|  |
| --- |
| **Lab 0: basic tools and access control**  **Created: Fall 2022, Updated: Spring 2023 (both by Amir Herzberg)**  **Monday sections submit by 1/29**  **Wednesday sections submit by 1/24** |

In this lab we will learn few basic tools that we will use throughout the course – and which you may find useful otherwise. In addition, we will learn some basic notions of computer security, specifically, we will learn a bit about the Linux access control (permissions) mechanism.

In all labs:

1. Work is done in teams of two students assigned by the lecturer before each lab (a pair). If you cannot cooperate well with your partner, *immediately* inform the TA; in addition, send email to the lecturer and fill out the form to split and submit individually or be assigned to a different partner. You can also use the form to request to avoid a partner when cooperation was not satisfactory, or to request to give preference to a particular partner (from the same section) in future labs. Do not change pairs (and definitely not add another student!) without authorization from the lecturer.
2. Both students in a pair should submit a well-written lab report in HuskyCT, with all commands and results obtained; use screen captures/recording from a laptop connecting over the VPN, not videos or photos from a phone. The lab report should clearly indicate names and NetIDs of the student(s) submitting, section and IP of VM. If you worked together (as a pair we assigned), it is fine to use the same report for both students, by including both your names.
3. Submit in HuskyCT any scripts or other code you develop as text within your reportDo not submit zip, .py or other files; if for some reason you want to also submit some code as a separate file, append the suffix *txt*.
4. Each student should also submit answers to the questions, including scripts/code, in the lab’s submission webserver, **submit.edu** (accessible only within the lab or over the VPN, using an IP address 172.16.48.8). The webserver will auto-grade many questions; you can use it to test your answers before submitting them. **On first use, chose `open account’ to choose your password.**
5. Make sure to maintain copies of all your work; use the VPN to transfer files between the VM and you workstation. You may also need to use output of one lab at a future lab.
6. In most labs, you will use challenge-files from the corresponding directory in your VM(s). The name of the directory is *Labi* where *i* is the number of the lab (*Lab0, Lab 1,…).* Usually, we provide the program which we used to generate these challenges, in a *LabGen* sub-directory of the lab’s directory; this is just for information, be careful not to corrupt your challenge files by running *LabGen* in the directory containing your challenges.
7. If you get `stuck’ in a question, ask help from the TA (in lab, or in office hours); you may also ask other students for hints (not solutions). And try to see if you can make progress in other questions in the lab.

**(Mandatory):** **Acknowledgment of Expectations and Ethical and Legal Implications**

1. **Each student** should fill in the `Acknowledgment of Expectation’ in HuskyCT.
2. **All students** should watch the `Ethical and Legal Implications’ lecture, using the `Computer Fraud Abuse Act’ presentation, both in HuskyCT. (The video shows mostly Prof. Fuller, so use the presentation in parallel to see the text.)

Contact the lecturer if anything is unclear.

**Do not proceed before completing these two steps!**

**Question 1 (5 points): VPN**

Remote-access Virtual Private Network (VPN) allows a computer to connect to a remote network, as if it is part of that network, allowing use of local network resources; many VPN protocols also use cryptography to protect the communication between the computer and the remote network, from tampering with the transmitted data, or eavesdropping on it. In this course, we will use a VPN to allow remote access to the lab computers, from your personal laptops or any other Internet-connected computer (on which you have authorization to install software such as VPN). This is important, it permits remote connection to the lab servers. The servers host the virtual machines you will be using to complete most labs for this class.

To connect to the lab’s VPN, you need to activate two-factor authentication for your UConn NetID login, from <https://netid.uconn.edu>, and to install and use a compatible VPN client. We recommend use of the FortiNet VPN client, which can be downloaded for free. **See video in HuskyCT for instructions** to all these steps, and also the file `VPN troubleshoot’ if you have problems. Note: this task can be done only from a computer on which you can install software; you cannot do it from the lab computers. So if you don’t have such computer right now, do this question later.

To use the submission server *submit.edu* (and to perform some labs) via the VPN, you will need to use [SSH tunneling](https://www.ssh.com/academy/ssh/tunneling). This will allow you access to the web server in the lab, by tunneling via your SSH connection. If you’re not yet familiar with SSH tunneling, we recommend you read about this important tool. We provide instructions for using SSH tunneling for this lab for student using MobaXterm (in Windows), in Discord and HuskyCT. We next provide a brief explanation (which also covers Mac users).

A **port** is a 16-bit integer identifier of TCP or UDP socket (roughly, connection) in the computer, allowing the same computer to hold multiple separate TCP/UDP connections (each using a different port).

