ATTACK DEFENSE

by PentesterAcademy

Name	Linux Capabilities
URL	https://attackdefense.com/challengedetails?cid=1824
Type	Privilege Escalation : Linux Capabilities

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

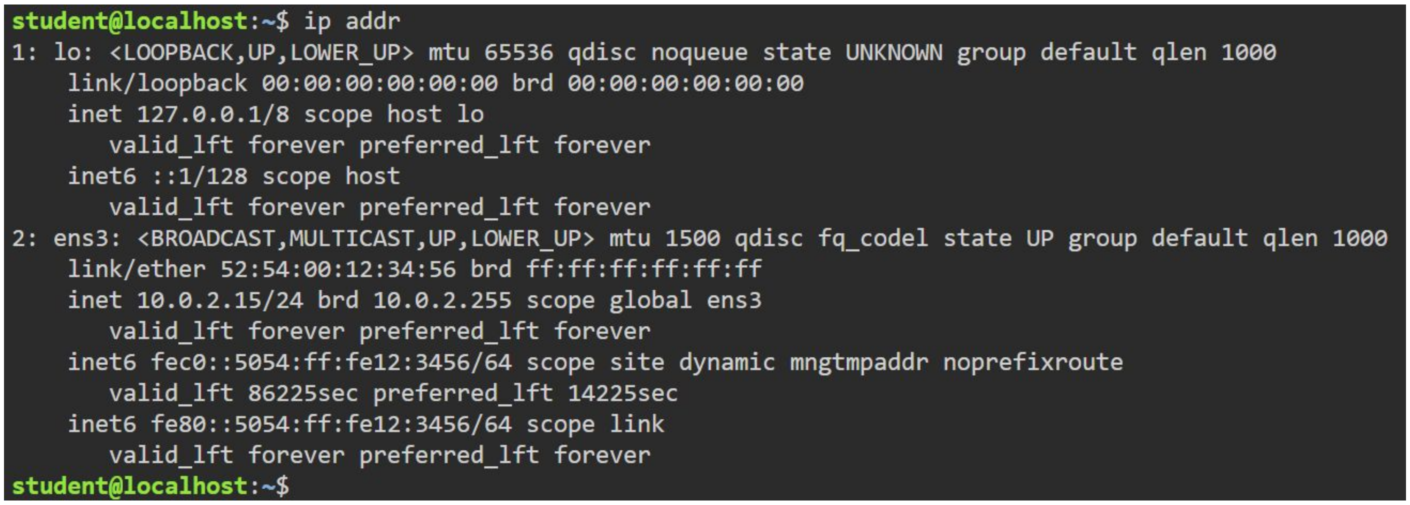
Objective: Learn about inherited and ambient Linux Capabilities sets using an example of semi-privileged environment.

Solution:

Our motive here is to be able to run all binaries which need CAP_NET_RAW and CAP_NET_ADMIN capabilities to work. However, as multiple binaries are in question, the idea is not to individually set capabilities on all binaries but to create a semi-privileged environment taking advantage of inherited and ambient sets.

Step 1: Check IP address and connected interfaces of machine

Command: ip addr

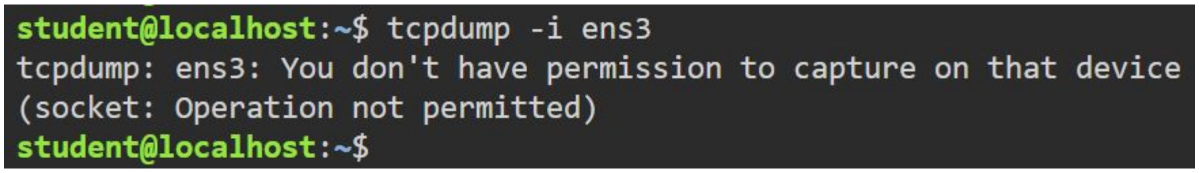


```
student@localhost:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global ens3
       valid_lft forever preferred_lft forever
    inet6 fec0::5054:ff:fe12:3456/64 scope site dynamic mngtmpaddr noprefixroute
       valid_lft 86225sec preferred_lft 14225sec
    inet6 fe80::5054:ff:fe12:3456/64 scope link
       valid_lft forever preferred_lft forever
student@localhost:~$
```

For now, focus on two programs that are covered in the first two challenges i.e. ping and tcpdump.

