Network Traffic Analysis[1]

1 Overview

One of the most important tasks you will perform while doing work related to network security is looking through packet traces. A packet trace is simply a recording of the packets that pass through some point on the network. Typically the packets are recorded at the lowest level possible, so the packets include link-layer headers, higher-layer headers (e.g., IP, TCP, HTTP), and application data.

In this project, you will be analyzing packet traces to identify attacks and other security-related network phenomena. The goal of the project is to cement a more solid understanding of network protocols and attacks and to help you gain familiarity with the standard tools used to view and analyze them.

This assignment is due by no later than 11:59pm on Wednesday, October 13th, 2021. See Section 4 for submission details.

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 may not look at any part of someone else's solution or collaborate with anyone else. You may consult published references, provided that you appropriately cite them (e.g., with program comments), as you would in an academic paper.

Objectives

- Gain exposure to core network protocols and concepts.
- Learn to apply manual and automated traffic analysis to detect security problems.

[1] Bits and pieces of this assignment have been borrowed from various sources, including a project designed by Paxson and Wagner for their computer security class, Matthew Green's security and privacy course, and the University of Michigan.

2 Getting started

Begin by downloading the traces you will need to analyze. You can find the traces on Brightspace. The packets within each trace are stored in the libpcap file format,[2] a simple and widely used standard.

Once you have obtained your traces, you may use any tools you like to analyze them and come up with your answers to the questions which appear later in this document. However, you must adequately justify and describe your approach to answering each of the problems. In particular, for Part 1, points will be deducted from correct answers if your approach does not scale well or is too generic (i.e., your approach requires a lot of manual inspection despite the potential for some automation, refined filtering, etc.). Note however that none of the questions require doing anything complicated like decoding compressed data or handling encryption and that scripting will mostly be helpful for counting or filtering tasks. Please see each part for detailed instructions on our expectations.

The most useful tool for completing the project is **Wireshark**, an open-source program for graphically viewing and analyzing packet traces: http://www.wireshark.org/. Wireshark (formerly known as Ethereal) is the most popular tool of this type and runs on all major operating systems. Another useful tool included with Wireshark is **tshark** – Wireshark's textual command-line counterpart.[3] Wireshark allows you to use a GUI to manually explore a trace, so Wireshark is probably more convenient for interactive use, but tshark will be essential if you want to analyze the trace with a script. Another tool similar to tshark is tcpdump, which is older and more well-known. All of these tools can be used in two modes: live capture (that is, recording) of packets from the network interface of the machine running the program, and reading a trace from a file. For this project, you will only need to use them in the latter mode. (Note that live capture often requires administrator access due to its security and privacy implications.)

We recommend you begin the project by loading the traces into Wireshark and spending a little time looking through them and familiarizing yourself with Wireshark's features. Below are some more tips to get you started. One of Wireshark's most important capabilities is its filtering system. Filtering lets you display only a subset of the packets. This is very helpful when dealing with large traces, to let you focus on a small subset of the packets. You can configure a filter by typing an expression into the box at the top of the window, to display only the packets relevant to what you are investigating. Wireshark provides a special language for these expressions, which you will probably want to learn (at least to some extent).

- Try clicking on the "Filter:" button to see a list of examples of filtering expressions. Select each of the filters listed in the popup box one by one and take a look at the expression that appears for each in the bottom box.
- Expressions can specify a protocol (e.g., `http`). You can also filter on values in headers (e.g., `ip.src==1.2.3.4` or `ip.src==1.2.0.0/16` or `tcp.port==80`). You can combine filters using logical expressions (e.g., `http || telnet` or `http && ip.src==1.2.0.0/16`).
- For a complete list of the supported protocols, click the "Expression..." button in the main window. The vast majority of these won't be useful on this project, but the list will give you an idea of how comprehensive the tool is.

[2] If you're curious, you can find a description of the format at the following address, although you will not likely want to write any code that reads the file directly: http://wiki.wireshark.org/Development/LibpcapFileFormat.

[3] For those of you using Wireshark for OS X, notice that tshark is one of the applications included with the Wireshark installation disk image in the "Utilities" folder. You may want to copy it from the disk image to your filesystem.

- To get an overview of the protocols that are used in the trace, you might try “Statistics” then “Protocol Hierarchy.” You can right-click on an individual protocol, then click on “Apply as Filter” and “Selected” to add a filter that selects just that protocol.
- To get a list of the endpoints that appear in the trace, you can click on “Statistics” then “Endpoints”, then select either the “IPv4”, “TCP”, or “UDP” tab at the top. You can right-click on individual end host addresses to add a filter that selects just packets associated with that endpoint.
- Another useful feature is Wireshark’s ability to reassemble TCP streams. Try right clicking on a TCP packet and selecting “follow TCP stream”. You can use this feature to read the contents (HTML and the like) of a web page someone loaded over HTTP, for example.
- You will probably want to maximize the Wireshark so that it uses the entire display, to maximize the number of packets you can view at a time.