When connecting using the VPN, the **firewall allows only traffic to specific ports**, most notably, port 22 which is connected (only) to the SSH server. So, when connecting using the VPN, we cannot communicate directly to port 80 where our web servers, including *submit.edu,* are listening. Instead, you should reach the web servers using [SSH local forwarding, aka SSH tunneling](https://www.ssh.com/academy/ssh/tunneling-example).

SSH tunneling works by relaying traffic sent to a *forwarded port* on the local SSH client machine (the lab laptop or your home computer), to a remote server and remote port (in our case, port 80), via an SSH server. You should be able to use as your forwarded port any number from 1024 till 9999; let’s say you use port 2222 (we picked 2222 since the SSH port is 22). Then, once the tunnel will be open, you’ll be able to connect to the web server in the lab (e.g., *submit.edu* ) by giving the URL <http://127.0.0.1:2222> to your browser. The IP 127.0.0.1 is reserved to signify the `local host’, i.e., the same computer (in this case, the laptop which you use, where the browser runs).

You’ll use your own VM (and IP) as the SSH server. You need to specify, to the tunnel, your SSH login information and the SSH port which is, by default, 22. The remote port would be 80, of course, and the ‘remote server’ will be [172.16.48.8](172.16.48.80) for *submit.edu.* In labs where you will need to similarly tunnel to other servers, you will similarly define additional tunnels (using other ports, e.g., 2223).

You can setup this SSH tunnel using any SSH client; MobaXterm provides a cute graphical interface which will also explain the operation of the tunnel, which is explained in a document available in HuskCT and Discord. MobaXterm also explain it all nicely in their application.

Using a MAC or other command-line SSH clients is also easy. Specifically, if forwarding from port 2222 and if your VM has IP address 172.16.x.y then you will use the command:

*ssh -L 127.0.0.1:2222:172.16.48.8*:80*cse@*[172.16.x](http://172.16.48.80).y

Students often misconfigure the SSH tunnel, so, be careful and check carefully that it works. A good reason to do this in the lab, if you can (i.e., if you have a non-lab laptop).

(Automatically validated, no need to submit anything.)

**Question 2 (5 points): SSH client and passwd**

After we connect to the lab, using the red Ethernet cable to connect your dedicated lab laptop – or via the VPN, we need to login into a specific virtual machine (VM). Each student is assigned a VM for the entire term; the TA will provide you with the IP address of your VM and your initial password. This is a virtual machine (computer) that runs on the lab servers’ hardware.

**To login to the lab’s laptops.** The user is cyberlab and the password should be cyberlab1382022. If that doesn’t work, you can try cyberlab1382019 (which was the older password).

We mostly connect via the Secure Shell (SSH) application, which allows you to remotely login into the Lab Virtual Machine (VM) assigned to each of you. SSH uses cryptography to protect the communication between the computer and the remote network, preventing tampering with the transmitted data, or eavesdropping on it.

SSH is pre-installed on some machines such as Macs; in other machines, specifically Windows, you need to install it, and we recommend the (free to install) *MobaXterm* SSH client. Follow the instructions in the HuskyCT video to download, install and configure your SSH client, as needed, and to login to your VM.

After you login, you should *change your VM’s password* from its default value. This can be done simply, by the command *passwd.* Do so and change your password.

(Automatically validated, no need to submit anything)

**Question 3 (15 points): directories: pwd , ls, man, cd and mkdirr**

Now that you are connected to your VM using SSH, type the three letters *pwd* and hit `enter’, i.e., run the *pwd* command. Copy exactly the output of this invocation of *pwd.*

The name of the *pwd* command stands for *print working directory*. Unix and Linux systems organize files into *directories*, which are sometimes referred to as *folders* since they are equivalent to Windows folders. Note: this lab applies to equally to Unix and Linux systems, so we’ll just use the term Unix (but apply to both).

Next, type the command *ls* and hit `enter’. As you would see, the *ls* command *lists* the files and directories within the current (working) directory. Notice that in the output you receive from *ls,* executed like this (without options), there is no visual difference in the way directories are displayed, vs. the way files are displayed. In fact, Unix (and Linux) systems consider a directory (aka folder) simply as a special kind of a file.

The name of one of the outputs is *Lab0*; this is a directory dedicated to this lab (Lab0); the directory contains files we would use in the lab, and a sub-directory called *Solutions* in which you will place answers (as well as provide the answers via HuskyCT; in redundancy we trust!).

But first, we need to *change directory* into *Lab0.* This is easy: type *cd Lab0* and hit `enter’. Do *pwd* again – and you will see that you have indeed moved to directory *Lab0*. Bravo!

