National Research
Tomsk
State
University

Laboratory 1 – Communication Protocols: Hands on

Passive Testing Techniques for Communication Protocols

February 13, 2016

The goal of this laboratory is to remind you how protocols work and how to interact with them (besides completing the environment setup that will be useful for our future laboratories), either using known tools or implementing your own. The final goal is not to test this services actively, but, to have tools to debug your passive testers, after all, it is convenient to check if your passive testers are performing the correct tests. Therefore we need a way generate the data.

Most of the times, different Operating Systems (OS) or tools within them can provide similar results, for instance, host −t A kitidis.tsu.ru (*nix command) and nslookup −querytype=A kitidis.tsu.ru (Windows command) both do a DNS query to the default DNS sever. As stated before in the course introductory lecture, you are free to use the OS or tool of your choice. However, instructions, hints, commands, libraries, and etc., will be provided only for the a standard Linux environment. In short, you're on your own if you decide to choose your own OS or tools, sorry :)

Submissions: Submissions should be done in a single compressed file (zip, or gz, including flavors of .tar.gz, and .tgz) of a directory containing all the necessary files. Regardless of how jealous you are about sharing your code, always include the code of your assignations. Include all your documents, executable files, and a Makefile to compile them. If you feel the need of putting a README.txt file, to clarify what are the contents of your submission feel free to do so. All submissions should be done via The Moodle Cloud Platform, https://pttcc.moodlecloud.com/ Make sure you are able to login and that you assign the course to yourself.

1 Using existing tools to test / interact with network protocol implementations

1.1 Data Visualization

The first thing that you need to know when interacting network protocols is that you need a tool to retrieve the network packets and to visualize their contents. Perhaps the world's most famous tool is the Wireshark application (that you might already know). Proceed with the following instructions, replying to the questions and adding screenshots in a file (for later submission in PDF format):

1. Please, launch the Wireshark application.

2. From the available devices to capture (capture->Interfaces... ctrl + i) choose the one with an IP address assigned (if you are on the provided VM, probably it'll be enp0s3), and click on the start button.

3. If by this time no packets are present, it means you either have no applications using the network, you are not connected to the network, or you didn't choose the correct device to capture packets from :)

4. Bring up a browser (Firefox installed by default in VM image), open your different web pages (e.g. http://kitidis.tsu.ru/, http://fctomtomsk.ru/). Stop the capture (we have enough for now).

5. Inspect several packets with Wireshark. When your local IP (the one chosen in step 2) address is the one sending packets, to different destination IP addresses, on the Ethernet protocol, the destination address changes? Why?

6. Discover one of your favorite sites IP address by applying the filter DNS to wireshark and look at the DNS queries / responses. Obtain the IP address. What is the Transport Layer Protocol being used? What is the destination port for the DNS queries (or source port for the responses)?

7. Put the following filter on Wireshark: http && ip.addr == IP_ADDRRESS, where IP_ADDRRESS corresponds to the IP address of one of your favorite sites, for instance, http && ip.addr == 92.63.68.61. Choose on packet where the protocol is HTTP and the source IP is your local iP address, what is the destination IP (in the Internet Protocol Layer), and what is the destination port? Provide a screenshot of the HTTP response of the associated packet (search for the response of this packet).

8. Leave Wireshark open, we will be using it soon. Open a terminal (konsole for the VM image).

9. While Wireshark is useful for hosts with Graphical User Interface (GUI), in many occasions you would want to capture the traces in systems without GUI. Execute the following command in the terminal: sudo tcpdump −i <INTERFACE>, where <INTERFACE> is the interface obtained in step 2 (for instance, $ sudo tcpdump -i enp0s3). Go to your browser and go to another site (to generate some traffic). After capturing some traffic, press ctrl + c to end the execution of tcpdump. Please observe the output. Note that not much information is displayed by tcpdump as it is, however, asking tcpdump for more details will provide much more information. Re-do the process, with the command

sudo tcpdump −i <INTERFACE> −XX −vvv −e, provide a screenshot. Do you see any difference?

10. With tcpdump you can also use filters. Filters can be complex, even specifying certain bits of the packets, including for instance, specific TCP flags. To enable a filter, put the filter expression at the end of the tcpdump command. For example, sudo tcpdump −i <INTERFACE> −XX −vvv −e port 80. Generate again some traffic and take a Screenshot of this.