To learn more about using Wireshark and tshark, see http://www.wireshark.org/docs/, where you can find everything from manuals to video lectures.

3 Questions

Part 1. Automating the Exploration of 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.

Download the pcap from https://www.cs.purdue.edu/homes/clg/CS526/projects/project2_part1.pcap (or from Brightspace), and examine it using Wireshark. Familiarize yourself with Wireshark's features and try exploring the various options for filtering and for reconstructing data streams. Remember, for this section your approach should be as automated and scalable as possible (ie you should not be scrolling through the entire trace by hand to find the solution).

1. **HTTP Sessions**
   For this problem, find all web servers that were *successfully* visited in the trace (that is, contacted via HTTP). Include any servers that engaged in a valid instance of the HTTP protocol, even if the status code returned was, for example, 404 rather than 200.[4] Submit a list of their IP addresses with your answer. Please note that you should not try to identify HTTPS traffic.[5]

2. **Directory Traversal**
   One simple way people attempt to exploit a web server is by making requests for files outside the normal directories it serves using pathnames with sequences like "../../../". (Of course, a reasonably well-implemented web server will not fall for tricks like this.) Find a host that appears to be attempting this type of attack and submit its IP address with your answer.

3. **Password Guessing**
   If you've ever looked through the logs of an SSH server, you've likely seen attempts to login through brute force guessing of usernames and passwords. Of course, the same attack is possible for any type of protocol with password authentication. There is one host that attempted such an attack against a password protected FTP server. Find that host and include the IP address of the attacker with your answer.

4. **Unencrypted Usernames and Passwords**
   Next, find an unencrypted username and password. Note that we are interested in a real username and password, so failed login attempts don't count. Examples of some protocols that can send usernames and passwords without encryption are Telnet, FTP, HTTP, and POP3. List the username and password with your answer.

[4] There is one unusual case that requires a special explanation. One client made an HTTP request that was cut off in the trace file for the project. That is, one or more of the first packets of the request are missing from the trace; you can tell by looking at the TCP flags. In packet that remains, you can only see the end of the requested URL and the headers which follow it. The server responded to this request with "302 Object moved". You should consider this a valid HTTP session and include that server with your answer.

[5] In fact, this is not even possible. HTTPS traffic will just appear as SSL/TLS traffic from the perspective of a trace file. Unless you were a party in the original traffic and thus have the key used, it will simply appear as encrypted TCP traffic. In reality, you might be able to do traffic analysis or something similar to probabilistically identify it as HTTPS traffic, but that is well beyond the scope of this course.

5. **Service Versions**
   Finding hosts running specific versions of servers is an important step in exploiting them; in general, older versions will have more vulnerabilities. For this problem, find the host running the oldest version of Apache. (Apache is the most widely used web server on the Internet.) Don't count "Apache-Coyote" as "Apache"; also, ignore any servers that don't specify their version. Submit that host's IP address with your answer.

6. **DNS and Source Port Randomization**
   Most clients now select a random UDP source port when making DNS queries to help prevent an attack due to Kaminsky.[6] For this problem, look for clients which do not use a random source port. There are exactly two such DNS resolvers (not including MDNS[7]). With your answer to this question, submit the IP addresses of the two DNS resolvers (not counting MDNS) that use the same source port for all the DNS queries they make (and make more than 1 query).

7. **TCP Sequence Numbers**
   It is important that the first sequence number chosen by hosts forming a TCP connection be unpredictable. If an adversary can guess the initial sequence number (ISN), they can easily mount TCP session hijacking attacks. In this particular trace, only a few of the TCP implementations appear to use fully random ISNs.[8] You may want to disable Wireshark's relative sequence number feature while working on this question.[9] Find the IP addresses of the two TCP endpoints that participate in 5 connections or more and that provide the broadest 32-bit coverage in their ISNs[10]. Submit theses IP addresses with your answer.

8. **Traceroute Scanning**
   Traceroute is a utility for finding the addresses of the routers along the IP route between the host it is being run on and an arbitrary destination. You can read about the utility here: http://en.wikipedia.org/wiki/Traceroute.

   Attackers sometimes use traceroute to find out about a victim's network infrastructure (routers and possibly firewalls). Identify the host that is running traceroute for detecting routers on a path. Submit the IP address of the host running traceroute and the IP address of the destination of the traceroute path with your answer.

9. **Cross-Site Scripting**
   In class, we discussed three types of cross-site scripting (XSS) attacks: *reflected* XSS, *stored* XSS, and *DOM-based* XSS. Recall that reflected XSS involves an attacker sending the victim a URL that contains a script inside the URL itself, so that the server that processes the URL includes the script within the body of the page it returns. Find evidence of reflected XSS. Specifically, submit the IP address of the server that has a reflected cross-site scripting vulnerability that was exploited in the trace. (To our knowledge, there is only one such server in the trace.)

