ETH zürich

Networked Systems
ETH Zürich — seit 2015

Communication Networks

Prof. Laurent Vanbever

Solution: Exercise 13 – Applications

13.1 DNS

For this and the following task you will have to change the DNS resolver your VM is using. To do that execute the following command (this change will be lost on reboot):

```
resolvectl dns enp1s0 192.168.122.98
```

Now answer the following questions using `dig(1)` on your VM:

a) What is the IP address of the web server at `www.lan`? Which group hosts the web server?

b) What is the name of the default gateway? The command to inspect the routing table is `ip route`.

c) If you resolve the name from the previous subtask, what do you get? Is that to be expected?

d) Resolve the name `self.lan`. What IP address is it pointing to? Can you explain what is happening here?

Solution:

a)
```
; <<>> DiG 9.18.1-1ubuntu1.1-Ubuntu <<>> www.lan
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 29243
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.lan.                       IN      A

;; ANSWER SECTION:
www.lan.            3600    IN      CNAME   group-51.lan.
group-51.lan.               3600    IN      A       192.168.122.98

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Fri Jun 03 12:23:47 CEST 2022
;; MSG SIZE  rcvd: 75
```

Here you can see the `www.lan` has a `CNAME` pointing to `group-51.lan` and then shows the IP address of `group-51.lan`. Note that in this case the DNS server is running on the same host as the web server.

Solution:

b) The output of `ip route` shows:

```
default via 192.168.122.1 dev enp1s0 proto dhcp metric 100
169.254.0.0/16 dev enp1s0 scope link metric 1000
192.168.122.0/24 dev enp1s0 proto kernel scope link src 192.168.122.98 metric 100
```

The first line tells us that the default route goes over 192.168.122.1. To find the name of an IP address, we use `PTR` records. To do that we use the `-x` flag with `dig(1)`. This returns the following response:

```
; <<>> DiG 9.18.1-1ubuntu1.1-Ubuntu <<>> -x 192.168.122.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 33818
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;1.122.168.192.in-addr.arpa.        IN      PTR

;; ANSWER SECTION:
1.122.168.192.in-addr.arpa. 3600 IN PTR     gateway.lan.

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Fri Jun 03 12:28:00 CEST 2022
;; MSG SIZE  rcvd: 80
```

The name of the default gateway appears to be `gateway.lan`.

c) To resolve that, we use again `dig(1)`, e.g. `dig gateway.lan` which returns:

```
; <<>> DiG 9.18.1-1ubuntu1.1-Ubuntu <<>> gateway.lan
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5505
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;gateway.lan.                       IN      A

;; ANSWER SECTION:
gateway.lan.                3600    IN      A       192.168.122.98

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Fri Jun 03 12:29:21 CEST 2022
;; MSG SIZE  rcvd: 56
```

The name does not point back to the same IP. This is suspicious at least. Maybe the DNS server is trying to trick people to use `192.168.122.98` as default gateway to inspect the traffic and forward it to the real gateway.

d) Resolving it `dig self.lan` returns your IP address for that name. Of course that is not normal. The reason why that works is that the DNS server provides a different view depending on the source IP of the request. Similar techniques could be used to perform load balancing, i.e. you return a different answer based on the location of the source IP.

13.2 HTTP

We know interrogate a web server on three levels: Once directly using `telnet(1)`, once using a command line utility called `curl` and finally via the web browser running on your local system.

a) What port is HTTP using? You can consult the file `/etc/services` on your VM. Using this file can you work out what port `ntp` is using? What protocol is it using?

b) Now connect to the web server using `telnet(1)`. Every time you press return, `telnet(1)` sends a carriage return (*CR*) and a line feed (*LF*) to the remote host. Type your request for the root directory like this:

```
GET / HTTP/1.0
```

and press enter twice to first end the current line and then to send an empty line.

c) Can you learn what software the web server is using?

d) Now use the `curl(1)` tool to request `www.lan` like this:

```
curl www.lan
```

Is the result different from the first task? Can you work out why?

e) Using `ssh(1)` you can connect the web browser on your computer to the web server like this:

```
ssh -L 127.0.0.1:8080:<web-server-name>:<www-port> -p <2000+X> student@duvel.ethz.ch
```

X your group number. Now navigate your browser to `127.0.0.1:8080` and look at the shown web page.

f) Your friend says there is a secret web server called `www2.lan` running on the same host as `www.lan`, but when you check with `dig(1)` there is no IP for `www2.lan`. Can you find a way to access the secret server?

Solution:

a) From `/etc/services`

```
gopher              70/tcp                          # Internet Gopher
finger              79/tcp
http                80/tcp          www             # WorldWideWeb HTTP
kerberos    88/tcp          kerberos5 krb5 kerberos-sec     # Kerberos v5
kerberos    88/udp          kerberos5 krb5 kerberos-sec     # Kerberos v5
```

So it is using port 80, and the TCP protocol. Note that there is also an entry for https (for HTTP over TLS) and port 443 (once for TCP, and once over UDP for HTTP/3). Another entry is http-alt for port 8080 and TCP.

