**Sudo**

**Sudo -l will show you what sudo permissions the current usr has** (access to which binaries)

* All sudo binaries offer ways to escalate privileges

Navigate to the /tmp dir and execute sudo -l

* It should show you that you have access to ‘find’, ‘vim, ‘nmap’ and ‘man’ amonh others

**Vim**: Sudo /usr/bin/vim.gtk test.txt

* Create a test.txt file
* At the bottom write ‘:sh’ to spawn a shell
* When you exit, a shell will be spawned as the root user

**Find**: Sudo /usr/bin/find /etc/passwd -exec /bin/bash \;

* #-exec tell it to execute /bin/bash for every file we hit with the ‘find’ function
* Executing this spawns a bash shell as the root usr

**Man**: Sudo /usr/bin/man less

* Will bring manual page for less
* Type !sh at the bottom and execute
* You will now be root

**Nmap**: exho “os.execute(‘/bin/sh’)” > /tmp/shell.nse && sudo nmap –script=/tmp/shell.nse

* #creating a file (.nse, nmap script) for nmap to use
* This file will execute /bin/sh spawning an interactive shell
* The second part of the command is telling nmap to use the file we’ve created as its script to use
* When you execute the cmd, nmap will ‘start’ and therefore spawn a shell with root access

**HW – Look up other ways to spawn a shell using binaries**

* Can be an older version of a binary or something inconspicuous that an admin wouldn’t update or care about

**Relative Path Sudo Binary**

**Relative path** = ls

**Absolute path** = /usr/bin/ls

When using relative paths, exploit opportunities can arise

Text

Description automatically generated

When the ls binary is called, the system will check each of these paths for the ls binary

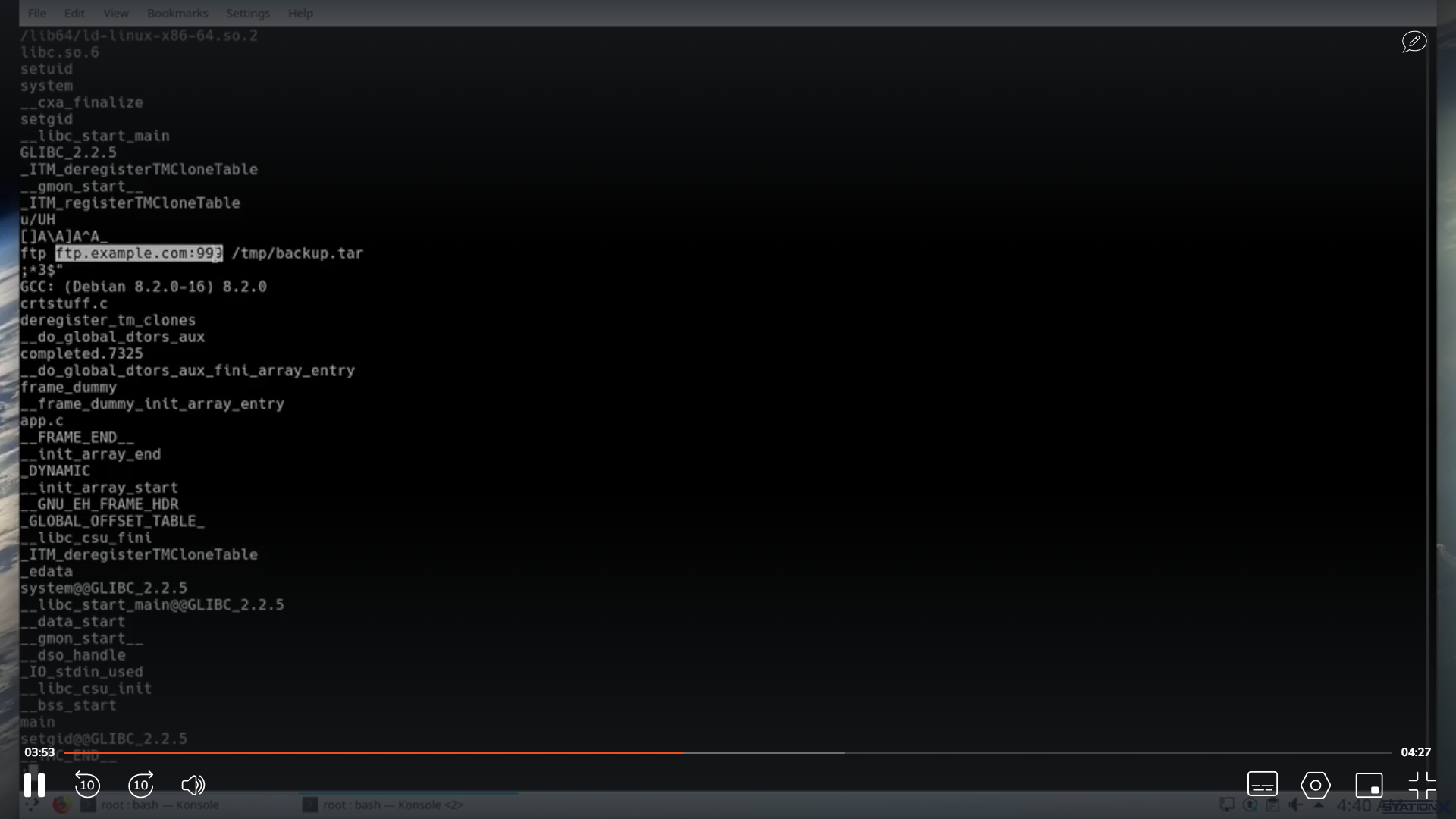
* If it can find it in the first one then it goes to the next and so on
* So /usr/local/sbin is always the first check and so on^