Step 2: Run tcpdump on ens3

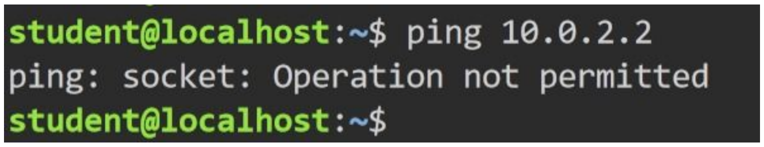
Command: tcpdump -i ens3



```
student@localhost:~$ tcpdump -i ens3
tcpdump: ens3: You don't have permission to capture on that device
(socket: Operation not permitted)
student@localhost:~$
```

Step 3: Run ping command to ping the gateway machine

Command: ping 10.0.2.2



```
student@localhost:~$ ping 10.0.2.2
ping: socket: Operation not permitted
student@localhost:~$
```

Unlike Effective, Permitted and Inherited capabilities, ambient capability can't be set using setcap utility.

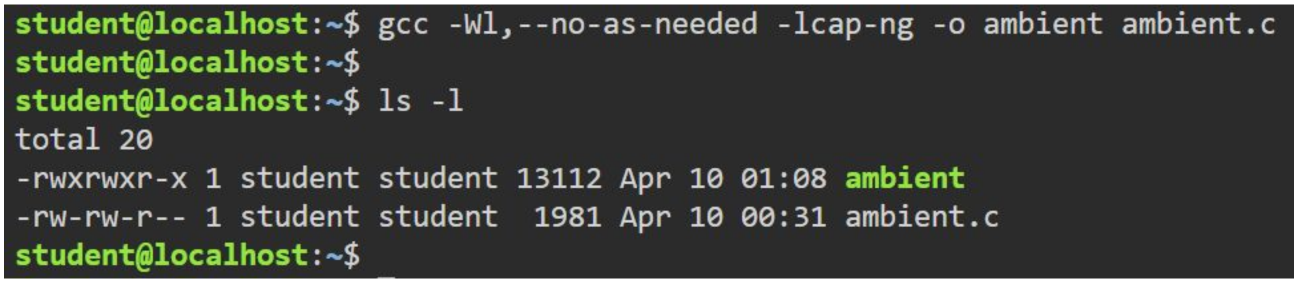
The C code for creating a binary that can set ambient capabilities, is given here:
https://gist.githubusercontent.com/infinity0/596845b5eea3e1a02a018009b2931a39/raw/db1c6fbb66825b7628ebf274d3ddebe4ba13795e/ambient.c

C code:

```
/*
 * Test program for the ambient capabilities
 *
 * compile using:
 * gcc -Wl,--no-as-needed -lcap-ng -o ambient ambient.c
 *  Set effective, inherited and permitted capabilities to the compiled binary
 *  sudo setcap cap_setpcap,cap_net_raw,cap_net_admin,cap_sys_nice+eip ambient
 *
 * To get a shell with additional caps that can be inherited do:
 *
 * ./ambient /bin/bash
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <sys/prctl.h>
#include <linux/capability.h>
#include <cap-ng.h>

static void set_ambient_cap(int cap)
{
        int rc;

        capng_get_caps_process();
        rc = capng_update(CAPNG_ADD, CAPNG_INHERITABLE, cap);
        if (rc) {
                printf("Cannot add inheritable cap\n");
                exit(2);
        }
        capng_apply(CAPNG_SELECT_CAPS);

        /* Note the two 0s at the end. Kernel checks for these */
        if (prctl(PR_CAP_AMBIENT, PR_CAP_AMBIENT_RAISE, cap, 0, 0)) {
                perror("Cannot set cap");
                exit(1);
        }
}

void usage(const char *me) {
        printf("Usage: %s [-c caps] new-program new-args\n", me);
        exit(1);
}

int default_caplist[] = {CAP_NET_RAW, CAP_NET_ADMIN, CAP_SYS_NICE, -1};
int *get_caplist(const char *arg) {
        int i = 1;
        int *list = NULL;
        char *dup = strdup(arg), *tok;

        for (tok = strtok(dup, ","); tok; tok = strtok(NULL, ",")) {
                list = realloc(list, (i + 1) * sizeof(int));
                if (!list) {
                        perror("out of memory");
                        exit(1);
                }
                list[i-1] = atoi(tok);
                list[i] = -1;
                i++;
        }
        return list;
}

int main(int argc, char **argv)
{
        int rc, i, gotcaps = 0;
        int *caplist = NULL;
        int index = 1; // argv index for cmd to start

        if (argc < 2)
                usage(argv[0]);

        if (strcmp(argv[1], "-c") == 0) {
                if (argc <= 3) {
                        usage(argv[0]);
                }
                caplist = get_caplist(argv[2]);
                index = 3;
        }

        if (!caplist) {
                caplist = (int *)default_caplist;
        }
        for (i = 0; caplist[i] != -1; i++) {
                printf("adding %d to ambient list\n", caplist[i]);
                set_ambient_cap(caplist[i]);
        }

        printf("Ambient forking shell\n");
        if (execv(argv[index], argv + index))
                perror("Cannot exec");

        return 0;
}
```

