SSH Commands Cheat Sheet

STATIONX
THE CYBER SECURITY COMPANY

What Is SSH?

SSH (short for “Secure Shell” or “Secure Socket Shell”) is a network protocol for accessing network services securely over unsecured networks. It includes the suite of utilities implementing it, such as:

- **ssh-keygen:** for creating new authentication key pairs for SSH;
- **SCP** (Secure Copy Protocol): for copying files between hosts on a network;
- **SFTP** (Secure File Transfer Protocol): for sending and receiving files. It’s an SSH-secured version of FTP (File Transfer Protocol), and it has replaced FTP and FTPS (FTP Secure) as the preferred mechanism for file sharing over the Internet.

An SSH server, by default, listens for connections on the standard Transmission Control Protocol (TCP) port 22. Your applications may listen for SSH connections on other ports.

SSH lets you securely manage remote systems and applications, such as logging in to another computer over a network, executing commands, and moving files from one computer to another. An advanced SSH functionality is the creation of secure tunnels to run other application protocols remotely.

Basic SSH Commands

The following are fundamental SSH commands. Commit as many to memory as you can.

Command	Description
`ssh`	**Connect to a remote server**
`ssh pi@raspberry`	Connect to the device `raspberry` on the default SSH port 22 as user `pi`

`ssh pi@raspberry -p 3344`	Connect to the device `raspberry` on a specific port 3344 as user `pi`
`ssh -i /path/file.pem admin@192.168.1.1`	Connect to `root@192.168.1.1` via the key file `/path/file.pem` as user `admin`
`ssh root@192.168.2.2 'ls -l'`	Execute remote command `ls -l` on `192.168.2.2` as user `root`
`$ ssh user@192.168.3.3 bash < script.sh`	Invoke the script `script.sh` in the current working directory spawning the SSH session to `192.168.3.3` as user `user`
`ssh friend@Best.local "tar cvzf - ~/ffmpeg" > output.tgz`	Compress the `~/ffmpeg` directory and download it from a server `Best.local` as user `friend`
`ssh-keygen`	**Generate SSH keys (follow the prompts)**
`ssh-keygen -F [ip/hostname]`	Search for some IP address or hostname from `~/.ssh/known_hosts` (logged-in host)
`ssh-keygen -R [ip/hostname]`	Remove some IP address or hostname from `~/.ssh/known_hosts` (logged-in host)
`ssh-keygen -f ~/.ssh/filename`	Specify file name
`ssh-keygen -y -f private.key > public.pub`	Generate public key from private key
`ssh-keygen -c -f ~/.ssh/id_rsa`	Change the comment of the key file `~/.ssh/id_rsa`
`ssh-keygen -p -f ~/.ssh/id_rsa`	Change passphrase of private key `~/.ssh/id_rsa`
`ssh-keygen -t rsa -b 4096 -C "my@email.com"`	Generate an RSA 4096-bit key with "`my@email.com`" as a comment: `-t`: Type of key (`rsa, ed25519, dsa, ecdsa`) `-b`: The number of bits in the key `-C`: Provides a new comment
`scp`	**Copy files securely between servers**
`scp user@server:/folder/file.ext dest/`	Copy from remote to local destination `dest/`
`scp dest/file.ext user@server:/folder`	Copy from local to remote
`scp user1@server1:/file.ext user2@server2:/folder`	Copy between two different servers
`scp user@server:/folder/* .`	Copies from a server folder to the current folder on the local machine
`scp -r`	Recursively copy entire directories
`scp -r user@server:/folder dest/`	Copy the entire folder to the local destination `dest/`
`scp user@server:/folder/* dest/`	Copy all files from a folder to the local destination `dest/`
`scp -C`	Option to compress data
`scp -v`	Option to print verbose info

`scp -p`	Option to preserve the last modification timestamps of the transferred files
`scp -P 8080`	Option to connect to remote host port 8080
`scp -B`	Option for batch mode and prevent you from entering passwords or passphrases
`sftp`	**Securely transfer files between servers**
`sftp -p`	Option to preserve the last modification timestamps of the transferred files
`sftp -P 8080`	Option to connect to remote host port 8080
`sftp -r`	Recursively copy entire directories when uploading and downloading. SFTP doesn't follow symbolic links encountered in the tree traversal.

SSH Configurations and Options

Have you ever wondered how SSH remembers your login credentials for various machines? This section is a brief reference on how to do so.

