ATTACK DEFENSE

by PentesterAcademy

Name	Linux Capabilities IV
URL	https://attackdefense.com/challengedetails?cid=1825
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 how to cap the capabilities of a service.

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.

Create a sample service with a binary and a service configuration file

Step 1: Paste the code (given below) in servicebinary.c file.

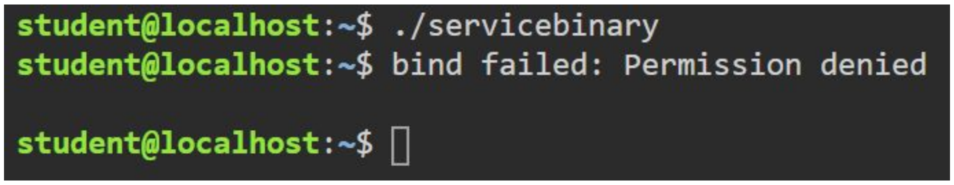
Code:

```
// Server side C/C++ program to demonstrate Socket programming
// Server code taken from: https://www.geeksforgeeks.org/socket-programming-cc/
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 80

int server()
```

```
{
        int server_fd, new_socket, valread;
        struct sockaddr_in address;
        int opt = 1;
        int addrlen = sizeof(address);
        char buffer[1024] = {0};
        char *hello = "Hello from server";

        // Creating socket file descriptor
        if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
        {
                perror("socket failed");
                exit(EXIT_FAILURE);
        }

        // Forcefully attaching socket to the port 8080
        if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                                        &opt, sizeof(opt)))
        {
                perror("setsockopt");
                exit(EXIT_FAILURE);
        }
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons( PORT );

        // Forcefully attaching socket to the port 8080
        if (bind(server_fd, (struct sockaddr *)&address,
                                        sizeof(address))<0)
        {
                perror("bind failed");
                exit(EXIT_FAILURE);
        }
        if (listen(server_fd, 3) < 0)
        {
                perror("listen");
                exit(EXIT_FAILURE);
        }
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                                        (socklen_t*)&addrlen))<0)
        {
                perror("accept");
                exit(EXIT_FAILURE);
        }
        valread = read( new_socket , buffer, 1024);
        printf("%s\n",buffer );
        send(new_socket , hello , strlen(hello) , 0 );
        printf("Hello message sent\n");
        return 0;
}

int main()
{
        if (fork() == 0) {
                server();
        }

        return 0;
}
```

The above program forks a PoC server which listens on port 80 for a connection. After forking the server, the main process exits.

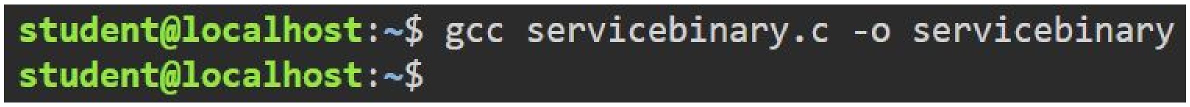
And, to open socket on port 80 (or any port below 1024), the a non-root user will need CAP_NET_BIND_SERVICE capability. So, it can't be run directly by the student user.



```
student@localhost:~$ ./servicebinary
student@localhost:~$ bind failed: Permission denied

student@localhost:~$ 
```

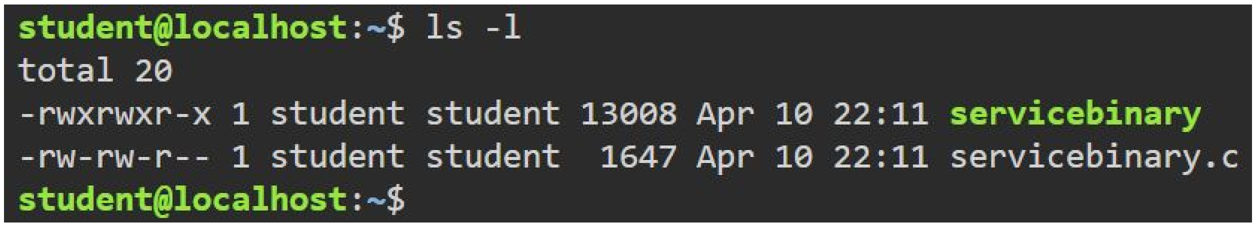
Step 2: Compile the servicebinary.c file.

