Technical Assignment - for edrakh

This is the documentation for the assignment tasks.

Task 1:
For the first task, we just create a directory with a simple 'mkdir' command. Instead of using multiple 'mkdir' commands, we create them from a single command using a simple regular expression. Then, we use 'touch' to create the files and 'cat' anything to these files.
Note that I used '/usr/bin/env' because I am running NixOS on my machine now, so this is the way to go as NixOS has a different File Hierarchy Structure (FHS).
output:

```
user@user: ls project/*/
project/docs/:
README.md

project/src/:
main.py

project/test/:
test_main.py
```

Task 2:
check task2.sh

output:

```
user@user: hostname
devops-student
```

Task 3:
Using IP to print and then pass it to grep to get all lines that start with 'inet' then filter loopback and then get the IP only by using awk.

output:

```
user@user: ./task3.sh
192.168.8.14/24
```

Task 4:

Installing MySQL depends on whether you build it from the Git repo or install it from the package manager. Whatever method you choose, for me, I use NixOS. By adding these lines to the configuration and rebuilding the NixOS , MySQL server gets installed.

```
services.mysql = {
enable = true;
package = pkgs.mariadb;
};
```

the mariadb is a fork of mysql.

the task4.sh file use MySQL command with -e to run commands directly from the shell, and there is no outputs if the task run successfully.

Task 5:

Using mysqldump to backup the database.

Of course, it would be better if we encrypted the .sql file with something like 'gpg' if the disk is not encrypted, but for now this is unnecessary.

Task 6:

For the backup.sh, I used rsync, which is a pretty good tool called rsync. rsync has a lot of options and features like compressing, but we just need the -a, –partial, and –append. You can see the man page for more info. Also, I used -v, but it isn't mandatory.

Task 7:

Although systemd timers have more features, 'cron' is a very good tool that serves most use cases.

To make a 'cron' service in Linux, first, you need to install it, and then use 'crontab -e' to edit the 'cron' config file.

In NixOS, we can create them easily by adding them to the 'configuration.nix' file like this:

```
services.cron =
enable = true;
systemCronJobs = [
"0 0 * * * ababa /home/ababa/edrakSoftware/task6/backup.sh /home/ababa/important/ /home/ababa/cronbackup/"
];
};
```

Task 8:
Adding the line below to configure the SQL server to only accept connections from localhost:

```
bind-address=127.0.0.1
```

For the password checking, after some research, I found a good plugin for MariaDB called ’simple_password_check’. Check this link for more info:
see the link.
Installing the plugin can be done either by configuring ’my.cnf’ or by running:
INSTALL SONAME ’simple_password_check’;

Test output:

```
MysqlDB:: CREATE USER ’test_plugin’@’localhost’ IDENTIFIED BY ’bad’;
ERROR 1819 (HY000): Your password does not satisfy the current policy requirements (simple_password_check)
```

Task 9:
There are a lot of monitoring tools like a tool called btop, written in C++ and it has a good UI. It is configured at runtime. For more advanced usage, there are (nmon, written in C-, and glances), both have the ability to export the logs to external files for later analyzing.
for the gpu there is nice tool called nvtop

For the GPU, there is a nice tool called nvtop.
For SQL logs, add these lines to the my.cnf file to enable logging: general_log = 1
general_log_file = /home/ababa/.mysql/logs

Also, by default, you can see the logs using systemctl tools.

```
user@user: journalctl -u mysql.service
```

Last Task:
The security in Linux servers is heavily dependent on the use-case, but here are general best practices that suit almost every use-case.

- Remove, disable, or block everything you don’t use, such as open ports, apps, and services.
- Ensure there is a good password policy by using tools like CrackLib, which includes the ’cracklib-check’ tool. Alternatively, enforce a good password with any policy kit, such as libpam-cracklib, or utilize pam_pwquality library.
  Arch wiki link
- It’s not a bad idea to disable SSH password login and enable key login. In theory, if you have good passwords, this wouldn’t make any difference; however, it provides an additional layer of protection. You can also create an SSH IP whitelist that blocks any IP address not in the list from using SSH.
  Also, changing the SSH port will help with bots or blind brute-force attacks, but wouldn’t significantly impact an organized attack.
- Control and limit who can use ’sudo’ or ’su’, disabling ’su’ completely is a good idea to consider.
- Use a firewall like ’ufw’.
- Utilize AppArmor, which is a Linux kernel security module that allows system administrators to restrict program capabilities, such as network access, file permissions, and other capabilities.
- Compile or use a kernel with Security-Enhanced Linux (SElinux).
- Use a vulnerability scanner like Nessus or Lynis.
- Utilize an Intrusion Prevention Software (IPS) like Fail2ban, which will help with brute-force attacks.
- Set ’hidepid’ to 2 for the ’/proc’ folder, which can be a good option for some use-cases. However, in general, the default behavior is more suitable for most use-cases.
- For physical security, see Arch wiki.