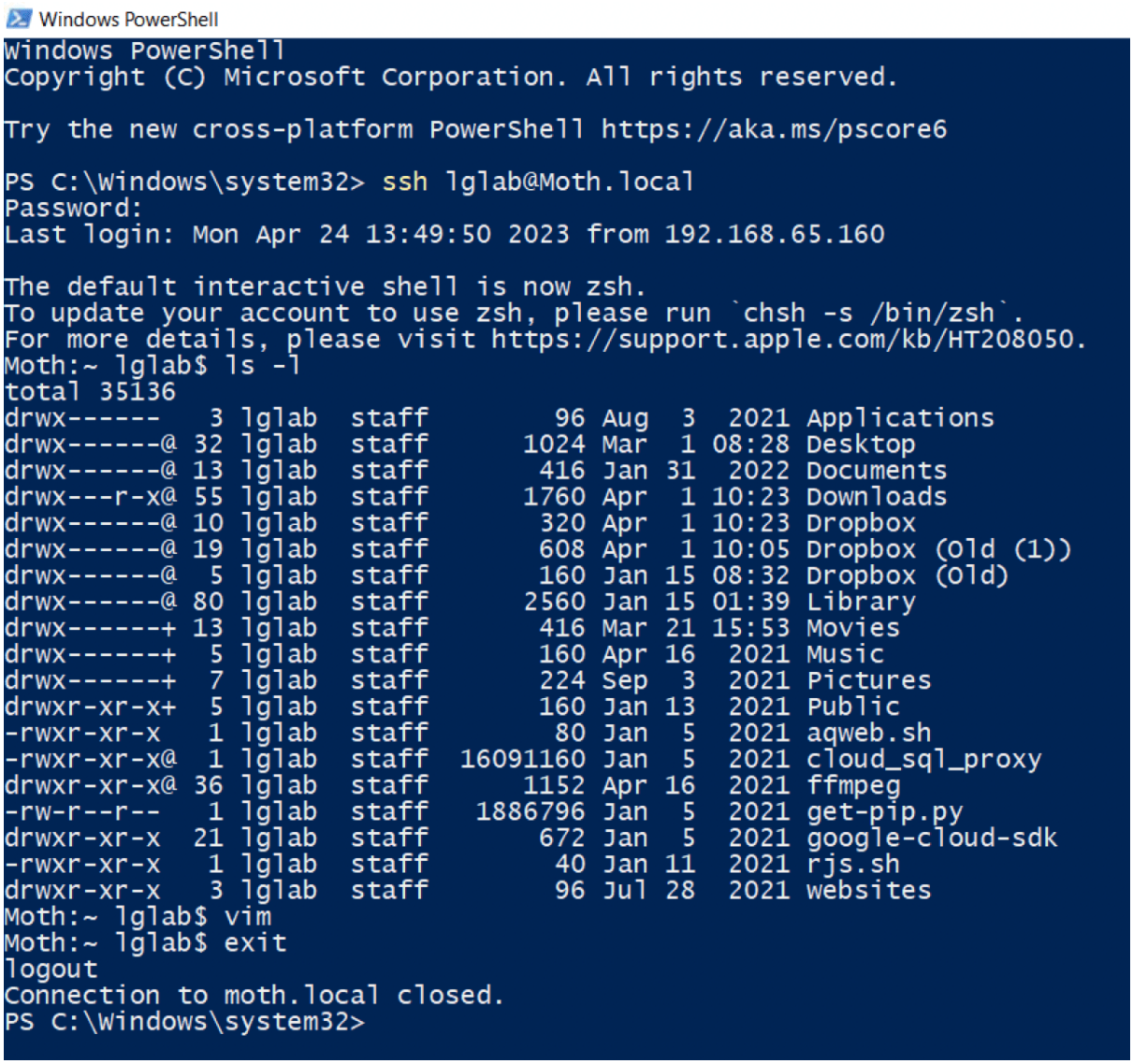
Command	Description
`man ssh_config`	**Open OpenSSH SSH client configuration files.** This manual lists all the OpenSSH parameters you can change.
`cat /etc/ssh/ssh_config | less`	View your OpenSSH client system-wide configuration file
`cat /etc/ssh/sshd_config | less`	View your OpenSSH server system-wide configuration file; the "d" stands for the server "daemon"
`cat ~/.ssh/config | less`	View your SSH client user-specific configuration file
`cat ~/.ssh/id_{type} | less`	View your SSH client private key; `type` is any of `rsa, ed25519, dsa, ecdsa`.
`cat ~/.ssh/id_{type}.pub | less`	View your SSH client public key; `type` is any of `rsa, ed25519, dsa, ecdsa`.
`cat ~/.ssh/known_hosts | less`	View your SSH client logged-in hosts
`cat ~/.ssh/authorized_keys | less`	View your SSH client authorized login keys
`ssh-agent`	**Hold private SSH keys used for public key authentication (RSA, DSA, ECDSA, Ed25519)**
`ssh-agent -E fingerprint_hash`	Specify the hash algorithm used when displaying key fingerprints. Valid `fingerprint_hash` options are `sha256` (default) and `md5`.
`ssh-agent -t lifetime`	Set up a maximum `lifetime` for identities/private keys, overwritable by the same setting in `ssh-add`. Examples of `lifetime`:

	• `600` = 600 seconds (10 minutes) • `23m` = 23 minutes • `1h45` = 1 hour 45 minutes
`ssh-add`	**Add SSH keys to the `ssh-agent`**
`ssh-add -l`	List your private keys cached by `ssh-agent`
`ssh-add -t lifetime`	Set up a maximum `lifetime` for identities/private keys. Examples of `lifetime`: • `600` = 600 seconds (10 minutes) • `23m` = 23 minutes • `1h45` = 1 hour 45 minutes
`ssh-add -L`	List the public key parameters of all saved identities
`ssh-add -D`	Delete all cached private keys
`ssh-copy-id`	**Copy, install, and configure SSH keys on a remote server**
`ssh-copy-id user@server`	Copy SSH keys to a `server` as a `user`
`ssh-copy-id server1`	Copy to some alias server `server1` with the default login
`ssh-copy-id -i ~/.ssh/id_rsa.pub user@server`	Copy a specific key to a `server` as a `user`

Remote Server Management

The operating systems of SSH servers are mostly Unix/Linux, so once you've logged in to a server via SSH, the following commands are largely the same as their counterparts in Unix/Linux. Check out our Unix commands cheat sheet and Linux command line cheat sheet for other file management commands applicable to SSH.

Command	Description
`cd`	Change the current working directory
`kill`	Stop a running process
`ls`	List files and directories
`mkdir`	Create a new directory
`mv`	Move files or directories
`nano`	Edit a file in the terminal using Nano
`ps`	List running processes
`pwd`	Display the current working directory
`tail`	View the last few (10, by default) lines of a file
`top`	Monitor system resources and processes
`touch`	Create a new file or update the timestamp of an existing file
`vim`	Edit a file in the terminal using Vim
`exit`	Close the SSH session



Using PowerShell to access a lab account on a network computer via SSH on Windows 10

Advanced SSH Commands

This table lists some complex SSH utilities that can help with network administration tasks: SSH File System (SSHFS), data compression, and X11 forwarding.

