

```
461
```

example content of `4.1.1.3.txt`

```
8.8.8.8
```

example content of `4.1.1.4_active.txt`

```
content_from_active_FTP!@($!@:+_
```

example content of `4.1.1.4_passive.txt`

```
content_from_passive_FTP!@($!@:+_
```

example content of `4.1.1.5.txt`

```
192.168.1.254
```

example content of `4.1.2.1.txt`

```
1970
```

example content of `4.1.2.2.txt`

```
www.example.com
```

```
blog.example.com
```

example content of `4.1.2.3.txt`

```
TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
```

example content of `4.1.2.4.txt`

```
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
```

example content of `4.1.2.5.txt`

```
john
```

example content of `4.1.2.6.txt`

```
========security is kewl; blah blah blah @#$!)$(@ this message is in a si
ngle line.============================
```

example content of `4.1.2.7.txt`

```
__utma=44258684.1258198627.1456687180.1457965376.1458137415.4; __utmc=442
58684; __utmz=44258684.1457965376.3.3.utmcsr=isss.illinois.edu|utmccn=(re
ferral)|utmcmd=referral|utmcct=/
```

List of files that must be submitted for Checkpoint 1

- `partners.txt`
- `4.1.1.1_mac.txt`
- `4.1.1.1_ip.txt`
- `4.1.1.2.txt`
- `4.1.1.3.txt`
- `4.1.1.4_active.txt`
- `4.1.1.4_passive.txt`
- `4.1.1.5.txt`
- `4.1.2.1.txt`
- `4.1.2.2.txt`
- `4.1.2.3.txt`
- `4.1.2.4.txt`
- `4.1.2.5.txt`
- `4.1.2.6.txt`
- `4.1.2.7.txt`

4.2 Checkpoint 2 (100 points)

4.2.1 Network Attacks (70 points)

In this part of the project, you will experiment with network attacks by:

- cracking password for a WiFi network protected with WEP (Wired Equivalent Privacy), a security protocol designed for WiFi networks
- analyzing network traffic
- obtaining a victim's credentials

At Siebel Center Room 1129 (CS460/ECE419 Lab), you will find a WEP-encrypted WiFi network named `cs461fa16`. We've created this network specifically for you to attack, and you have permission to do so. On the network, you will find a server and client(s), all wirelessly connected.

Your goal is to obtain a client's secret retreived from the server.

You SHOULD use suggested tools to complete the tasks. If you need to use anything else, you MUST ask a TA for permission.

If you run into issues due to having incompatible wireless adapter, you MAY rent a USB wireless adapter during TA's office hours. Rent period is 24 hours. If returning early or during weekends, please arrange a meeting with a TA by sending an email to **ece422-staff@illinois.edu**.

Setup

- **For Mac (OS X) Users**
  We **STRONGLY recommend you to find a partner who has a non-Apple laptop**.

- **For Others**
  We **STRONGLY recommend using Kali Linux Live booted from a USB drive**.
  1. Download Kali Linux Light 32 bit:
     `http://cdimage.kali.org/kali-2016.2/kali-linux-light-2016.2-i386.iso`
  2. Create a Kali Live Bootable USB drive:
     `http://docs.kali.org/downloading/kali-linux-live-usb-install`
  3. Add persistence to your drive to allow storing your data and configuration:
     `http://docs.kali.org/downloading/kali-linux-live-usb-persistence`
  4. Boot from your USB drive, then choose "Live USB Persistence" from the menu.
  5. Install Wireshark:

     ```
     apt-get update
     apt-get install wireshark
     ```

  Warning: Use the Kali USB drive on your own computers only. DO NOT use the drive on school computers, such as EWS or any other lab machines.

4.2.1.1 WEP cracking (15 points) (*Difficulty: Medium*)

First, you will need to crack the WEP encryption key for the network using Aircrack-ng Suite. Aircrack-ng website provides many helpful tutorials (e.g. `http://www.aircrack-ng.org/doku.php?id=simple_wep_crack`). You SHOULD go through the tutorials to understand the purpose of each tool and learn how to use them.

As you will learn from the tutorial, WEP cracking process usually involves performing certain attacks to generate traffic so that data can be collected faster. For this project, however, you MUST NOT use those attacks as they severly disrupt the network.

We've made sure that there's sufficient traffic generated on the network to allow successful WEP cracking within a reasonable amount of time (< 1 hour). If the process is taking too long, please notify a TA.

What to submit Submit `4.2.1.1.txt` that contains the WEP key in *ASCII characters*.

4.2.1.2 Client IP address (5 points) (*Difficulty: Medium*)

Examine the network traffic with Wireshark and identify the IP address(es) of the client(s).
Note: Other students (and yourself) on the network are not considered as clients.

What to submit Submit `4.2.1.2.txt` that contains the IP address(es). Write one address per line.

4.2.1.3 Server IP address (5 points) (*Difficulty: Medium*)

Examine the network traffic with Wireshark and identify the IP address of the server.
Note: We consider a host as a server only if it provides services to other hosts. Other students (and yourself) on the network are not considered as servers.

What to submit Submit `4.2.1.3.txt` that contains one IP address.

4.2.1.4 Services (12 points) (*Difficulty: Medium*)

Find the services (below port 4096) provided by the server you identified in 4.2.1.3.
In addition to examining the network traffic with Wireshark, consider using nmap, a powerful tool for probing remote hosts, as well as interacting with the server yourself.

