Project 2

Due Thursday, April 8, 11:59pm

One of the most common tasks you are likely to encounter 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 a packet trace 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.

Getting Started

Begin by downloading the trace you will need to analyze. You can find your trace at the URL
`http://inst.eecs.berkeley.edu/~cs161/sp10-proj2/cs161-XX.trace`
where "cs161-XX" is your course account (e.g., cs161-kj or cs161-du). Be sure to download the trace that matches your account. Each one is different, so if you download the one for someone else's account, all your answers will be wrong! The packets within each trace are stored in the libpcap file format[1], a simple and widely used standard.

Once you have obtained your trace, you may use any tools you like to analyze it and come up with your answers to the questions which appear later in this document. You will not need to submit any code with your answers. However, in some cases you may find it useful to write one or more simple scripts[2] to help find the answers. Many of the questions can be answered by manually inspecting the trace with existing tools.

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. You can find it installed on the instructional machines[3], or install it on your own machine. Another useful tool included with Wireshark (and also installed on the instructional machines) is tshark—Wireshark's textual command-line counterpart. Wireshark allows you to use a GUI to manually explore a trace, so Wireshark is probably more

[1] 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` .

[2] None of the questions require doing anything complicated like decoding compressed data or handling encryption. The only reason one might write a script would be for simple counting or filtering tasks.

[3] If you want to use Wireshark on the instructional machines, we recommend that you use one of the SunRay machines in 271, 273, or 277 Soda. Those machines are connected directly to Solaris SPARC servers, and so should give good responsiveness and performance. Wireshark uses a graphical interface, so we don't recommend that you try to use Wireshark by logging into instructional machines remotely (e.g., using SSH): it probably won't work well, if it works at all. **Warning: Most instructional labs, including 271, 273, and 277 are closed from 4pm Friday March 19 to 8am Monday March 29**; see `http://inst.eecs.berkeley.edu/notices.html` .

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 (but not installed on the instructional machines). 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/privacy implications.)

We recommend you begin the project by loading the trace into Wireshark and spending a little time looking through it and familiarizing yourself with Wireshark's features. Here 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.
- 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.

Collaboration

You must do this project entirely on your own. You may not collaborate with other students. You may share general information on how to use Wireshark with other students if it is not specific to the questions on this project, but you must not share tips, advice, hints, etc. on how to solve any of the questions on this project or how to use Wireshark to solve these questions.

You must write up your solutions entirely on your own. You must never read or copy the solutions of other students, and you must not share your own solutions—not even partial solutions—with other students. If you have any questions, please contact the instructors.

Submissions and Grading

Like Project 1, all submissions for this project will be electronic. For each of the questions in the following section, create a (7-bit ASCII) text file named `q1.txt`, `q2.txt`, ..., `q10.txt`. Please also create a file named `login.txt` that contains the name of your class account (e.g., "cs161-xy"). You do not need to (and should not) include your account name or anything other than the answer in the other files. To submit, put all these files into a directory you've created for this project, and then run `submit proj2`.

All questions for this project will be graded completely automatically, and you will see your score as soon as you submit the files. If your score is not as high as you would like, you can keep making submissions up until the deadline. If you make a submission after the deadline, we will grade that submission and dock points accordingly based on how late it is. To prevent people from determining the answers by trying all possibilities, your score will only be printed for the first 5 submissions you make. After those, you may continue to resubmit, but your score will not be printed. We HIGHLY recommend that you only submit once you believe you have answered all the questions correctly. You only will receive 5 submissions with auto-grading; there will be NO exceptions.

Most questions ask for a list of values. For these, you may separate your answers with commas and/or whitespace, and the ordering of the values does not matter. Partial credit will be awarded for these questions according to the number of correct values submitted, but each incorrect value cancels out a correct value. So, for example, if the correct answer to a question is the list "A B C D", a submission of "A B C E" would receive $\frac{3-1}{4} = \frac{1}{2}$ of the points for that question.

Questions

1. (9 pts.) 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 (in `q1.txt`) as your answer. Please note that you should not try to identify HTTPS traffic[5].

[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 in your answer. For more details see the April 5th newsgroup posts about this case.

[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.

2. **(9 pts.) 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.

3. **(10 pts.) 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 submit the IP address of the attacker.

4. **(10 pts.) 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 as your answer.

5. **(10 pts.) 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.

6. **(13 pts.) DNS and Source Port Randomization**

   Recall that most clients now select a random UDP source port when making DNS queries to help prevent the Kaminsky attack. For this problem, look for clients which do *not* use a random source port. There are exactly two such DNS resolvers (not including MDNS[6]). As 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. **(13 pts.) TCP Sequence Numbers**

   As explained a few weeks ago in lecture, 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.[7] You may want to disable Wireshark's relative sequence number feature while working on this question.[8] 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. Submit a list of the two IP addresses.

[6] 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.

[7] FYI, it turns out that the TCP implemenations 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.

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

8. **(13 pts.) 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.

9. **(13 pts.) 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.)

10. **(0 pts.) Feedback - Optional**
    Submit a text file, `q10.txt`, with any feedback you may have about this project. What was the hardest part of this project in terms of understanding? In terms of effort? Any feedback you'd like to provide on the class (e.g., what's the single thing we could do to most improve the class?). We appreciate any comments you may have. Your answers will not affect your grade.