To conduct X11 forwarding over SSH, do these three things:

1. Set up your client (`~/.ssh/config`) to forward X11 by setting these parameters:
   ```
   Host *
    ForwardAgent yes
    ForwardX11 yes
   ```
2. Set up your server (`/etc/ssh/sshd_config`) to allow X11 by setting these parameters:
   ```
   X11Forwarding yes
   X11DisplayOffset 10
   X11UseLocalhost no
   ```
3. Set up X11 authentication on your server by installing `xauth`.

Command	Description
`sshfs`	Mount a remote server's file system on a local directory. Remember to install this program onto your machine before use. Example installation commands: • `sudo apt install sshfs # Ubuntu/Debian` • `sudo yum install fuse-sshfs # CentOS` Learn to install apps on various Linux distributions here.
`ssh -C hostname`	Compress SSH traffic to improve performance on slow connections. Alternatively, insert `Compression yes` into your SSH configuration files.
`ssh -o "Compression yes" -v hostname`	An alternative method to compress SSH traffic to improve performance on slow connections. This is the same as inserting `Compression yes` into your SSH configuration files.
`ssh -X user@server`	Enable X11 forwarding over SSH: forward graphical applications from a remote `server` as a `user` to a local machine.
`ssh -o ForwardX11=yes user@server`	Enable X11 forwarding over SSH: forward graphical applications from a remote `server` as a `user` to a local machine.
`ssh -x`	Disable X11 forwarding
`ssh -Y`	Enable trusted X11 forwarding. This option is riskier than `ssh -X` as it forwards the entire display of the SSH server to the client.

Tunneling

These SSH command line options create secure tunnels.

Options	Description	Syntax / Example

`-L`	Local port forwarding: forward a port on the local machine (SSH client) to a port on the remote machine (`ssh_server` as `user`), the traffic of which goes to a	`ssh user@ssh_server -L local_port:destination:remote_port`
		
		`# Example`

	port on the `destination` machine. The parameters `local_port` and `remote_port` can match.	`ssh root@192.168.0.1 -L 2222:10.0.1.5:3333`

`-J`	ProxyJump; ensure that traffic passing through the intermediate/bastion hosts is always encrypted end-to-end.
	
	ProxyJump is how you use bastion hosts to connect to a remote host with a single command.
		`ssh -J proxy_host1 remote_host2`
		
		`ssh -J user@proxy_host1 user@remote_host2`
		
		`# Multiple bastion hosts/jumps`
		`ssh -J user@proxy_host1:port1,user@proxy_host2:port2 user@remote_host3`

`-R`	Remote port forwarding: forward a port `remote_port` on the remote machine (`ssh_server` as `user`) to a port on the local machine (SSH client), the traffic of which goes to a port `destination_port` on the `destination` machine.
	
	An empty `remote` means the remote SSH server will bind on all interfaces.
	
	Additional SSH options in the example:
	`-N`: don't execute remote commands; useful for dedicated port forwarding
	`-f`: run SSH in the background.
		`ssh -R [remote:]remote_port:destination:destination_port [user@]ssh_server`
		
		`# Example`
		`ssh -R 8080:192.168.3.8:3030 -N -f user@remote.host`

`-D`	Set up a SOCKS Proxy to tunnel traffic from a `remote_host` on which you're the `user` to a `local_port_number`.
	
	Additional SSH options in the example:
	`-q`: quiet mode; don't output anything locally
	`-C`: compress data in the tunnel, save bandwidth
		`ssh -D local_port_number user@remote_host`
		
		`# Example`
		`ssh -D 6677 -q -C -N -f me@192.168.5.5`

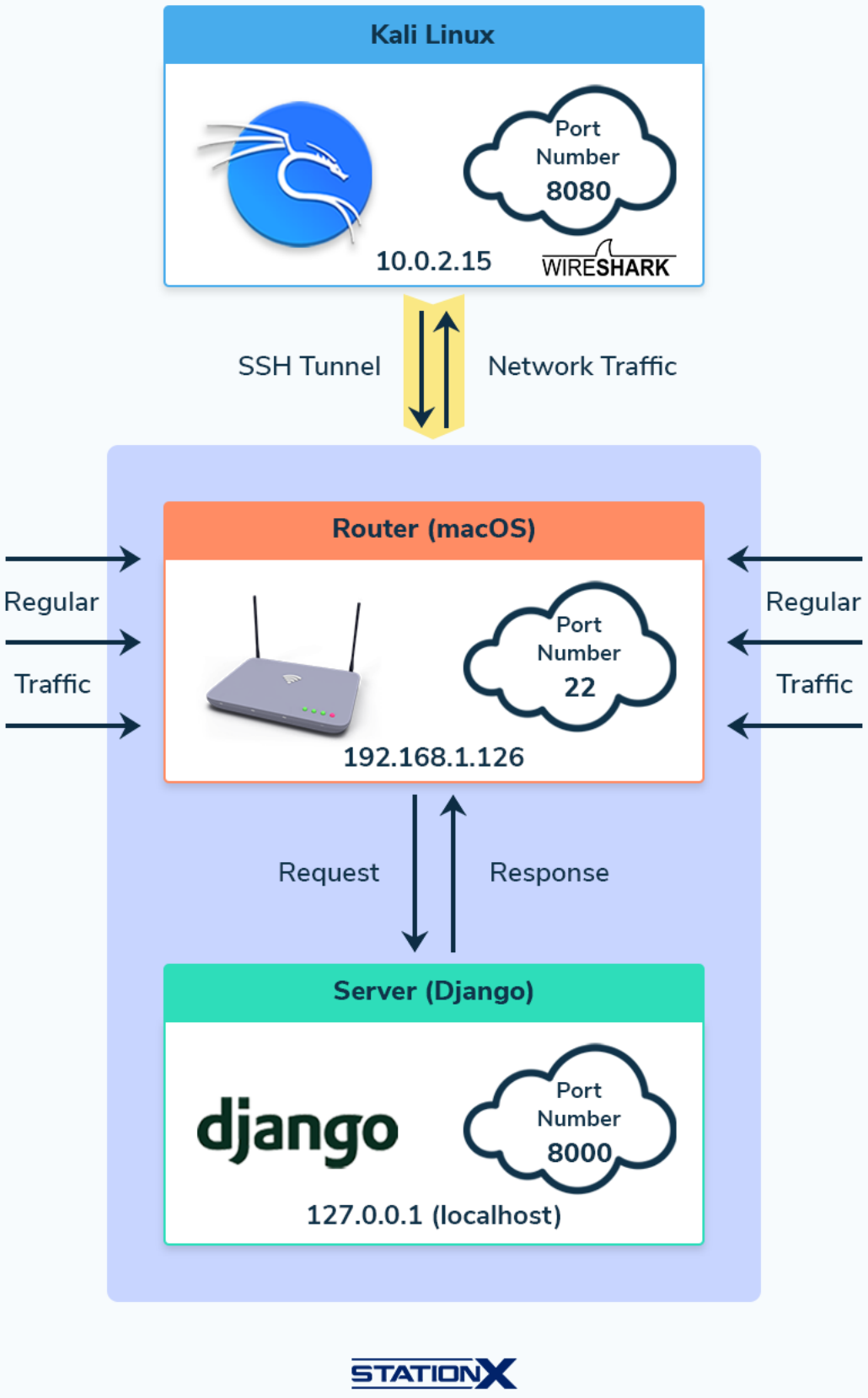
	`-N`: don't execute remote commands; useful for dedicated port forwarding `-f`: run SSH in the background.	