Step 4: Copy this code into the ambient.c and compile it.

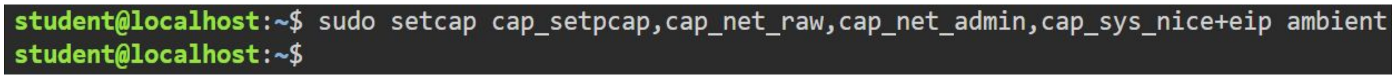
Command: gcc -Wl,--no-as-needed -lcap-ng -o ambient ambient.c



```
student@localhost:~$ gcc -Wl,--no-as-needed -lcap-ng -o ambient ambient.c
student@localhost:~$
student@localhost:~$ ls -l
total 20
-rwxrwxr-x 1 student student 13112 Apr 10 01:08 ambient
-rw-rw-r-- 1 student student  1981 Apr 10 00:31 ambient.c
student@localhost:~$
```

Step 5: Set effective, inherited and permitted capabilities to the compiled binary

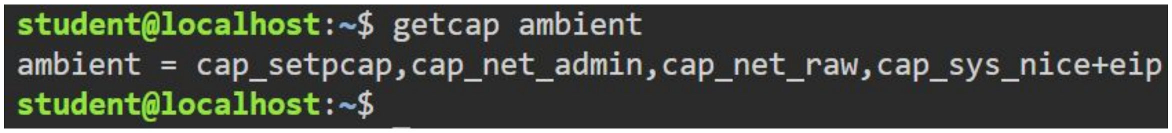
Command: sudo setcap cap_setpcap,cap_net_raw,cap_net_admin,cap_sys_nice+eip ambient



```
student@localhost:~$ sudo setcap cap_setpcap,cap_net_raw,cap_net_admin,cap_sys_nice+eip ambient
student@localhost:~$
```

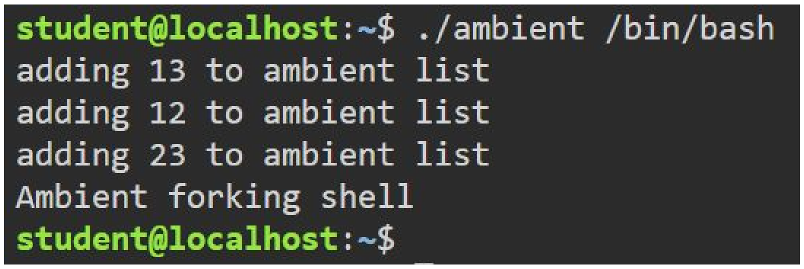
Step 6: Verify the permission set on the ambient binary

Command: getcap ambient


```
student@localhost:~$ getcap ambient
ambient = cap_setpcap,cap_net_admin,cap_net_raw,cap_sys_nice+eip
student@localhost:~$
```