Command: gcc servicebinary.c -o servicebinary


```
student@localhost:~$ gcc servicebinary.c -o servicebinary
student@localhost:~$
```

The servicebinary executable is ready.

Command: ls -l


```
student@localhost:~$ ls -l
total 20
-rwxrwxr-x 1 student student 13008 Apr 10 22:11 servicebinary
-rw-rw-r-- 1 student student  1647 Apr 10 22:11 servicebinary.c
student@localhost:~$
```

Step 3: Create a service config file for it (i.e. /lib/systemd/system/servicebinary.service). Paste the following config in the file

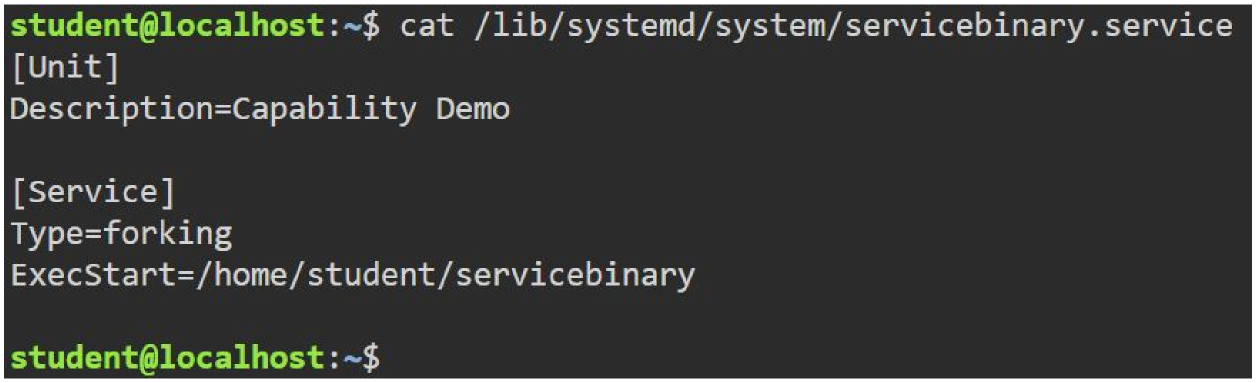
Command: sudo vim /lib/systemd/system/servicebinary.service

Configuration:

```
[Unit]
Description=Capability Demo

[Service]
Type=forking
ExecStart=/home/student/servicebinary
```

Verify the configuration file.

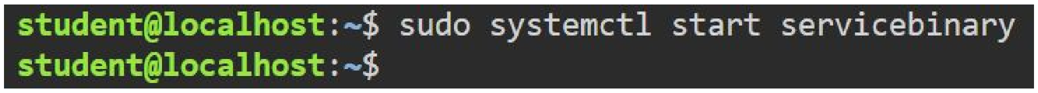
Command: cat /lib/systemd/system/servicebinary.service



```
student@localhost:~$ cat /lib/systemd/system/servicebinary.service
[Unit]
Description=Capability Demo

[Service]
Type=forking
ExecStart=/home/student/servicebinary

student@localhost:~$
```

Step 4: The service can be started using systemctl command.

Command: sudo systemctl start servicebinary



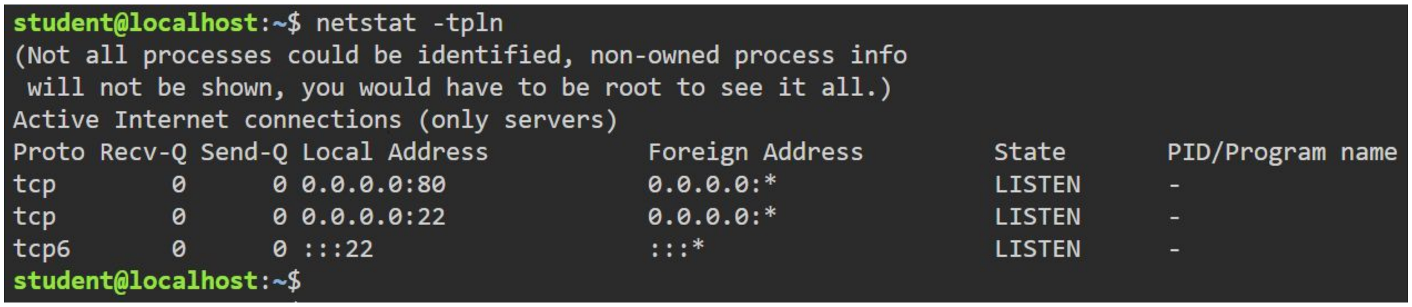
```
student@localhost:~$ sudo systemctl start servicebinary
student@localhost:~$
```