SSH Tunneling Demonstration

Let's show you two ways to pipe traffic from your router into Wireshark and monitor your network activity. The first demonstration involves installing programs onto a system used as a router; the second, without.

Using Django

SSH Tunneling Demo

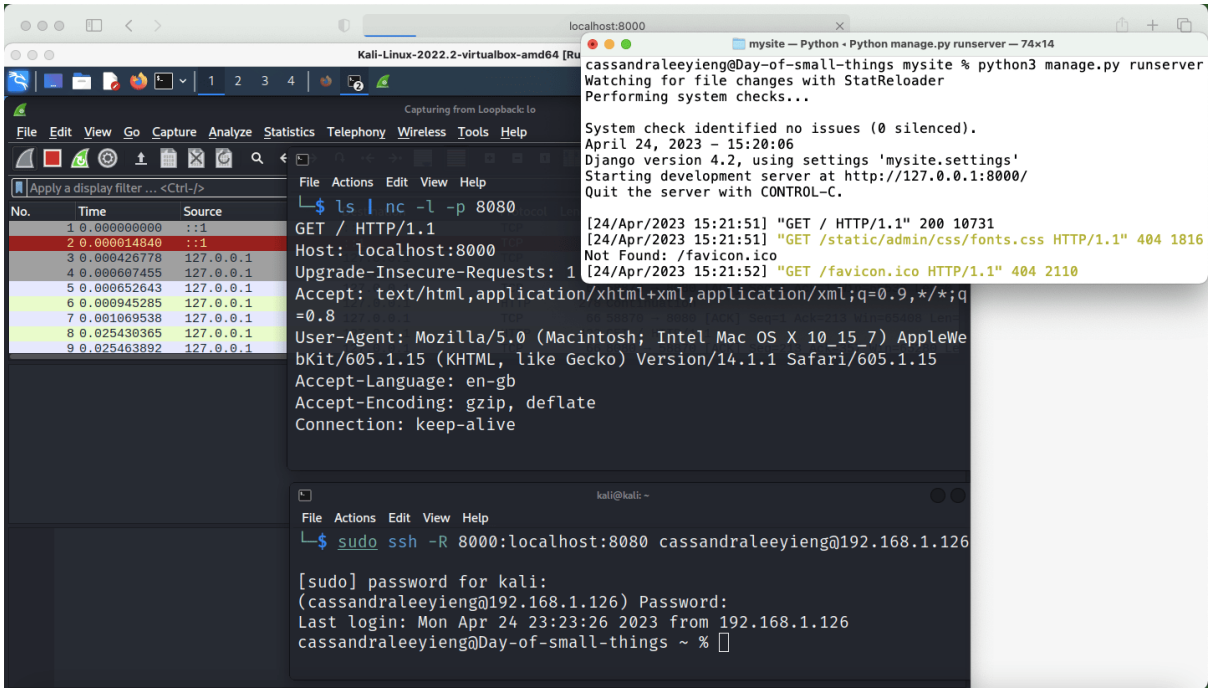


STATIONX

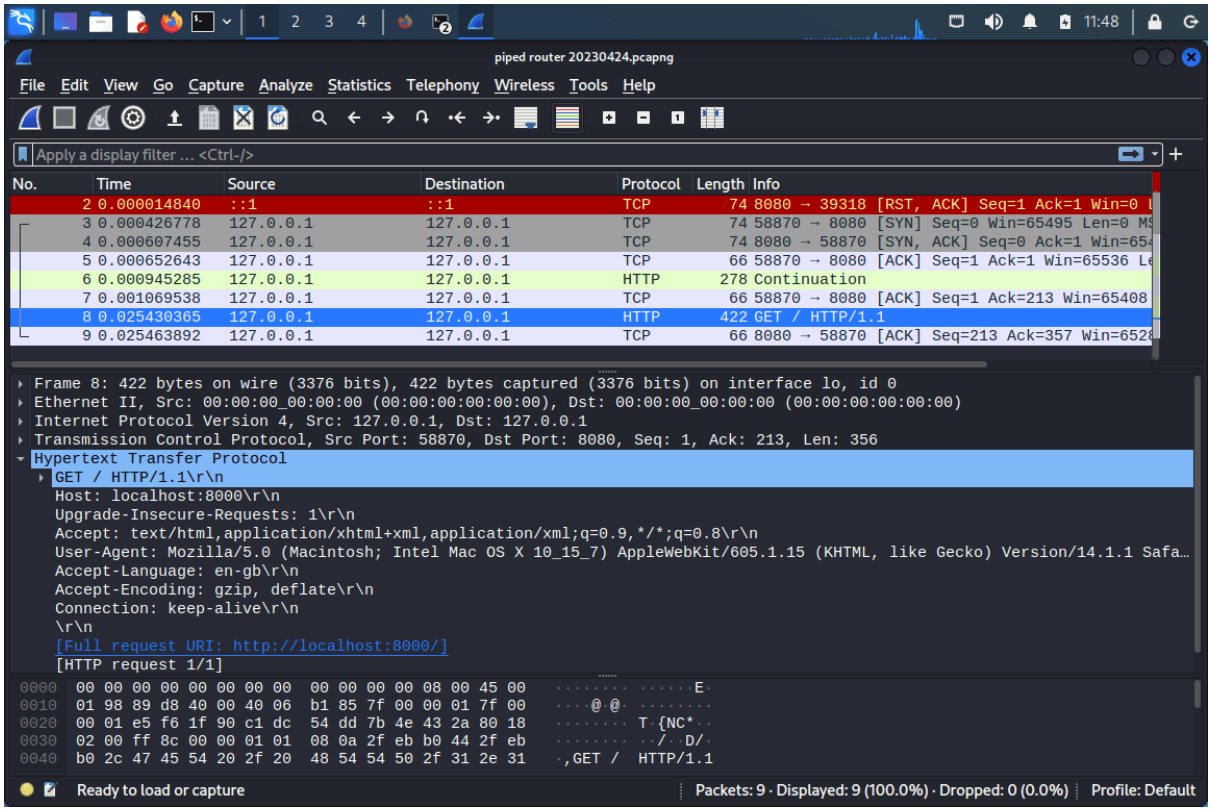
As a demonstration, we're piping traffic from a router into Wireshark, so that we can monitor live web traffic through an SSH tunnel. (The router below is a macOS computer hosting a Kali Linux virtual machine using the Wireshark instance installed on the latter.)

The setup is as follows:

1. **On the router:** Enable remote access via SSH. (NOTE: On the macOS system, go to **System Preferences** > **Sharing** > turn on **Remote Login** and note the login username and hostname. For your router setup, check your specific manufacturer's guide to enable remote access via SSH.)
2. **On the router:** Install Python Django and start up the Django template server on http://127.0.0.1:8000 using the Terminal command string `django-admin startproject mysite; cd mysite; python manage.py runserver` (or `python3 manage.py runserver`). Note the Django web app uses port 8000.
3. **On Kali Linux:** execute this command to listen on port 8080: `ls | nc -l -p 8080`
4. **On Kali Linux:** execute this command in a different Terminal tab/window. Below, `8000` is the router's Django port, `8080` is the Kali Linux listening port on `localhost`, and the command involves remote port forwarding (`-R`): `sudo ssh -R 8000:localhost:8080 user@router_ip`
5. **On Kali Linux:** start Wireshark and select the loopback interface (`lo`) as the capture device. Wireshark should be sniffing packets on `lo` now.
6. **On the router:** visit http://127.0.0.1:8000 in a web browser. (Note `localhost` and 127.0.0.1 are equivalent.) The Django server wouldn't load; it freezes instead because of the rerouted traffic.
7. **On Kali Linux:** You should expect the following results:

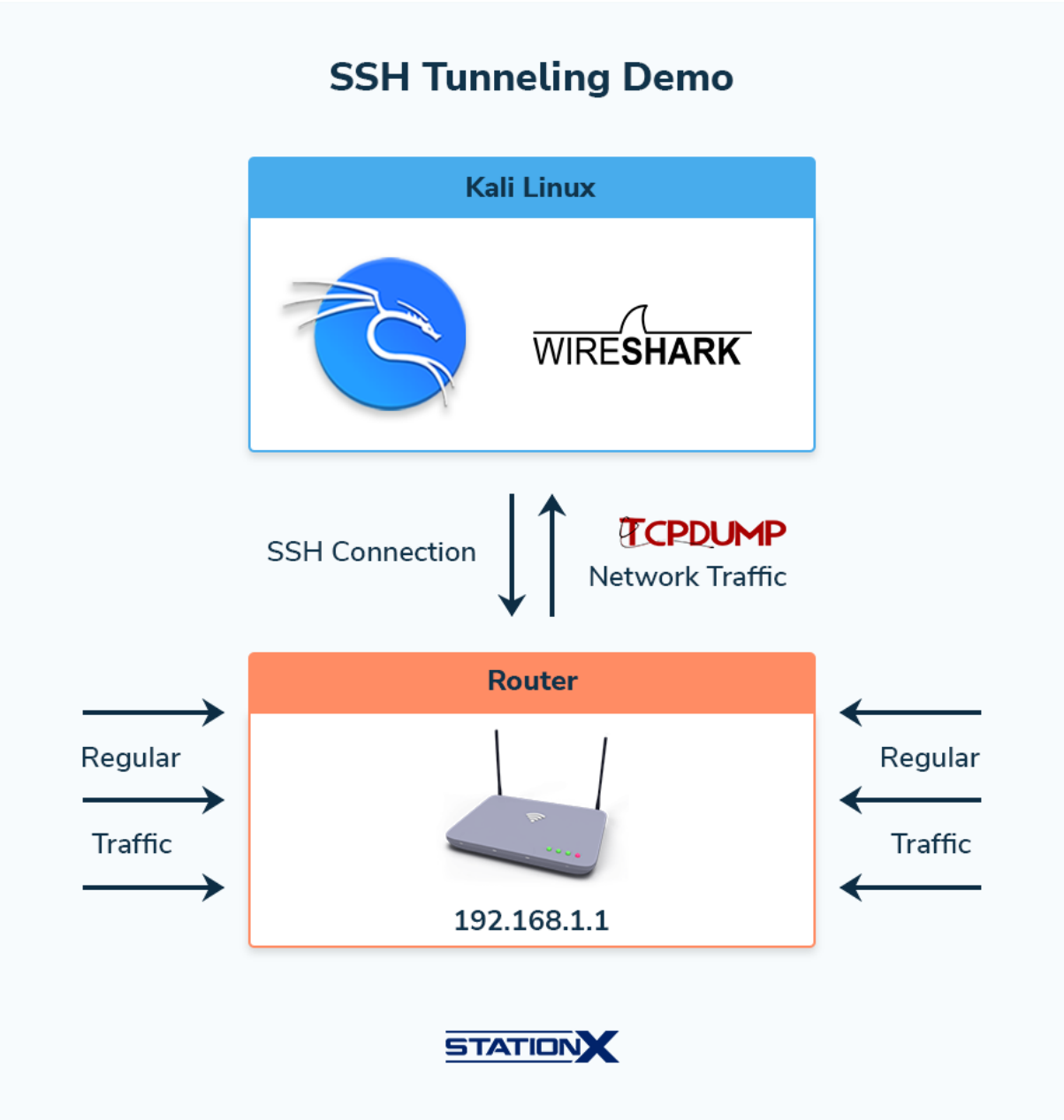


Piping traffic of Django web app http://127.0.0.1:8000 on the macOS router into the Wireshark instance on Kali Linux



Wireshark HTTP packet corresponding to the Django web app on the router

Using tcpdump



The following is an alternative method for capturing remote web traffic passing through a router.

In Kali Linux, you'll log in to your router via SSH, capture packets with the command-line packet capturing tool tcpdump, and pipe the traffic into Wireshark.

Here is the required command with the option flags explained:

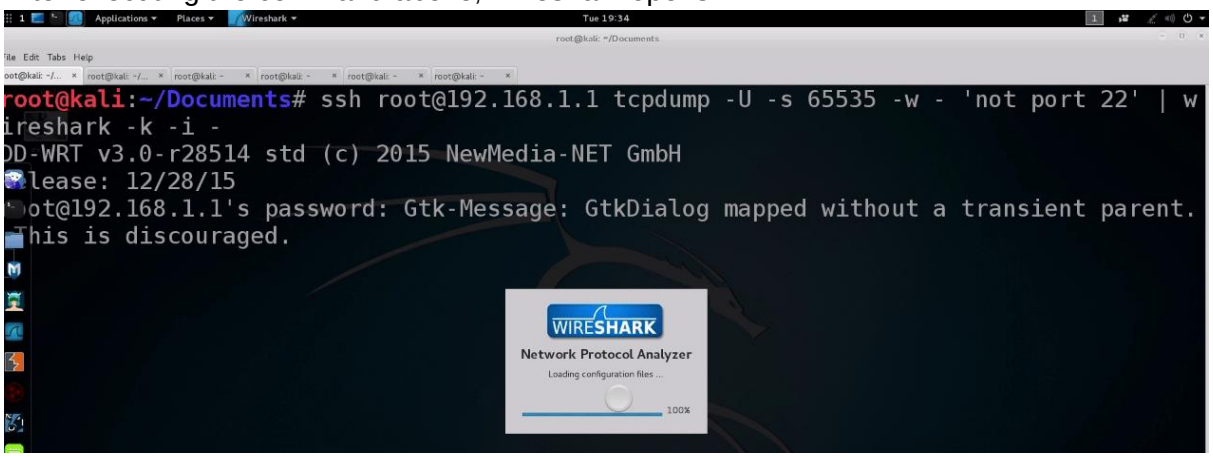
```
ssh [username]@[hostname/ip] tcpdump -U -s 65525 -w - 'not port 22'
| wireshark -k -i -
```

