PREPARE DIRECTORY STRUCTURE

The script takes care of creating the directory structure, and giving the permissions, in the prepare function. Here is how that structure looks like:

/opt/security/working
/opt/security/errors
/opt/security/bin

Also the script copies the PGP keys and SSH keys required to the security directory:

shaal25790.asc, shaal25790_public.asc, shaal25790.id, shaal25790_public.id

EXECUTION OF THE SCRIPT

To execute the script we navigate to the directory where it is stored

```
cd /code/
```

Then we call the script, sending parameters like this:

```
./tth-shaal25790.sh "https://shady-cnd.no" "shady-logs.no" "shaal25790"
```

Parameter #1: server url where the IoC file is stored.

Parameter #2: the upload url, to upload the report file.

Parameter #3: the identity of the user.

AUTOMATE SCRIPT EXECUTION

This can be done using the cron jobs. This system enables us to schedule tasks/jobs to execute automatically in specific times and repeat in specific periods. Each user has own crontab where they can schedule jobs, which is stored under /var/spool/cron/crontabs

Example: a cron job that runs the command curl http://www.google.com every Tuesday at 5:30 PM: 30 17 * * 2 curl http://www.google.com

Now we can create an automated job for our script, to be automatically executed every day at 2 am. Using the command crontab -e a text editor opens a tmp file, where you place your lines. This is for safety, to prevent you from risking all your acrual crontabs if the new one fails due to syntax error, so to check your new crontab for errors, if there are no errors, your actual crontab will be updated. You should never touch your actual crontab directly, use **crontab -e** instead. Then we add these lines:

MAILTO="shaal25790@stud.noroff.no"

0 2 * * * /bin/bash /code/tth-IOC.sh

By default, cron will send an email to the owner of the crontab whenever a cron job produces output. We can change that default email by defining this variable in the cronjob:

the 0 represents minuts, 2 represents hours, then day of month (1-31), month (1-12), day of the week (0-7 where both 0 & 7 represent sunday)

Save & exit the editor, so the cronjob is created for your user. Make sure the script has the execute permissions: sudo chmod +x /code/tth-IOC.sh

DOWNLOAD FILES FROM URL

To download files as mentioned in the specifications, we need to create a website that is hosted on the localhost of Ubuntu. The sever name will be the url passed as the first parameter to the script, and the files are located in the base url.

First, check if Apache is installed using this:
apache2 -v

If it is installed the apache version should appear. If you get a message saying “apache2 command is not found”, then you should install it yourself using:

sudo apt update -y

sudo apt install apache2 -y

After installation is done, the directory /var/www should be created, if not, just create it and give permissions and ownership to the user www-data using:

sudo mkdir /var/www/shady-cnd

sudo chown www-data:www-data /var/www/shady-cnd

sudo chmod 755 /var/www/shady-cnd

Add the line “127.0.0.1 shady-cnd.no” to the /etc/hosts

ACCESS VIA HTTPS

The website is required to be accessed via HTTPS. So, we install **SSL Certificate**. Create a new OpenSSL configuration file with the following content: openssl.cnf

```
[req]
distinguished_name = req_distinguished_name
x509_extensions = v3_req
prompt = no
[req_distinguished_name]
CN = shady-cnd.no
[v3_req]
keyUsage = keyEncipherment, dataEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1 = shady-cnd.no
IP.1 = 127.0.0.1
```

This configuration file sets up a certificate for localhost that's valid for IP address 127.0.0.1.
Then generate a self-signed certificate using this command with the previous config file:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout shady-cnd.key -out shady-cnd.crt -config openssl.cnf
```

this generates shady-cnd.crt as the certificate and shady-cnd.key as private.key, so we use them in apache configuration file:

```
sudo nano /etc/apache2/sites-available/shady-cnd.conf
```

```
<VirtualHost *:80>
   ServerAdmin shaal25790@stud.noroff.no
   ServerName shady-cnd.no
   ServerAlias www.shady-cnd.no
   DocumentRoot /var/www/shady-cnd/
   ErrorLog ${APACHE_LOG_DIR}/error.log
   CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>

<VirtualHost *:443>
   ServerName shady-cnd.no
   DocumentRoot /var/www/shady-cnd
   SSLEngine on
   SSLCertificateFile /code/shady-cnd.crt
   SSLCertificateKeyFile /code/shady-cnd.key
   <Directory /var/www/shady-cnd/>
      AllowOverride All
   </Directory>
</VirtualHost>
```

Then enable the SSL Module and the website again then restart apache:

```
sudo a2enmod ssl
sudo a2ensite shady-cnd
sudo systemctl reload apache2
```

But here can encounter an error caused by self-signed certificates are not trusted by default by most systems, since they're not issued by a recognized Certificate Authority (CA). So we solve this by copying the certificate to the Trusted CA Directory, then update the ca Certificate Directory:

```
sudo cp /code/shady-cnd.crt /usr/local/share/ca-certificates/
sudo update-ca-certificates
```

Now, you should be able to download https://shady-cnd.no/IOC-20231122.ioc via HTTPS.

NOTICE: don't forget to create the directory /var/www/uploads also, since it is checked in the IoC files.

VALIDATE FILES IN /sbin, /bin, /usr/sbin, /usr/bin AND /usr/lib MATCH THE HASHES IN THE SYSTEM PACKAGE DATABASE

To do that I use the **dpkg** command, since it is the built in alternative that can do the job. However, the command itself does not provide a build in way to verify the integrity of all files installed in the specified directories. So, we use the dpkg -V command to check the integrity of the installed packages, but it doesn't check all files and can't specify directories. So, we use a combination of commands to achieve that. This way, we can run away with this, without having to install any tools/utilities on Ubuntu (ex. debsums and aide), to satisfy the specifications.

UPLOAD

Check if the user you are connecting with, does exists on the remote server:

```
cat /etc/passwd | grep shaal25790
```

If the user doesn't exist on the remote server (which is localhost itself in my own case), you need to create it, and create a password for it:

```
sudo useradd -m shaal25790
```

```
sudo passwd shaal25790
```

it is important our ssh public key is present on the remote server, so that it doesn't ask for the password every time we use ssh. To add our ssh public key to the authorized_keys of the remote server:

```
sudo chmod 600 /home/shaal25790/.ssh/authorized_keys # very important 600
```

```
ssh-copy-id -i ~/.ssh/id_rsa.pub ${USERNAME}@${UPLOAD_URL}
```

Also inside **/etc/ssh/sshd_config** should you find:

```
PubkeyAuthentication yes
```

Add this line to the **/etc/hosts**

```
127.0.0.1        shady-cnd.no
```

All of these configurations are for local upload-server. However, in our assignment, the environment should have taken care of these already, as mentioned in the specification.