The service is running on the machine.

Scenario I: Service running as root

Step 5: Check if the port 80 is open and is in listen mode.

Command: netstat -tpln

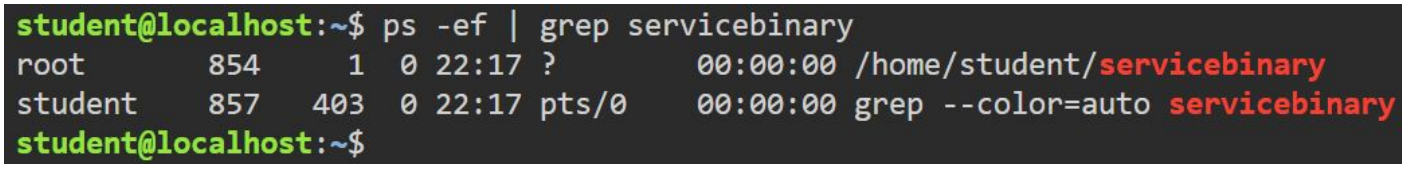


```
student@localhost:~$ netstat -tpln
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
student@localhost:~$
```

The port 80 is open.

Step 6: Currently the binary is running with the root user, check the PID of the binary.

Command: ps -ef | grep servicebinary

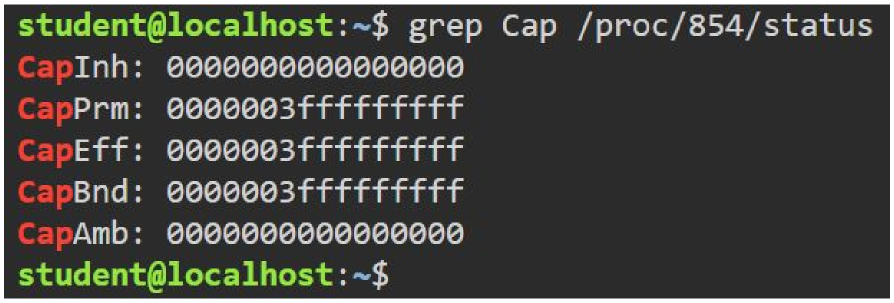


```
student@localhost:~$ ps -ef | grep servicebinary
root         854       1  0 22:17 ?        00:00:00 /home/student/servicebinary
student      857     403  0 22:17 pts/0    00:00:00 grep --color=auto servicebinary
student@localhost:~$
```

The PID is 854.

Step 7: Check the capability sets for the running binary using its PID

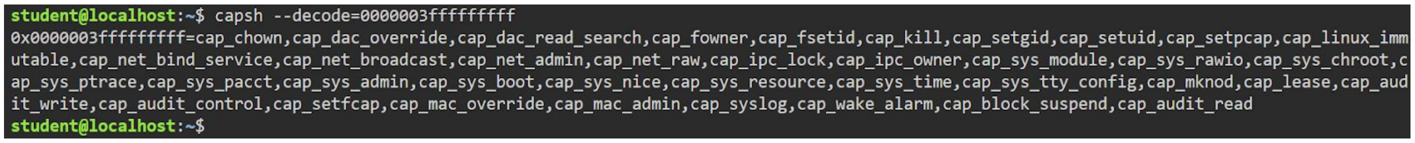
Command: grep Cap /proc/854/status



```
student@localhost:~$ grep Cap /proc/854/status
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
student@localhost:~$
```

Step 8: The Capabilities listed in this file are not in human-readable form. Decode permitted capability (CapPrm) using capsh.