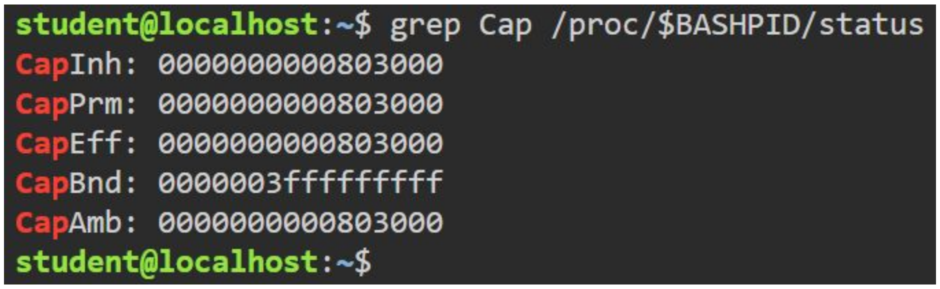
Step 7: Run bash using ambient binary

Command: ./ambient /bin/bash


```
student@localhost:~$ ./ambient /bin/bash
adding 13 to ambient list
adding 12 to ambient list
adding 23 to ambient list
Ambient forking shell
student@localhost:~$
```

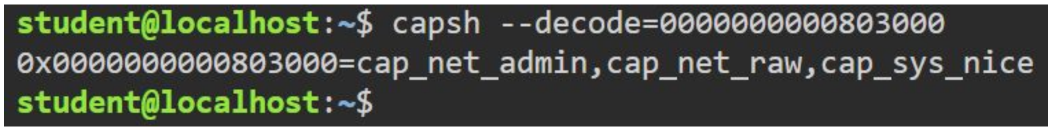
Step 8: Check the capability sets for the spawned bash shell

Command: grep Cap /proc/$BASHPID/status


```
student@localhost:~$ grep Cap /proc/$BASHPID/status
CapInh: 0000000000803000
CapPrm: 0000000000803000
CapEff: 0000000000803000
CapBnd: 0000003fffffffff
CapAmb: 0000000000803000
student@localhost:~$
```

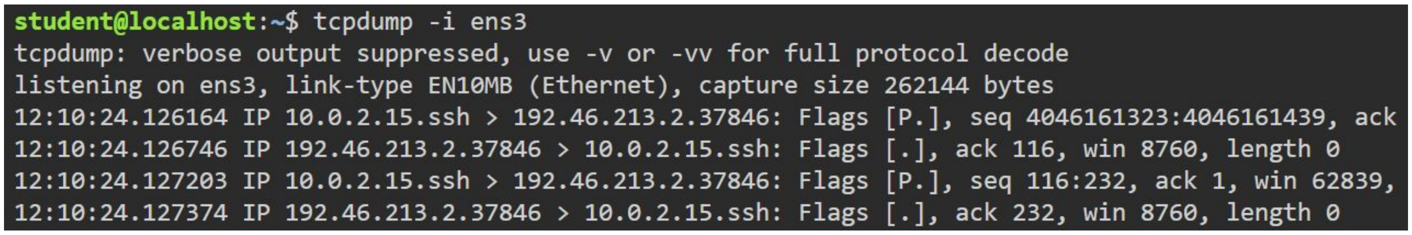
Step 9: The Capabilities listed in this file are not in human-readable form. Decode permitted capability (CapPrm) using capsh.

Command: capsh --decode=0000000000803000


```
student@localhost:~$ capsh --decode=0000000000803000
0x0000000000803000=cap_net_admin,cap_net_raw,cap_sys_nice
student@localhost:~$
```

