CS60, Final Projects

Sergey Bratus, Spring 2017

Date due: Code for your project will be due on Wednesday May 31. Your project report will be due on Saturday June 3.

Status of this document: This document will be continually updated with technical details on how to develop and test your projects.

Platform: Depending on the project your solution should work in the class VM (*cs60base* or *cs60mini*) or on the Raspberry Pi you will be issued. You are allowed to use *any* programming languages and packet-handling libraries you like, including but not limited to Python and Scapy. Correctly working C/C++ projects will receive 10% extra credit (or more if your code impresses us).

Your program will run as root, and can be split into as many parts as you like, including shell scripts, libraries, etc.

Submission: Your code must be submitted through the CS department LabGit system at `https://gitlab.cs.dartmouth.edu/`. Create a directory called `project` in your `cs60`. Include your report as PDF file or a plain text file (or use Markdown). Please do **not** submit reports as MS Office files.

Teamwork: You can choose to work alone or in a team of up to 3 people. Work by teams will be expected to have bigger scope and implement more features.

Project A: "NAT Gateway"

Build a program that runs on Linux or MacOS and implements the functionality of a network's default gateway and NAT. Your program should correctly perform packet forwarding and address translation with the OS native forwarding and NAT functionality disabled, and should allow hosts using your computer as a gateway to browse the web and ping normally.

In the following, *your host* means either your Linux VM (if you run MacOS), your actual computer (if you run Linux), or the issued Raspberry Pi. Remember that even if you develop on your Linux machine, you will need to make sure your program builds and runs in the VM or on the Pi, so that we can test it.

- Your program will take a network configuration (IP address, netmask, default gateway to the Internet) sufficient for establishing Internet connectivity for your host. You will use this information to configure your host's network interface(s). On Linux, assume that automatic configuration tools such as Network Manager are not present or have been shut down (as in the class VM), and disable them if necessary.
- Your kernel's built-in forwarding and NAT functionality will be disabled. You are allowed to use IPtables and PF for other purposes such as blocking certain kinds of packets.
- Your program will use raw sockets of `AF_PACKET` type[1] or the *libpcap* library to receive Ethernet frames. Use an `AF_PACKET` raw socket to emit Ethernet frames.
- You will configure other computers to use your host as the default gateway. You can use a DHCP server and example configurations in the *dhcp/* subdirectory of the class directory, or you can configure these other computers manually. If you are using a wireless LAN, you should disable the DHCP server on your wireless access point.[2]
- Your program must do what a default gateway does: rewrite headers of Ethernet frames that reach it so that their IP payloads can reach their targets. It should also do what a NAT box does: rewrite the IP headers of outgoing packets from local machines behind the NAT, so that the packets appear to come from the (routable) NAT box itself, and the responses can be directed to it and would reach it. Then the NAT box rewrites these responses so that they reach the local machines that originally initiated the respective connections or requests.

[1] See "man 7 packet". Use the `htons(ETH_P_ALL)` protocol as the third argument of the socket() call.

[2] E.g.., `https://www.voipmechanic.com/turning-off-dhcp-wireless-router.htm`

Equipment for this project: Ethernet switch (if your computers have "wired" Ethernet interfaces); a Wi-Fi router (if your computers have no Ethernet interfaces and you cannot use a USB or a Thunderbolt Ethernet adapter that would give you a "wired" interface). Raspberry Pi (optionally).

Project B: "VPN Gateway and client"

You will implement a VPN server on a Raspberry Pi host and VPN clients for it for your computer(s) or VM(s). After your client establishes a connection to your VPN server, your computer's network services such as the SSH daemon or a web server should be reachable from any computer running your VPN client.

For example, suppose your VPN server Pi has the address of 129.170.x.y, and you have two laptops (or VMs) on which the VPN client is running. The first client to establish connection with the server gets the virtual address 10.10.0.2, the second gets 10.10.0.5. Then "ping 10.10.0.2" should work on the second laptop, and "ping 10.10.0.5" should work on the first, and so should TCP connections between the two (for applications that are listening; use Netcat to test the simplest kinds of connections). The server itself should be reachable from both laptops as "10.10.0.1".

Your solution will include:

- Configuration and use of virtual network interfaces TUN (on Linux) or UTUN (on MacOS[3]).
- Programs that read (receive) and write (send) IP packets on these interfaces, wrap them in new IP headers, and send them to the VPN server, and handle the server's responses accordingly.
- The server program that operates as a regular TCP/IP server, listening on a fixed port known to the clients. Since the clients initiate the connections to the server, they will be able to connect with each other—with the server forwarding their packets for each other—even if they operate from behind NATs and would not normally be able to connect to each other.
- Optionally, you will implement authentication between clients and the server, so that only authorized clients can connect. Also optionally, you will implement encryption to hide the contents of the packets exchanged via the server.

Your solution will operate on the IP (Layer 3) layer. Optionally, you can implement a solution that operates on Layer 2; for that, you will need to use the TAP driver rather than TUN.[4]

Equipment for this project: A computer or VM with a static IP, for the server. This can be a Raspberry Pi or a cloud VPS. Several computers or VMs capable of running VPN clients.

Project C. Custom packet-level project.

You may propose your own project that involves manipulation of packets at Layer 2 and Layer 3 of the OSI model. I will need to approve your project and clarify the deliverables. Please do not start working on a custom project until you have cleared it with me.

Terms and conditions: You are allowed to use any externals materials, printed or electronic. You are allowed to discuss any issues and technical tricks, but the code you submit must be your own: you are not allowed to copy solutions from other students. Abide by the Honor Code; if in doubt, ask.

[3] See `http://www.cs.dartmouth.edu/~sergey/netreads/utun/`

[4] See explanations in `pytap.py` of my `http://netfluke.org` sample code regarding loading TUN/TAP drivers on MacOS since Yosemite.