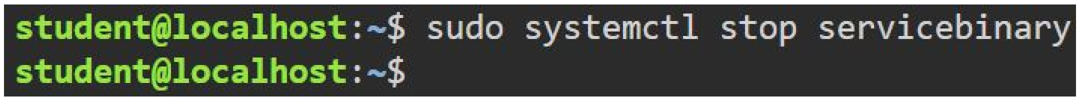
Command: capsh --decode=0000003fffffffff



```
student@localhost:~$ capsh --decode=0000003fffffffff
0x0000003fffffffff=cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_imm
utable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,cap_sys_chroot,c
ap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_aud
it_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read
student@localhost:~$
```

Step 9: Stop the service

Command: sudo systemctl stop servicebinary



```
student@localhost:~$ sudo systemctl stop servicebinary
student@localhost:~$
```

Scenario II: Service running as student user with capability

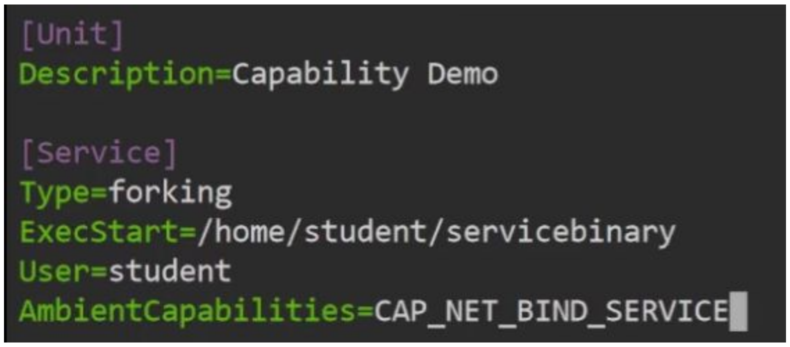
Step 10: Change the service config file /lib/systemd/system/servicebinary.service to run it with student user

Command: sudo vim /lib/systemd/system/servicebinary.service

Configuration:

```
[Unit]
Description=Capability Demo

[Service]
Type=forking
ExecStart=/home/student/servicebinary
User=student
AmbientCapabilities=CAP_NET_BIND_SERVICE
```

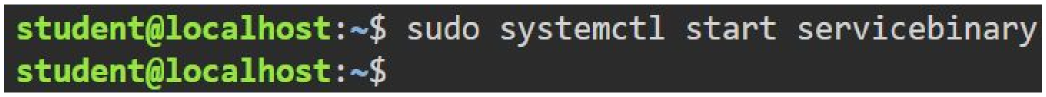
Command: vim /lib/systemd/system/servicebinary.service



```
[Unit]
Description=Capability Demo

[Service]
Type=forking
ExecStart=/home/student/servicebinary
User=student
AmbientCapabilities=CAP_NET_BIND_SERVICE
```

Step 11: The service can be started using systemctl command.

Command: sudo systemctl start servicebinary

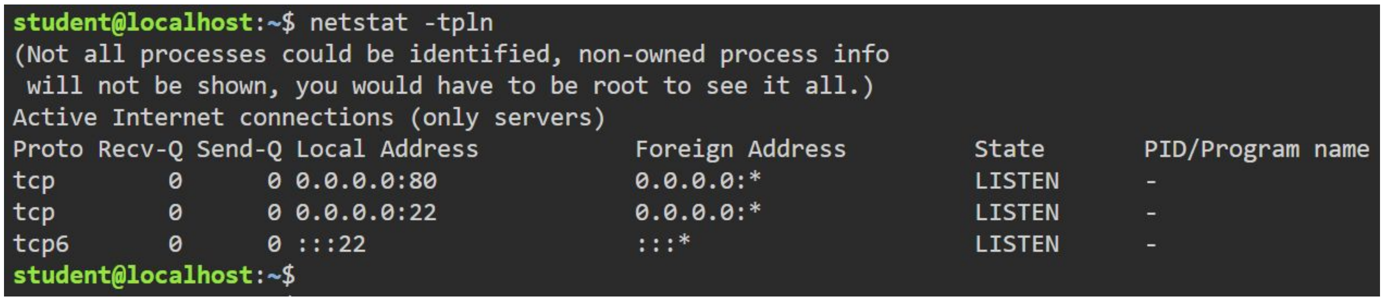


```
student@localhost:~$ sudo systemctl start servicebinary
student@localhost:~$
```

The service is running on the machine.