11. You might want to use Wireshark to visualize data from a capture with tcpdump. Execute the command: sudo tcpdump −i <INTERFACE> −w <FILENAME>, Where <FILENAME> is the name of the file you want to save (Wireshark has also the capability to save the capture). Quit tcpdump (ctrl + c). Go to Wireshark and open the saved file (ctrl + o). Attach the captured packet to your final submission. Enjoy! :)

At this point you should have experienced how to visualize any value of the packet protocols using well-known tools. Get familiar with such tools, they will be useful for debugging.

1.2 Protocol Implementations interactions

The second thing you need to know about interacting with network protocols is that mostly you will be interacting with protocol server implementations. There is a large number of clients that you might use to interact with them, commonly you can use the destined clients as web browsers, ssh clients, etc., to force certain behaviors. Nevertheless, sometimes you might want to provide specific values or observe specific responses not provided by the standard clients. Luckily, there are many tools that let you interact in a less restricted manner with such protocols, in this part, we will learn about some of them. Sadly, If you want to test client implementations, you will either have to obtain a server implementation that has a very good debug mode, obtain an open source implementation and modify the code, to test the clients, or you will have to do your own implementation. Nonetheless, in the next part we explore writing code for network applications.

tools to test if a server is alive (ping) or the route it takes to reach another host (traceroute) are known and might be interesting for network troubleshooing, we won't focus on such tools, unless they are useful for our purposes such as the host command to do DNS queries.

Proceed with the following instructions, replying to the questions and adding screenshots in a file (for later submission in PDF format):

1. The telnet command can be useful to connect to TCP-based services and send messages as desired. Let's start easy. execute the command telnet kitidis .tsu.ru 80. Inside the connection, issue the HTTP message GET / HTTP/1.0 followed by two enters. See, how the server replies with the code OK, and how the running web

application miserably fails, even revealing potentially sensitive information of the location of certain files of the configuration. Take a screenshot and report it to the site maintainer :3

2. After being disappointed that you could not get your desired HTML, re-connect using telnet. This time issue a different message, GET / HTTP/1.1 followed by an enter, then, issue the message, Host: kitidis .tsu.ru, followed by two enters. it should look similar to this:

   ```
   Connected to kitidis.tsu.ru.
   Escape character is '^]'.
   GET / HTTP/1.1
   Host: kitidis.tsu.ru
   Connection: close

   HTTP/1.1 200 OK
   ...(much more text, including the HTML index file, in fact)
   ```

   What is the value of the HTTP header X-Powered-By:?

3. Open a terminal, and type host −t MX <DOMAIN>, where <DOMAIN> is the domain name of your personal email address, for instance sibmail.com. The previous command will obtain the server name that you need to deliver email (or several, choose the response with the smallest number at the left of the name). If out of curiosity you want to know the real IP of that server, you can issue the command host −t A <SERVER>, where <SERVER> is the obtained reply from the first DNS query.

4. We will be using telnet to interact with your mail server supplying custom data, using the Simple Mail Transfer Protocol (SMTP). Have a prepared list made with the following text:

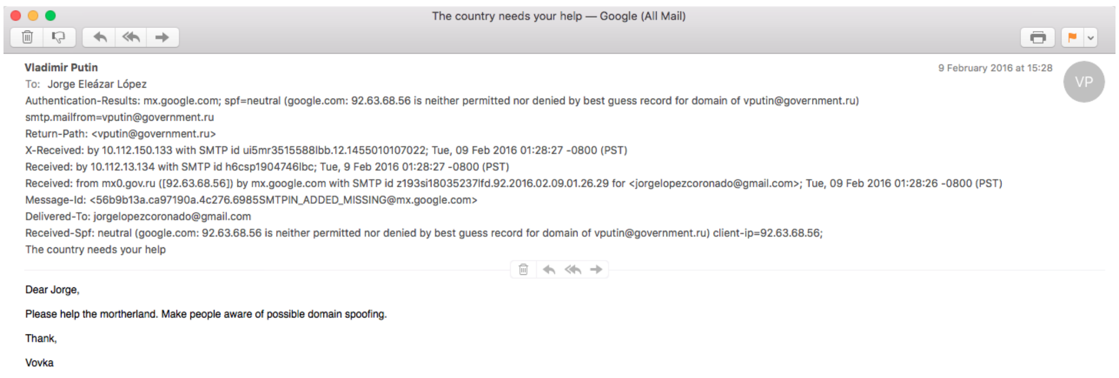
   - HELO mx0.gov.ru.
   - MAIL FROM:<vputin@government.ru>
   - RCPT TO:<<EMAIL>>, where <EMAIL> is your email address
   - DATA
   - From: Vladimir Putin <vputin@government.ru>
   - To: <NAME> <<EMAIL>>, where <EMAIL> is your email address, and <NAME> is your name
   - DATE: <DATE>, where <DATE> is the output of the date command on a terminal
   - Subject: <SUBJECT>, where <SUBJECT> is the subject of the email (you will make a screenshot out of this, just for you to consider your level of politeness :p).