Step 10: Run tcpdump on interface ens3

Command: tcpdump -i ens3

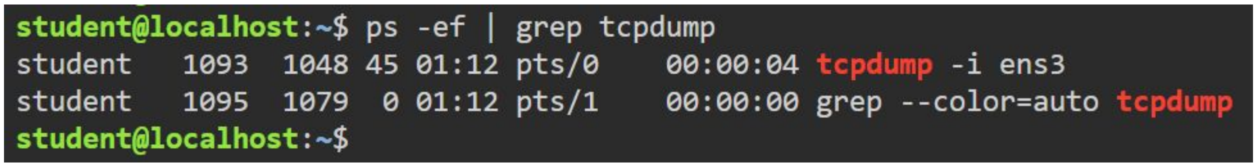


```
student@localhost:~$ tcpdump -i ens3
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens3, link-type EN10MB (Ethernet), capture size 262144 bytes
12:10:24.126164 IP 10.0.2.15.ssh > 192.46.213.2.37846: Flags [P.], seq 4046161323:4046161439, ack
12:10:24.126746 IP 192.46.213.2.37846 > 10.0.2.15.ssh: Flags [.], ack 116, win 8760, length 0
12:10:24.127203 IP 10.0.2.15.ssh > 192.46.213.2.37846: Flags [P.], seq 116:232, ack 1, win 62839,
12:10:24.127374 IP 192.46.213.2.37846 > 10.0.2.15.ssh: Flags [.], ack 232, win 8760, length 0
```

This time it is able to capture the traffic.

Step 11: Check the PID of the tcpdump process.

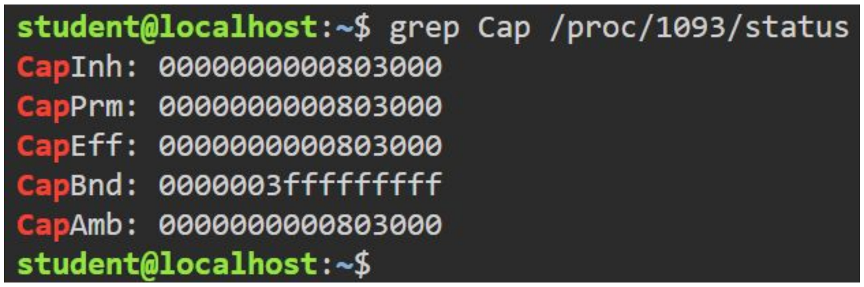
Command: ps -ef | grep tcpdump



```
student@localhost:~$ ps -ef | grep tcpdump
student    1093  1048 45 01:12 pts/0    00:00:04 tcpdump -i ens3
student    1095  1079  0 01:12 pts/1    00:00:00 grep --color=auto tcpdump
student@localhost:~$
```

Step 12: Check the capability sets for tcpdump process

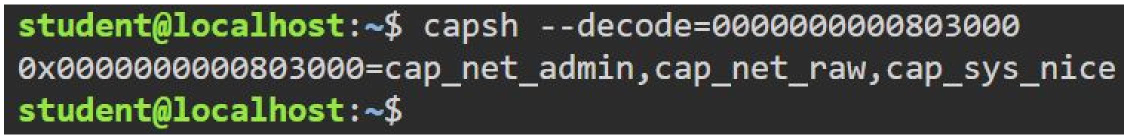
Command: grep Cap /proc/1093/status



```
student@localhost:~$ grep Cap /proc/1093/status
CapInh: 0000000000803000
CapPrm: 0000000000803000
CapEff: 0000000000803000
CapBnd: 0000003fffffffff
CapAmb: 0000000000803000
student@localhost:~$
```

Step 13: The Capabilities listed in this file are not in human-readable form. Decode permitted capability (CapPrm) using capsh.

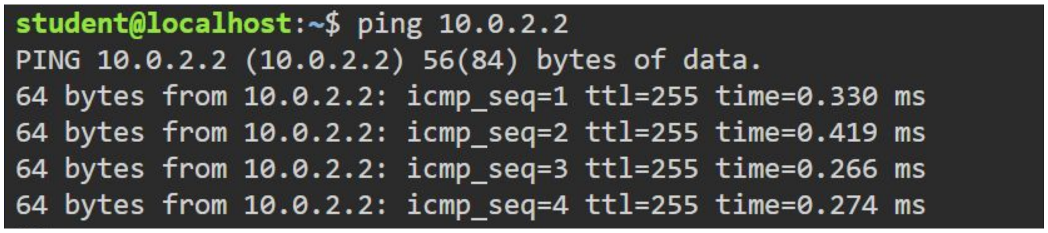
Command: capsh --decode=0000000000803000



```
student@localhost:~$ capsh --decode=0000000000803000
0x0000000000803000=cap_net_admin,cap_net_raw,cap_sys_nice
student@localhost:~$
```