Step 12: Check if the port 80 is open and is in listen mode.

Command: netstat -tpln

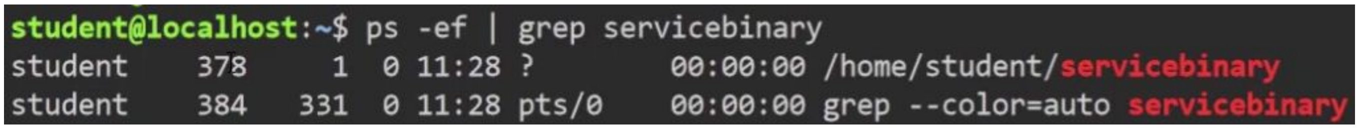


```
student@localhost:~$ netstat -tpln
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
student@localhost:~$
```

The port 80 is open.

Step 13: This time the service is running as the student user with CAP_NET_BIND_SERVICE set as Ambient capability. Get PID of sleep.

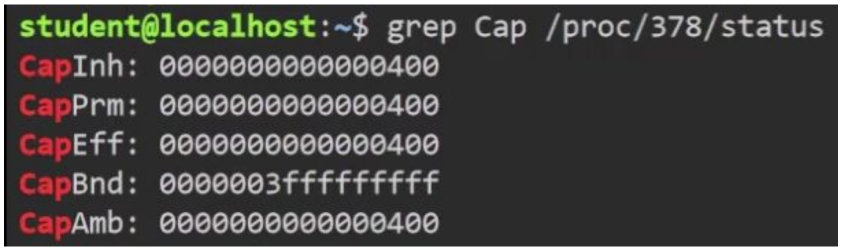
Command: ps -ef | grep servicebinary



```
student@localhost:~$ ps -ef | grep servicebinary
student    378      1  0 11:28 ?        00:00:00 /home/student/servicebinary
student    384    331  0 11:28 pts/0    00:00:00 grep --color=auto servicebinary
```

Step 14: Check the capability sets for the sleep

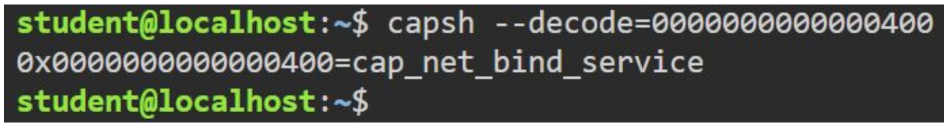
Command: grep Cap /proc/378/status



```
student@localhost:~$ grep Cap /proc/378/status
CapInh: 0000000000000400
CapPrm: 0000000000000400
CapEff: 0000000000000400
CapBnd: 0000003fffffffff
CapAmb: 0000000000000400
```

Step 15: The Capabilities listed in this file are not in human-readable form. Decode permitted capability (CapPrm) using capsh.

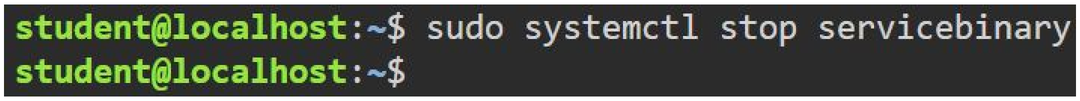
Command: capsh --decode=0000000000000400



```
student@localhost:~$ capsh --decode=0000000000000400
0x0000000000000400=cap_net_bind_service
student@localhost:~$
```

Step 16: Stop the service

Command: sudo systemctl stop servicebinary



```
student@localhost:~$ sudo systemctl stop servicebinary
student@localhost:~$
```

Scenario III: Service running as root user with bounding capability set defined

Step 17: Change the service config file /lib/systemd/system/servicebinary.service