From that file we can also learn that NTP is using port 123 and the UDP protocol.

b) Sending the request as described yields the following response:

```
Trying 192.168.122.98...
Connected to group-51.lan.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Fri, 03 Jun 2022 10:38:54 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 15
Connection: close

Hello group <your-group-number>
Connection closed by foreign host.
```

Where `<your-group-number>` is of course your group number.

Solution:

c) The server line indicates that it is `nginx` running on Ubuntu.

d) The response to `curl` is:

```
Hello group <your-group-number>
I see you know how to use curl
```

How does the web server know that you are using curl? Well curl sends a User-Agent header to the server. Our web server searches for the string "curl" in the User-Agent header when preparing a response. You can verify that by sending a HTTP request over telnet with a User-Agent header which contains curl, e.g.:

```
Trying 192.168.122.98...
Connected to group-51.lan.
Escape character is '^]'.
GET / HTTP/1.0
User-Agent: curl

HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Fri, 03 Jun 2022 10:42:46 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 46
Connection: close

Hello group <your-group-number>
I see you know how to use curl
Connection closed by foreign host.
```

e) You just get the same response as telnet displayed in your browser.

f) You correctly deduce that the other web site is hosted on the same server using virtual hosts. So by passing a Host header over telnet, you can get the contents of that server:

```
Trying 192.168.122.98...
Connected to group-51.lan.
Escape character is '^]'.
GET / HTTP/1.0
Host: www2.lan

HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Fri, 03 Jun 2022 10:44:32 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 42
Connection: close

Welcome to the secret web server group <your-group-number>
Connection closed by foreign host.
```

Where `<your-group-number>` is of course your group number.

13.3 (Extra) IP Sockets – Establishing a TCP connection with C

In this task you will establish a TCP connection to a server using C. All the tools you need (a C compiler and the appropriate headers) are already installed on you VM. To make sure you are not interfering with other members of your group, work inside a directory named after your `nethz` username. You can create a directory using `mkdir(1)` and switch to it with `cd`:

```
mkdir <nethz-name>
cd <nethz-name>
```

Test your setup To test the setup use the file `example.c` from the course web site. You can either download it on your computer and then upload it with `scp(1)` to your VM, or download it directly to your VM using `curl(1)`. You can now compile it using:

```
cc example.c
```

and execute the resulting file using:

```
./a.out
```

You should see the content of `example.c` in your terminal.

Useful functions, structs etc. *Hint:* As shown in the presentation at the beginning of the session, text followed by a number in parentheses like `socket(2)` are references to a manual page. `socket` is the name of the page and 2 is the section number. To read the page use the `man(1)` command. For example to read the manual of `socket(2)` do:

```
man 2 socket
```

Type *q* to exit the man page again.

The following functions, structs and man pages might be helpful to implement your TCP connection with C:

- **`struct sockaddr_in`** Holds an IP address and a port. Can be used as an argument to `connect(2)`. The structure layout is documented in `ip(7)`. Recall that a dot (`.`) is used to access `struct` elements and `->` if the variable is a pointer to a `struct`.
- **`inet_aton(3)`** Converts a string to `struct in_addr`.
- **`htons(3)`** Converts a 16-bit integer from *host* to *network* byte order. As your VMs use Little Endian, and network byte order is Big Endian you will have to use this to convert the destination port accordingly.
- **`socket(2)`** Used to obtain a descriptor. `socket(2)` is responsible for allocating descriptors for local (`AF_UNIX`) connections and remote connections (`AF_INET`, `AF_INET6`) among others. To select a protocol one has to use one of the `SOCK_*` flags. `SOCK_STREAM` is used for TCP, `SOCK_DGRAM` for UDP.
- **`connect(2)`** This connects a file descriptor obtained from `socket(2)` to a remote host.
- **`write(2)`** Write data to a file descriptor.
- **`read(2)`** Read data from a file descriptor.
- **`close(2)`** Closes a file descriptor (and the connection with it).

The following functions/built-ins can be of use as well:

- **`err(1)`** Exit the program with an error message.
- **`sizeof`** C built-in which returns the size of the argument in bytes as `size_t`.
- **`fwrite(3)`** Writes a buffer to a file. `stdin`, `stdout` and `stderr` are `FILE *` which read from standard input and write to standard output/error respectively.

Implementation As a starting point for your implementation we provide a skeleton file on the course website (`skeleton.c`), which already contains the relevant includes. Be careful to check the return values of the system calls. `err(1)` exits the process and prints a message according to the value of `errno(3)`.

Try to complete the skeleton file following the hints in the comments. Are you able to establish the TCP connection?

Debugging You can see all outgoing TCP connections using the `ss(8)` command like this:

```
ss -t
```

Next steps Here are some ideas on how to extend your program:

- Use `getaddrinfo(3)` to convert `www.lan` to the corresponding IP address.
- Switch to use `HTTP/1.1` instead. For that you will need to include a `Host` header.
- Allow the user to specify a hostname on the command line.

Solution: Have a look at the file `solution.c` on our website. Note that we are *not* using the IP of `www.lan` but instead the IP of `vvz.ethz.ch` so that you can also try this example at home (outside of our VMs).