- [enter] <CONTENT> [enter][enter].[enter], where <CONTENT> is the content of the mail

Execute the command telnet <SERVER> 25, where <SERVER> is the obtained reply from the first DNS query (e.g. sibmail.com). The mail server will greet you. Start typing (copy-pasting) the prepared list of commands. Finish each line with an enter (in exception in the content part that to start it requires two as well as to finish), and wait for the server's replies. Provide two screenshots. The telnet session and the received mail (check in your spam if you don't have it and the server accepts it, if you don't have it, don't provide the screenshot, but a justification). The session will go somehow similar to this:

```
Connected to gmail-smtp-in.l.google.com.
Escape character is '^]'.
220 mx.google.com ESMTP z193si18035237lfd.92 - gsmtp
HELO mx0.gov.ru
250 mx.google.com at your service
MAIL FROM:<vputin@government.ru>
250 2.1.0 OK z193si18035237lfd.92 - gsmtp
RCPT TO:<jorgelopezcoronado@gmail.com>
250 2.1.5 OK z193si18035237lfd.92 - gsmtp
DATA
354  Go ahead z193si18035237lfd.92 - gsmtp
From: Vladimir Putin <vputin@government.ru>
To: Jorge Lopez <jorgelopezcoronado@gmail.com>
date: Tue Feb  9 15:27:28 NOVT 2016
Subject: The country needs your help

Dear Jorge,

Please help the mortherland. Make people aware of possible domain
spoofing.

Thank,

Vovka

.
```

Here's the resulting email:



```
Vladimir Putin                                                                 9 February 2016 at 15:28
To:  Jorge Eleázar López
Authentication-Results: mx.google.com; spf=neutral (google.com: 92.63.68.56 is neither permitted nor denied by best guess record for domain of vputin@government.ru)
smtp.mailfrom=vputin@government.ru
Return-Path: <vputin@government.ru>
X-Received: by 10.112.150.133 with SMTP id ui5mr3515588lbb.12.1455010107022; Tue, 09 Feb 2016 01:28:27 -0800 (PST)
Received: by 10.112.13.134 with SMTP id h6csp1904746lbc; Tue, 9 Feb 2016 01:28:27 -0800 (PST)
Received: from mx0.gov.ru ([92.63.68.56]) by mx.google.com with SMTP id z193si18035237lfd.92.2016.02.09.01.26.29 for <jorgelopezcoronado@gmail.com>; Tue, 09 Feb 2016 01:28:26 -0800 (PST)
Message-Id: <56b9b13a.ca97190a.4c276.6985SMTPIN_ADDED_MISSING@mx.google.com>
Delivered-To: jorgelopezcoronado@gmail.com
Received-Spf: neutral (google.com: 92.63.68.56 is neither permitted nor denied by best guess record for domain of vputin@government.ru) client-ip=92.63.68.56;
The country needs your help

Dear Jorge,

Please help the mortherland. Make people aware of possible domain spoofing.

Thank,

Vovka
```

5. Finally, nowadays most of the implementations have adopted the equivalent SSL/TLS encrypted protocols, for example, HTTPS. OpenSSL let's you interact with such services. Execute the command: `openssl s_client −connect www.google.ru:443` (Note that the port for HTTP is 80 and for HTTP is 443) do the same command as in 2, changing the Host value for www.google.ru. What is the value of the HTTP header X-Frame-Options?