What to submit Submit `4.2.1.4.txt` that contains names of application layer protocols used by the services (in standard acronyms). Write one protocol per line.

4.2.1.5 Server's secret (5 points) (*Difficulty: Easy*)

Find the *server*'s secret file using the services you found in 4.2.1.4.

What to submit Submit the secret file renamed as `4.2.1.5.txt`.

4.2.1.6 Client's credentials (20 points) (*Difficulty: Hard*)

The client(s) retrieve secret messages from the server using credentials that expire in an hour. Obtain the credentials. You may find multiple pairs; pick any one of them.
Hint: You may want to read up on the HTTP Basic Authentication method, which is specified in RFC 2617.
Note: We strongly recommend using 32 bit version of Wireshark for this part. Bugs within 64 bit version may prevent you from completing the task.

What to submit Submit `4.2.1.6.txt` that contains the username in first line and password in second. You MUST submit only one pair.

4.2.1.7 Client's secret message (6 points) (*Difficulty: Easy*)

Obtain the client's secret message using the credentials you found in 4.2.1.6. Each retrieval attempt generates a different secret message; pick any one of them.

What to submit Submit `4.2.1.7.txt` that contains the secret message. When obtaining the secret message, you MUST use the same credentials you submit for 4.2.1.6. You MUST submit only one.

4.2.1.8 Jail time (2 points) (*Difficulty: Easy*)

What is the maximum number of years in jail that you could face under 18 USC § 2511 for intercepting traffic on an encrypted WiFi network without permission?

What to submit Submit `4.2.1.8.txt` that contains the maximum number of years in jail. Write the number only.

4.2.2 Anomaly Detection (30 points) (*Difficulty: Medium*)

In 4.1.1, you manually explored a network trace and detected a port scanning activity. Now, you will programmatically analyze trace data to detect such activity.

Port scanning can be used offensively to locate vulnerable systems in preparation for an attack, or defensively for research or network administration. In one port scan technique, known as a `SYN` scan, the scanner sends TCP `SYN` packets (the first step in the TCP handshake) and watches for hosts that respond with `SYN+ACK` packets (the second 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 network trace, you can identify source addresses that may be attempting a port scan.

Your task is to develop a Python program that analyzes a PCAP file in order to detect possible `SYN` scans. You MUST use `dpkt` Python library for packet manipulation and dissection.
For more information about `dpkt`:

- PyDoc documentation - `pydoc dpkt`
- official website - `https://github.com/kbandla/dpkt`

You may also find this tutorial helpful:
`https://jon.oberheide.org/blog/2008/10/15/dpkt-tutorial-2-parsing-a-pcap-file`

Specifications

- Your program MUST take one argument, the name of the PCAP file to be analyzed:
  ex) `python2.7 4.2.2.py capture.pcap`
- Your program MUST print out 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. For the purpose of this assignment, this rule applies even if the number of packets is very small. For example, following cases are all considered as attacks:

```
SYN=4 ACK+SYN=1
SYN=4 ACK+SYN=0
SYN=1 ACK+SYN=0
```

- Each IP address MUST be printed only once.
- Your program MUST silently ignore packets that are malformed or that are not using Ethernet, IP, and TCP.

Example output: A sample PCAP file captured from a real network can be downloaded at:
`https://subversion.ews.illinois.edu/svn/fa16-ece422/_shared/mp4/lbl-internal.20041004-1305.port002.dump.anon.pcap`
(Source: `ftp://ftp.bro-ids.org/enterprise-traces/hdr-traces05`)
For this input, your program's output MUST be these lines, in any order:

```
128.3.23.2
128.3.23.5
128.3.23.117
128.3.23.158
128.3.164.248
128.3.164.249
```

What to submit Submit `4.2.2.py`, a Python program that accomplishes the task specified above. You SHOULD assume that the grader uses `dpkt` 1.8. You MAY use standard Python system libraries, but your program SHOULD otherwise be self-contained. We will grade your detector using a variety of different PCAP files.

Checkpoint 2: Submission Checklist

Inside your mp4 SVN 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 **Friday, November 11** at **6:00pm**.

SVN Repository

`https://subversion.ews.illinois.edu/svn/fa16-ece422/NETID/mp4`

Team Members

`partners.txt` : a text file containing NETIDs of both members, one NETID per line.
Place the NETID of the student making the submission in the first line.

example content of `partners.txt`

```
your_netid
partner_netid
```

Solution Format

example content of `4.2.1.1.txt`

```
cs46!
```

example content of `4.2.1.2.txt`

```
1.2.3.4
127.0.0.1
8.8.8.8
```

example content of `4.2.1.3.txt`

```
1.2.3.4
```

example content of `4.2.1.4.txt`

```
ftp
telnet
smtp
pop3
```

example content of `4.2.1.6.txt`

```
username1
p@ssw0rd123
```

example content of `4.2.1.7.txt`

```
1suPerSECretMesSageNKJn23kjsdjnK+Lskvnlksdf10dm23/sdfinvkwer2ThisShouldBe
InASingleLine23+4/
```

example content of `4.2.1.8.txt`

```
10
```

List of files that must be submitted for Checkpoint 2

- `partners.txt`
- `4.2.1.1.txt`
- `4.2.1.2.txt`
- `4.2.1.3.txt`
- `4.2.1.4.txt`
- `4.2.1.5.txt`
- `4.2.1.6.txt`
- `4.2.1.7.txt`
- `4.2.1.8.txt`
- `4.2.2.py`