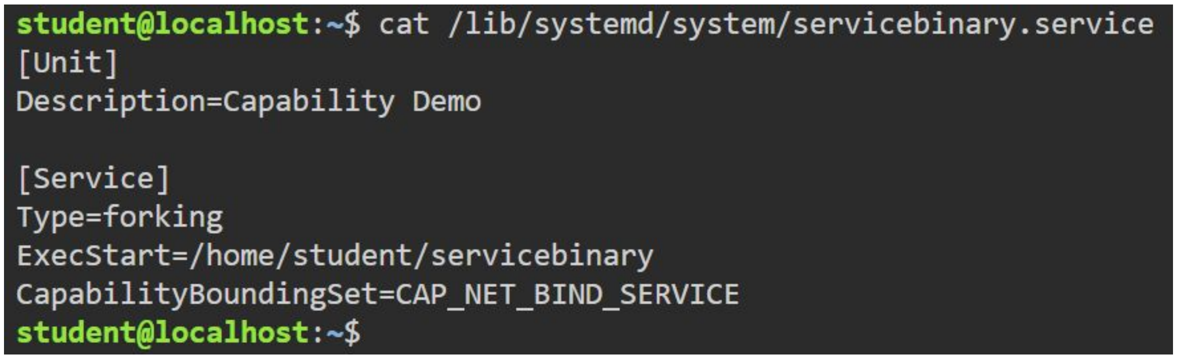
Command: vim /lib/systemd/system/servicebinary.service

Configuration:

```
[Unit]
Description=Capability Demo

[Service]
Type=forking
ExecStart=/home/student/servicebinary
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
```

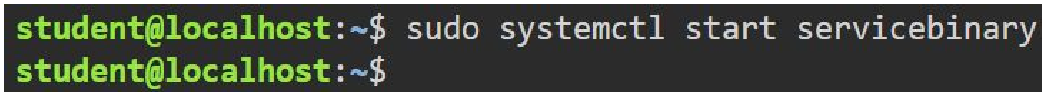
Verify the changes made

Command: cat /lib/systemd/system/servicebinary.service


```
student@localhost:~$ cat /lib/systemd/system/servicebinary.service
[Unit]
Description=Capability Demo

[Service]
Type=forking
ExecStart=/home/student/servicebinary
CapabilityBoundingSet=CAP_NET_BIND_SERVICE
student@localhost:~$
```

Step 18: The service can be started using systemctl command.

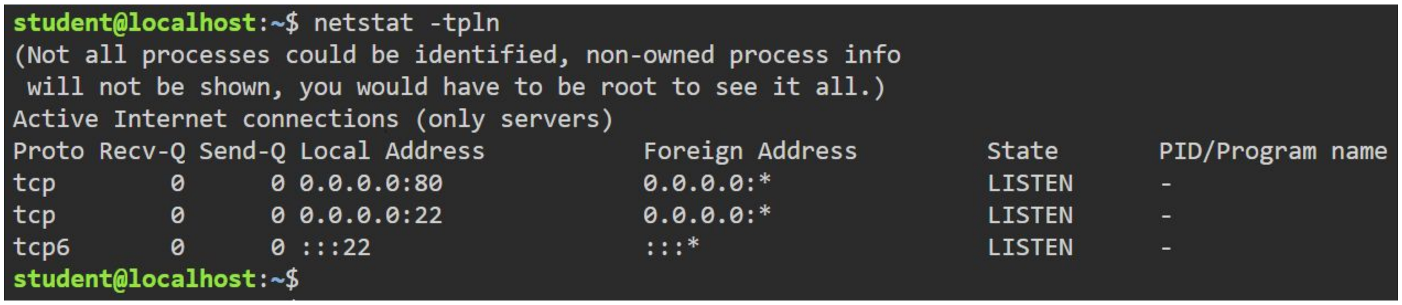
Command: sudo systemctl start servicebinary


```
student@localhost:~$ sudo systemctl start servicebinary
student@localhost:~$
```

The service is running on the machine.

Step 19: Check if the port 80 is open and is in listen mode.