Step 14: Run the ping command

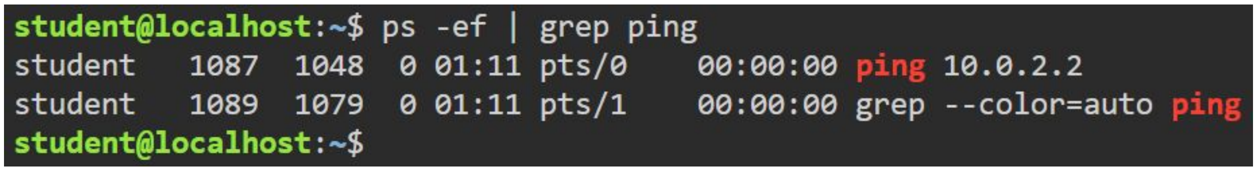
Command: ping 10.0.2.2



```
student@localhost:~$ ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=255 time=0.330 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=255 time=0.419 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=255 time=0.266 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=255 time=0.274 ms
```

Step 15: Check the PID of the ping process.

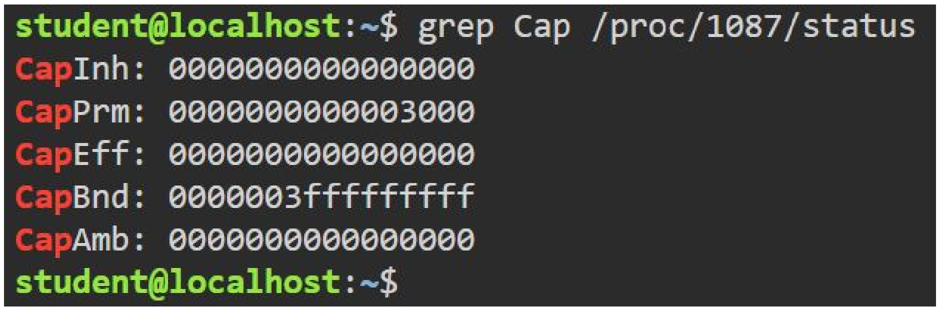
Command: ps -ef | grep ping



```
student@localhost:~$ ps -ef | grep ping
student    1087  1048  0 01:11 pts/0    00:00:00 ping 10.0.2.2
student    1089  1079  0 01:11 pts/1    00:00:00 grep --color=auto ping
student@localhost:~$
```

Step 16: Check the capability sets for ping process

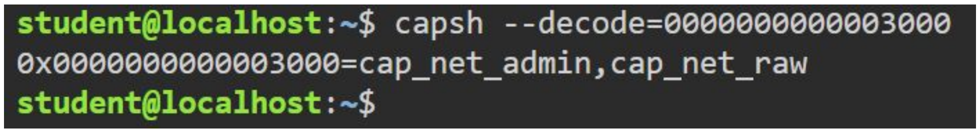
Command: grep Cap /proc/1087/status



```
student@localhost:~$ grep Cap /proc/1087/status
CapInh: 0000000000000000
CapPrm: 0000000000003000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
student@localhost:~$
```

Step 17: The Capabilities listed in this file are not in human-readable form. Decode permitted capability (CapPrm) using capsh.

Command: capsh --decode=0000000000003000



```
student@localhost:~$ capsh --decode=0000000000003000
0x0000000000003000=cap_net_admin,cap_net_raw
student@localhost:~$
```

Observations

- Both utilities worked properly in the semi-privileged environment without user setting any capabilities/setuid bits on the binaries.
- Ambient and inherited capabilities set on bash session got passed down to the utilities.
- One can observe the permission set difference between a "Capability-aware" utility (i.e. ping) and a "Capability-dumb" (i.e. tcpdump) utility.

References:

- Capabilities man page (http://man7.org/linux/man-pages/man7/capabilities.7.html)
- Linux capabilities in practice (https://blog.container-solutions.com/linux-capabilities-in-practice)
- Linux Audit (https://linux-audit.com/linux-capabilities-101/)
- Working with Linux capabilities (https://www.vultr.com/docs/working-with-linux-capabilities)