The *ls* command has some interesting options and variations, which you can learn by googling, looking in Wikipedia – or by reading the manual. Reading the manual entry for Unix commands is relatively easy, since the manual is wholly contained within the system, using the command *man* (yes, short for `manual’). For example, to learn about the options of *ls*, type *man ls* and hit enter, you will get all the details you need (and a bit more, probably). In particular, you will find one simple and very important option for *ls,* that will result in more detailed output for each file in the directory, including whether it is a file or a directory, its size, the date and time at which it was last updated, and its permissions (which we’ll discuss soon). You may want to use this option to provide the answer to this question.

Before we finish this question, let us introduce one last directory command: *mkdir*. This command simply creates a new directory, within your working (current) directory. Use *mkdir* to create a directory *D3* within directory *Solutions.*

**Submit in submit.web (in lab) and in your report in HuskyCT:** (A) the name of a file in *Lab0* beginning with the letters ‘Q3.’, and (B) the size of this file (in bytes).

**Optional (but recommended): tmux**

Many times you may want to use a separate terminal windows to do different things, e.g., to display the `man’ entry for a command in one terminal window, and try to use it in another; or to edit your program in one terminal window, and run it in another, and so on. It’s really useful. You can do it by running multiple SSH windows to your VM, but let us teach you a more efficient, `Unix-style’ way to do it: *tmux.* And *tmux* has another very important feature: if your session gets disconnected – it happens – you can reconnect to the `lost’ session; this can be a lifesaver. So we recommend you try tmux; it may make you more productive. You decide.

To learn about tmux, see the video we put in HuskyCT, and/or use other resources, e.g., <https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/> (which also briefly discuss some alternative tools).

**Question 4 (10 points): vim (or another text editor)**

One basic thing you will need to do is to write and edit files, using a *text editor*. Unix and Linux systems typically have few editors. If you are not familiar with any of them, we suggest you use *vim;* itis one of the simple editors, and our staff would probably be able to assist you with questions about it. But if you prefer to use another editor, e.g., *nano*, that’s fine with us. Whatever works for you!

The *vim* editor works differently than typical word processing systems like MS Word or Google Docs; basically, it does not *depend* on usage of the mouse. That is, you can do a lot of editing without having your fingers leave the keyboard – something that many programmers find to improve efficiency. We will not cover *vim* here; it is covered well by many websites and videos, and it is really up to you whether to use it at all, and, if you decide to use it, how much to invest in learning its features allowing efficient editing. One good resource for learning about it (or deciding if you want to) is the [Beginner’s Guide to Vim](https://linuxfoundation.org/blog/classic-sysadmin-vim-101-a-beginners-guide-to-vim/) article.

Anyway… use *vim* or your favorite editor, to open file *Passwords.txt* in the Lab0 directory; this is a simple ‘cleartext’ passwords file, where each line contains a user-id followed by a comma and a password. Find the password for user-id U44444 (by using an editor search mechanism, not by just scrolling…).

**Submit in the submission webserver and in your report in HuskyCT:** The password for user-id U44444 (starting with and including the “P”). You should always submit answers in both HuskyCT and submission webserver: *in redundancy we trust*.

**Question 5 (10 points): permissions, sudo, and chmod**

Unix systems are designed with the goal of ensuring security of the system and of the data of benign users, from a potentially attacking user of the system. Namely, Unix supports *access control*, i.e., restriction on the access and use of different resources (mostly, files and directories). The restrictions are based on two identifiers to which we map every username: *UID,* a *user-identifier,* and *GID,* a *group-identifier*; a user may be mapped to multiple groups, one of them considered `primary’*.* Both UIDs and GIDs are positive integers (up to some value, often 65535). Several usernames share the same *UID,* but then they are basically the same account, and have the same permissions.

Each resource is *owned* by a specific user (i.e., *UID*) and group (i.e., *GID*), which are identified in the output of the *ls -l* command (using the lower-case L option, *-l*). By default, these are the UID and the (primary) GID of the user that created the file. The *ls -l* command also lists the *permissions* of the resource. Unix considers three *types of permissions: read (r)*, *write (w)* and *execute (x)*. Unix allows setting different permissions to a resource, for three *types of users*: for the *owner* of the resource, for users in the same *group* (i.e., with the same GID) as the owner, and for other users.