Find / -perm -4000 2>/dev/null

* -perm -4000 is our sudo binaries
* 2>/dev/null sends our errors to devnull so we don’t have to see them

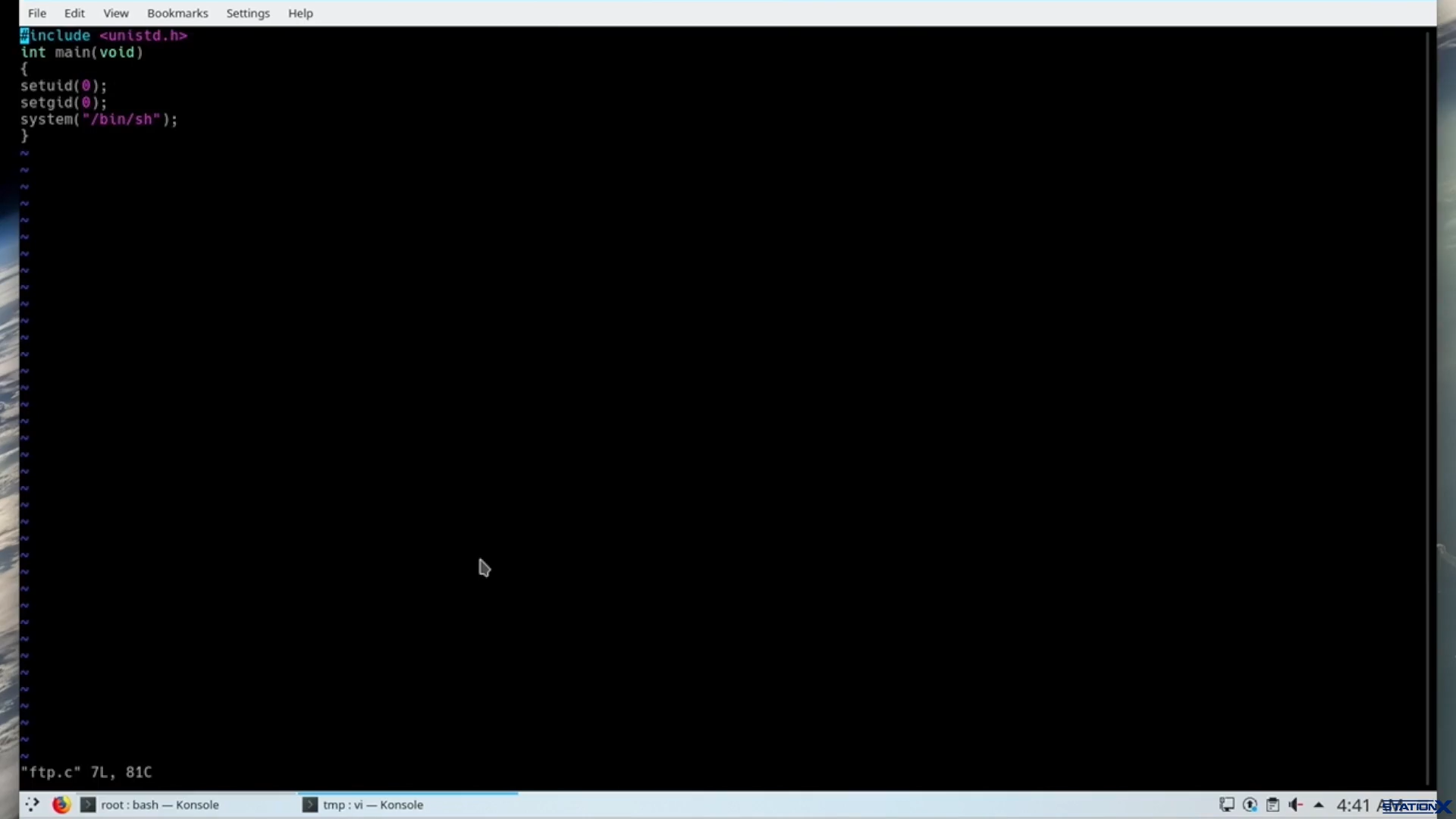
Strings: /usr/bin/backup

* A sudo binary
* This cmd will dump all the strings that binary uses to the screen



In the strings, we can see that ‘backup’ is trying to do a back up by calling ftp to tmp/backup.tar

* We can see to the left of the highlight that backup is using a relative path for ftp, not absolute
* This means we can write our binary and put it in the path so that when it tries to call ftp, it actually calls our binary



In the /tmp, make this file = ftp.c ^^

* Inside is some simple code in C to spawn a shell, which will need to be compiled first
* Set uid and gid to 0 which is root
* Then execute as system which is /bin/sh

Gcc ftp.c -o ftp

* Compile file in the /tmp dir

The directory containing the new ftp must be added to the path ‘backup’ is executing from

* Export PATH=/tmp:$PATH
  + $PATH prepend our path to the front of the queue
  + As in our path will now be the first one checked in the cascade

**Now running /usr/bin/backup will spawn a shell with root access**

**To do this, we abused a sudo bitset binary to believe the relative bath the binary is located in, is our path which also contains our custom binary which spawns a shell**

* **Executing the regular binary will now execute ours as a result**

**Anything that’s calling a binary with a relative path can be abused this way as long as its being run as sudo**

Could also use a semlink pointing to a point called ftp

Text

Description automatically generated

The codes at the bottom will help to test this out ourselves