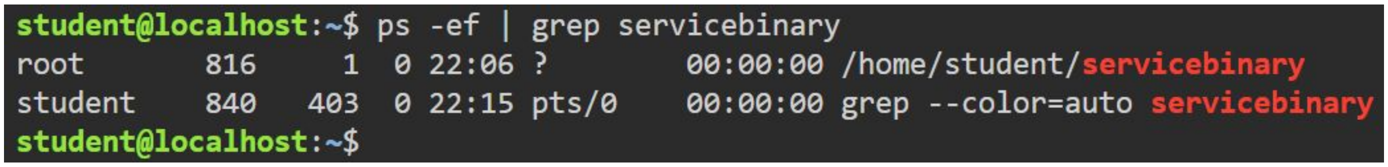
Command: netstat -tpln


```
student@localhost:~$ netstat -tpln
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
student@localhost:~$
```

The port 80 is open.

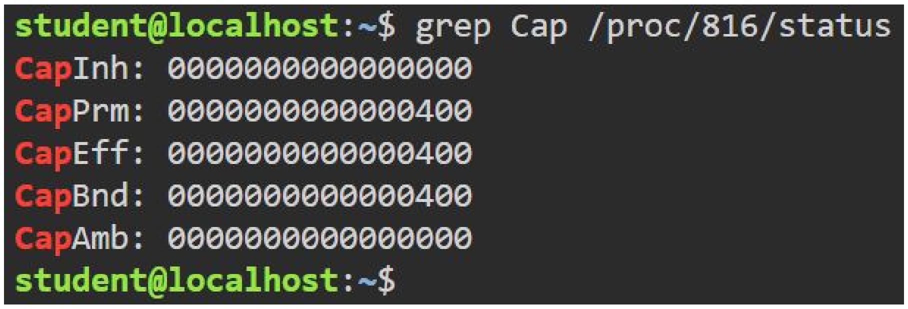
Step 20: This time the service is running as the student user with CAP_NET_BIND_SERVICE set as Bounding set. Get the PID of the process.

Command: ps -ef | grep servicebinary


```
student@localhost:~$ ps -ef | grep servicebinary
root         816       1  0 22:06 ?        00:00:00 /home/student/servicebinary
student      840     403  0 22:15 pts/0    00:00:00 grep --color=auto servicebinary
student@localhost:~$
```

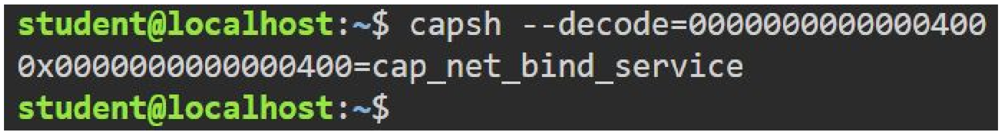
Step 21: Check the capability sets for this process

Command: grep Cap /proc/816/status


```
student@localhost:~$ grep Cap /proc/816/status
CapInh: 0000000000000000
CapPrm: 0000000000000400
CapEff: 0000000000000400
CapBnd: 0000000000000400
CapAmb: 0000000000000000
student@localhost:~$
```

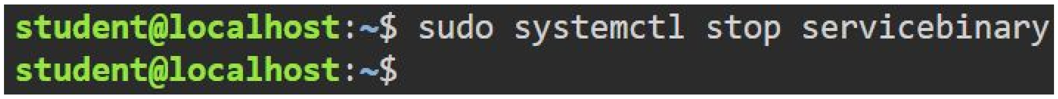
Step 22: The Capabilities listed in this file are not in human-readable form. Decode permitted capability (CapPrm) using capsh.

Command: capsh --decode=0000000000000400


```
student@localhost:~$ capsh --decode=0000000000000400
0x0000000000000400=cap_net_bind_service
student@localhost:~$
```

Step 23: Stop the service

Command: sudo systemctl stop servicebinary


```
student@localhost:~$ sudo systemctl stop servicebinary
student@localhost:~$
```

Learning Takeaway:

- Privileges/capabilities can be dropped for services by making changes to the service file.
- This practice can be used to minimize the attack surface.

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)
- Practical use of Linux Capabilities (https://www.youtube.com/watch?v=WYC6DHzWzFQ)