Specifically, the output of *ls -l* begins, for each resource, with a string of 10 characters:

* Character 1 (the first character) can be either *d* (indicating this is a directory) or – (indicating this is a regular file).
* Character 2 can be either *r* (indicating read permission) or – (indicating read is not allowed), for the owner of the file.
* Character 3 can be either *w* (indicating write permission) or – (indicating write is not allowed), for the owner of the file.
* Character 4 can be either *x* (indicating eXecute permission) or – (indicating execution is not allowed), for the owner of the file.
* Characters 5-7 are mapped similarly to characters 2-4, except applying to users belonging to the same group as the owner of the file.
* Characters 8-10 are mapped similarly to characters 2-4, except applying to other users.

To change the permissions, use the *chmod* command. Use *man chmod* – or other method, e.g., some website - to learn how to change permissions, to files you own. It’s not too complex.

You may sometimes need to change permissions to files of other users or perform other tasks which are not permitted to `regular’ users. Such tasks can be done only by users which are defined as *superusers.* To issue a command which requires `superuser privilege’, such as *chmod* to a file owned by another user, prefix the command (e.g. chmod) by *sudo*, which stands for `super-user do’ (i.e., do as a super user). The *sudo* commands, e.g., *sudo chmod a+w,* will request re-entry of your password.

To complete this question:

* Use *ls -l* to find, in your *Lab0* directory, a file Q5 for which you do *not* have read permission.
* Change permissions as necessary, to be able to read the contents of this file.
* Use the *cat* command (or another way) to read the contents of this file (Q5).

**Submit in the submission webserver and HuskyCT:** the *contents* of Q5*.*

Note: to copy a file you may use the *cp* command, and to remove (delete) a file, you can use *rm*.

**Question 6 (10 points): grep, redirection, pipes and more**

Unix has many cool utilities and features that you may find useful, in this course and later on, when working on different Unix systems (or on systems inspired by Unix). In this question we will learn four: the *grep* and *more* commands, and the *redirection* and *pipes* mechanisms*.*

Let’s begin with *grep* and *more.* Both are nifty utilities. The *more* utility simply allows us to view the contents of a file (also done by *cat* utility), but with the added feature that if the output is too long for the screen, then *more* will display the file one screen at a time, allowing us to `step thru’ the file. Try, for example, *more Passwords* (in the Lab0 folder, of course).

The *grep* utility returns lines from a given text file that contain a given string or pattern. As usual, use *man grep* or another method to learn the basic usage of grep (it has some nice features).

We often may want to save the output of *grep* (or other command/program) into a file. One way to do it is using the *redirection mechanism,* which uses the > or >> symbols. For example, do *ls > lsfile* to have the output of *ls* saved in the file *lsfile*, instead of being printed on the screen. This is sometimes very useful! Try now *ls >> lsfile*; can you see what was the impact of using >> rather than >? We’re sure you can figure it out.

As a simple example of using redirection, consider the (common) case where *ls -l* outputs too many lines, as when the directory contains many files and sub-directories. To see the entire output, we can do *ls -l > temp* and then *more temp,* to `step through’ the output of *ls -l.* But this example isn’t so useful, simply since there aren’t that many files and directories in *Lab0.*

But here’s a challenge, where you may find *grep, pipes/redirection* and *more* useful: the file *Passwords* contains *one* line for a user whose name is of the form *U6xxxxxY,* where *xxxxx* is a string of five numerals. *Find it!*

There are other ways to efficiently find the answer. For example, you can also find it using a `smart’ search, where you specify the search string using a *regular expression (RegEx).* You can do it with *vim* (and several other editors). There are many things you can do in this course in different ways… and many tools in Unix. Still, it would be nice for you to (also?) find the answer as instructed, to see the use of combinations of *grep* and *pipes/redirection*.

**Submit in the submission webserver and HuskyCT:** The string you found in the format *U6xxxxxY.*

**Question 7 (10 points): python**

Finally, let’s run a Python program. Python programs are stored with either the extension .py (plain text) or the extension .pyc (bytecode, i.e., compiled Python)*.* The *Lab0* folder contains a Python program called Q7, using one of these formats (a .py or a .pyc). Find it and run it, using the *python3* Python interpreter (as *python3 name.py* or *python3 name.pyc,* as appropriate). Note that for *python3* to *read* the file, your user must have the corresponding (read) permission; this may not be the case originally, but you know how to check and correct such issues, right?

Once the program runs, it would output a string, starting with the letter S, followed by some digits.

**Submit in the submission webserver and HuskyCT:** The string outputted from the python program, Q7.py.

**Further comments**

There are many additional useful Unix commands and mechanisms; you are encouraged to read and explore. Some of these include the commands/utilities *cp,* for copying files and folders (lookup the recursive option, *-r*); the *ps* command, for displaying the active processes; and the *kill* command, to kill processes (lookup the -9 option, which is `forceful kill’).