[6] I suggest googling for information about this attack and studying it.

[7] MDNS refers to DNS traffic that is sent using "IP multicast", a type of transmission similar to IP broadcast. Rather than using the standard UDP port for DNS, which is 53, MDNS uses port 5353, and Wireshark displays the corresponding protocol as MDNS. So for this problem, ignore such traffic.

[8] FYI, it turns out that the TCP implementations in the trace that don't use fully random ISNs still likely have considerable protection against sequence-number guessing. If you're curious, see http://www.ietf.org/rfc/rfc1948.txt for the standardized way that TCPs are supposed to safely pick their sequence numbers.

[9] See http://wiki.wireshark.org/TCP_Relative_Sequence_Numbers for details.

[10] For the purposes of this project, we define broadest coverage to be the endpoints that utilize the biggest range of values for their initial sequence numbers.

What to submit Submit a PDF named `purdueid_answers.pdf` containing your answers.

Part 2. Interpreting Network Traces

Now that you have explored how to use Wireshark to learn about network traffic and how to automate the discovery of useful information, we will work on not only extracting the relevant data but also interpreting the implications of various network behaviors.

Download the pcap for this part from https://www.cs.purdue.edu/homes/clg/CS526/projects/project2_part2.pcap (or from Brightspace).

Concisely answer the questions below. Each response should require at most 2–3 sentences. For this section, you may obtain the answers in any way that you choose, and we will not be putting an emphasis on scalability of solutions (though scalability and automation using tools like Wireshark will probably make this much easier for you).

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 (e.g., a large corporation, an ISP backbone, etc.)? Point to evidence from the trace that supports this.
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?
   (c) Based on the packet capture, what's one major vulnerability of the FTP protocol?
   (d) Name at least two network protocols that can be used in place of FTP to provide secure file transfer.
4. 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?
   (b) How would this let an attacker impersonate the user on Facebook?
   (c) How can users protect themselves against this type of attack?
   (d) What did the user do while on the Facebook site?

What to submit Submit a PDF named *purdueid*_answers.pdf containing your answers.

Part 3. Anomaly Detection

In Parts 1 and 2, you manually explored network traces with the help of Wireshark. Now, you will programmatically analyze a pcap file to detect suspicious behavior. Specifically, you will be attempting to identify port scanning.

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.

Your task is to develop a Python program that analyzes a pcap file in order to detect possible `SYN` scans. To do this, you will use `dpkt`, a library for packet manipulation and dissection. It is available in most package repositories. You can find more information about `dpkt` at https://github.com/kbandla/dpkt and view documentation by running `pydoc dpkt`, `pydoc dpkt.ip`, etc.; there's also a helpful tutorial here: https://jon.oberheide.org/blog/2008/10/15/dpkt-tutorial-2-parsing-a-pcap-file/.

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

```
  python2.7 detector.py capture.pcap
```

The output (printed to stdout) should be 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. Your program should silently ignore packets that are malformed or that are not using Ethernet, IP, and TCP.

A large sample pcap file captured from a real network can be downloaded at https://www.cs.purdue.edu/homes/clg/CS526/projects/project2_part3.pcap (or from Brightspace). (You can examine the packets manually by opening this file in Wireshark.) For this input, your program's output should 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 a Python program that accomplishes the task specified above, as a file named `detector.py`. You should assume that `dpkt` 1.9.7.2 and `scapy` 2.4.5 are available, and 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.

4 Deliverables, deadline, and other information

Please the project online via Brightspace.[11] You can submit the components as many times as you like (n.b., subsequent submissions overwrite previous ones). The deadline is firm so please do not wait until the deadline to upload your assignment. You must upload a gzipped tarball (`.tar.gz`) named `project2.`*`purdueid`*`.tar.gz` to Brightspace. Refer to the Project 1 specifications for instructions on creating your tarball.

For this project you will submit the following files:

Filename	**Description**
`purdueid``_answers.pdf`	A PDF file containing your responses to the project questions in Parts 1 and 2.
`question_`*`n`*`.sh/py/pl`	When applicable, include any scripts used to help answer project questions in Parts 1 and 2.
`detector.py`	Your Python program for `SYN` scan detection.
other files	Include any additional files or programs created to support your efforts.

Table 1: Your submission tarball should include the following files. Please name it `project2.`*`purdueid`*`.tar.gz`

The files *`purdueid`*`_answers.pdf` should contain your responses for each project question. As discussed in Section 2, each answer must include an adequate justification and description of your approach. You may use any word processor or typesetting environment of your choosing in creating this file, but you must submit a valid PDF file. Please note: we will deduct points from your project for excessive violation of reasonable organization, good grammar, or proper spelling.

[11] See the *Project Submission* link on Brightspace.