At this point you should know how to interact with some network protocols. Mostly those who are text based, and use the TCP transport. Sometimes, it is not possible to issue the protocol message as simple as we did. Or UDP-based services which are connectionless will be hard to make such interactions (although netcat or nc is able to do it partially). Given those reasons, we explore how to create our own very simple network based applications in the following section.

2 Implementing Network Protocols 101

In this last part of the laboratory, we will implement a client and a server for a network protocol.

The Very Simple Network Protocol (VSNP) is not standardized (yet). It is a protocol that we will be using throughout our entire course and there is not RFC to specify its functionality other than the following text.

The VSNP client issues a 2-byte ID to the VSNP server. The ID can be generated randomly, increasingly, or with any other strategy. The protocol does not enforce a special meaning of the ID. The server, upon receiving a request (composed only by the ID), replies the 2-byte ID, and a 2-byte pseudo-random number. If the request ID is an even number, the server must reply with an odd number. If the request ID is an odd number, the server must reply with an even number. The random numbers replied should have an equal distribution. Both the ID and the response number should be treated as unsigned values. When the client receives the generated number, it can do anything with the value. The VSNP server works on the port TCP 1010 by standard, but, it can operate in other TCP ports.

For this laboratory, implement a VSMP client that displays the received value, and a VSNP server that displays the IP address, source port, request ID, and response number.

The client should connect to a VSNP server specified in the arguments with the -s flag. Also an optional -p file for the port, if -p is not provided, the port is assumed to be 1010. The server takes the same arguments, for this laboratory most probably you will listen in the server just in the IPv4 localhost address(127.0.0.1). Both programs should include normal failure messages, for instance if no server replies on the specified IP address/port, then an error message should be provided, etc.

For this purpose you are given a compressed archive placed on the MoodleCloud and the local FTP, VSNP.tgz. The contents of the file are as follows:

- Makefile: this is the file that allows you to easily compile your source files. Do not modify it unless you know what you're doing
- NetClient.c: This file contains the implementation of auxiliary functions for the client connection. Do not modify it unless you know what you're doing
- NetClient.h: This file contains the header functions of auxiliary functions for the client connection. Do not modify it unless you know what you're doing. Read from this file the available functions you have as a client (mainly connectToServer)
- NetHost.c: This file contains the implementation of auxiliary functions for both, client and server implementations. Do not modify it unless you know what you're doing
- NetHost.h: This file contains the header functions of auxiliary functions for both, client and server implementations. Do not modify it unless you know what you're doing. Read the available functions (mainly readFromHost and writeToHost)
- NetServer.c: This file contains the implementation of auxiliary functions for the server connection. Do not modify it unless you know what you're doing
- NetServer.h: This file contains the header functions of auxiliary functions for the client connection. Do not modify it unless you know what you're doing. Read from this file the available functions you have as a server (mainly serverClients, not necessary, an example is given in the main file how to use it).
- VSNPClient: Client executable compiled for the VM
- VSNPClientMain.c: The main client implementation should be done here if you are using C (I will provide better help if you code in C), in here, you use the connecToServer function in main and the readFromHost and writeToHost functions
- VSNPClientMain.cpp: Same as VSNPClientMain.c, but imported headers for c++
- VSNPServer: Server executable compiled for the VM
- VSNPServerMain.c: The main server implementation should be done here if you are using C (I will provide better help if you code in C), in here, you implement properly the function serviceToClient, using the proper readFromHost and writeToHost functions

- VSNPServerMain.cpp: Same as VSNPServerMain.c, but imported headers for c++

It might look scary and big, but, I coded all for you just to use the functions stated above and implement the functionality of VSNP. The instructions how to run and to compile:

- To compile the server with the c file run make server.c
- To compile the server with the c++ file run make server.cpp
- To compile the client with the c file run make client.c
- To compile the client with the c++ file run make client.cpp
- To clean the generated executables run make clean
- To execute the server run sudo ./VSNPServer, stop execution with ctrl + c
- To execute the client run VSNPClient VSNPClient

Remember: the port and host should be taken from the arguments in your final implementation. Also, a Sublime text editor is already installed in your VM in case you don't feel comfortable with text-based editors.

That is all for this lab, I hope you enjoy it!