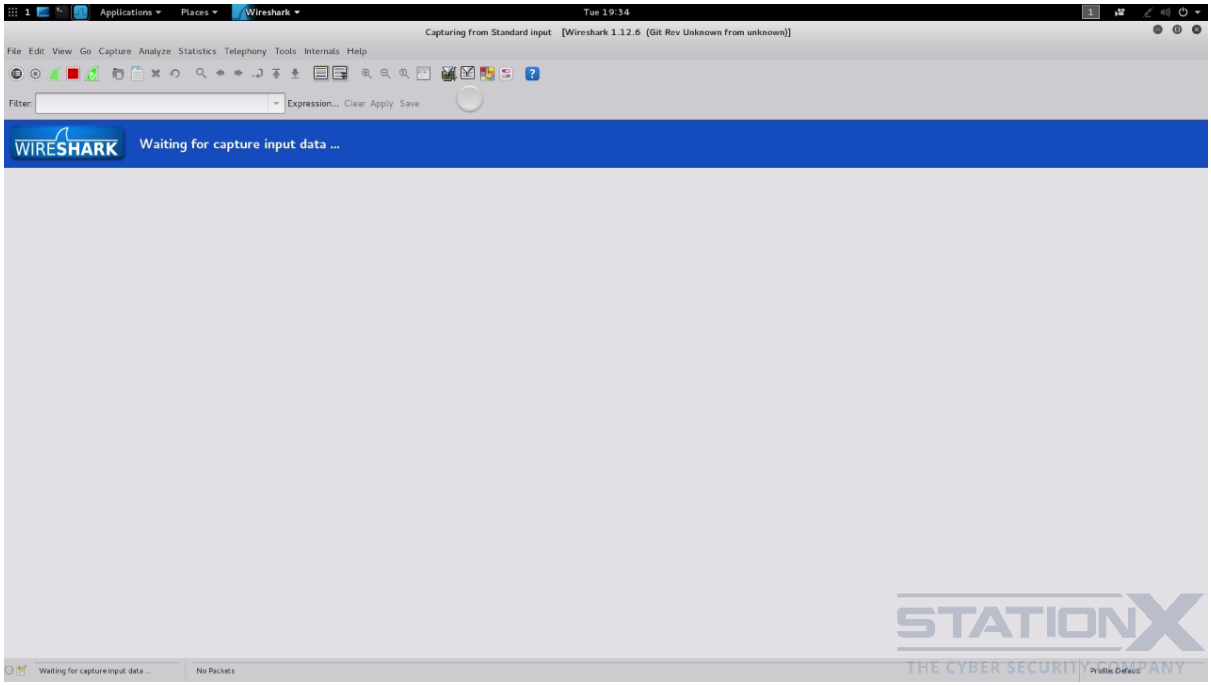
- `-U`: No buffering. Produce real-time output.

- `-s 65525`: Grab 65525 bytes of data from each packet rather than the default of 262144 bytes. 65525 is the maximum transmission unit of a Point-to-Point Protocol packet that Wireshark can handle. Adjust this number as you see fit.
- `-w`: Write each packet to the output packet capture file on your local disk in Kali Linux. Combining `-U` and `-w` means tcpdump writes to your output file as the packets pour in, rather than until the memory buffer fills up.
- `'not port 22'`: This is to prevent tcpdump from echoing the SSH packets sent between your machine and the router.
- `-k -i -`: Start the capture immediately and use the command before the pipe character (`|`) as the capture interface.

Example of piping router traffic to Wireshark via tcpdump

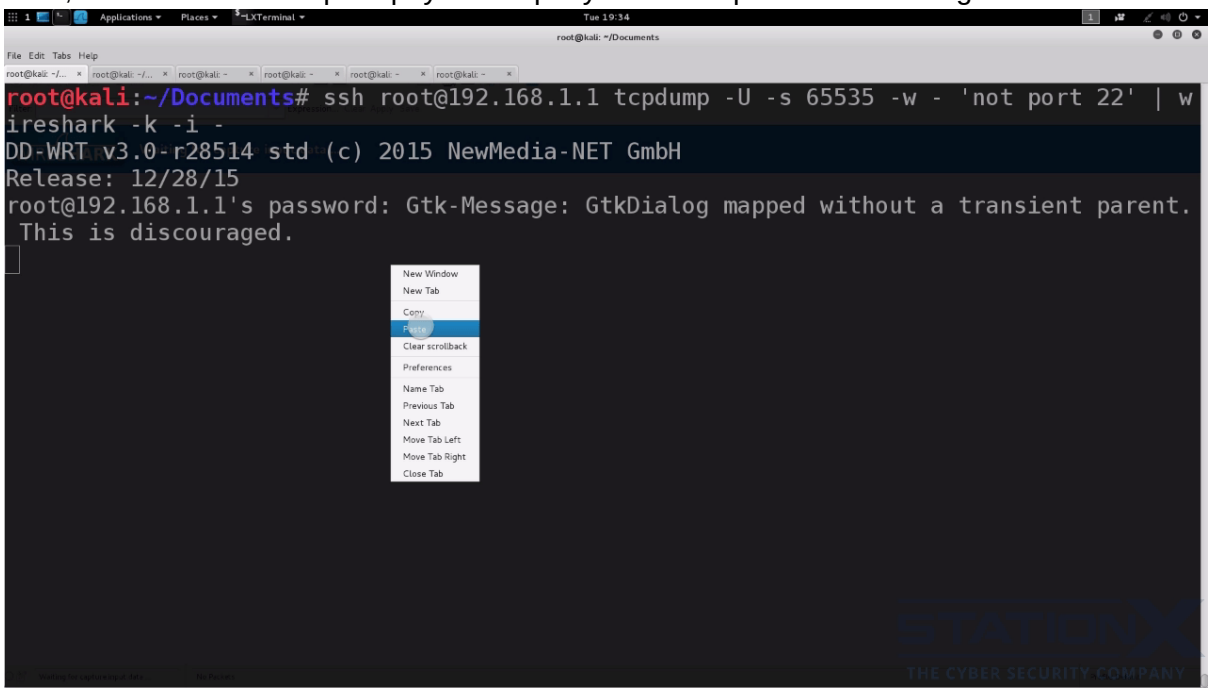
After executing the command above, Wireshark opens:



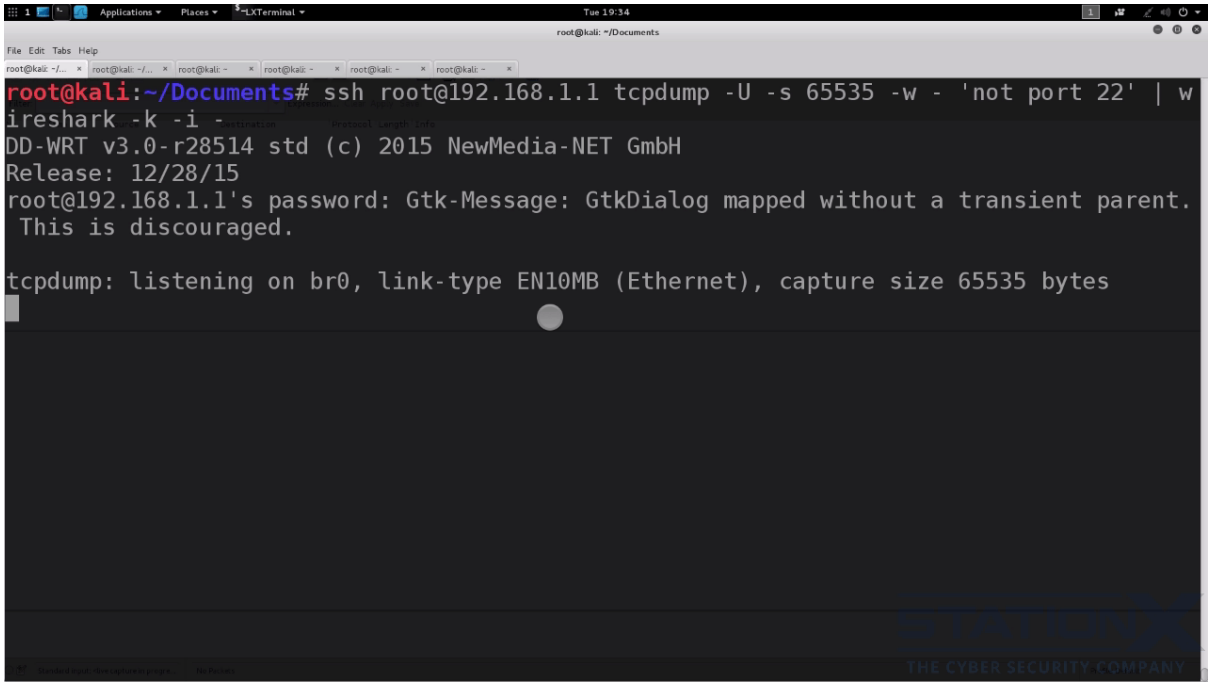


Wireshark triggered

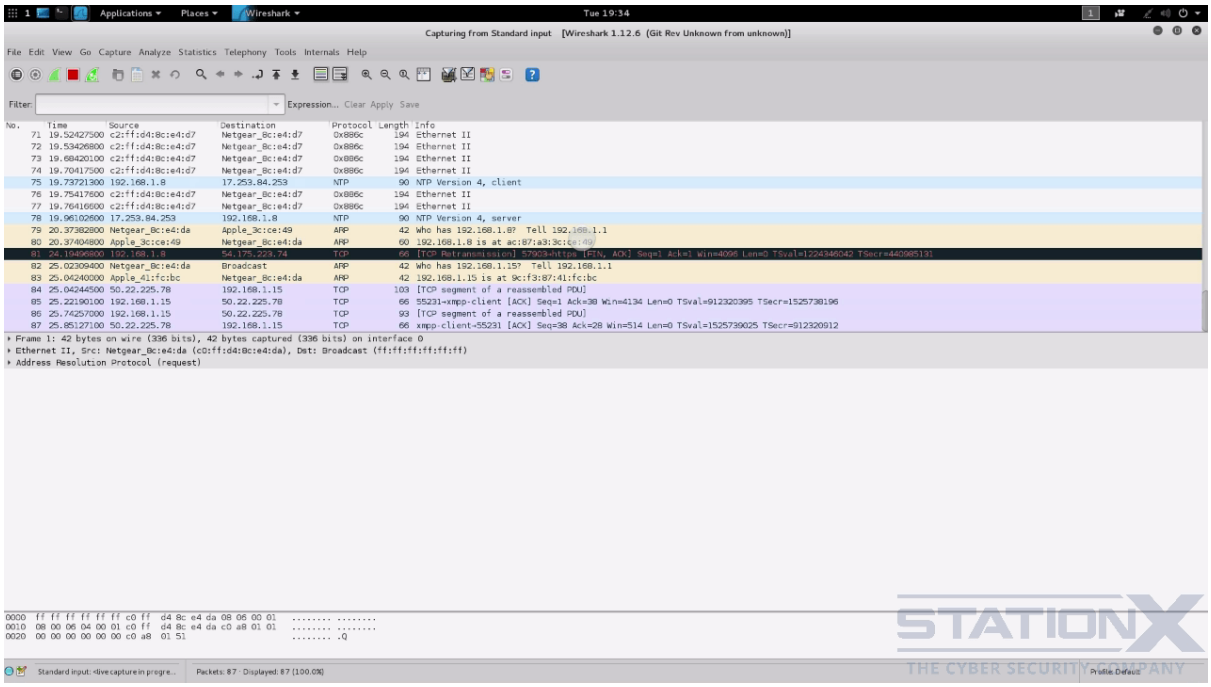
Next, the SSH client will prompt you to input your router password. Pasting it suffices:



SSH login successful. Now, tcpdump packet capture begins:



Meanwhile, Wireshark receives the piped traffic from tcpdump:



That's it.

Conclusion

We have covered SSH, SCP, SFTP, SSH configuration commands such as `ssh-agent`, `ssh-add`, and `ssh-copy-id`, and various SSH tunneling commands.

Here are some tips for using SSH more efficiently and securely:

- Disable X11 and TCP forwarding because attackers can use such weaknesses to access other systems on your network. Change the options on `sshd_config` to be `AllowTcpForwarding no` and `X11Forwarding no`.
- Change the default options on `sshd_config`, such as changing the default port from 22 to another number.
- Authenticate clients using SSH certificates created with `ssh-keygen`.
- Use a bastion host with the help of tunneling commands.
- Restrict SSH logins to specific IPs, such as adding user filtering with the `AllowUsers` option in `sshd_config`.

Thanks to its security measures and the ubiquity of networking tasks, SSH is indispensable for computer data communications. Hence every student and professional in IT and cyber security needs a working knowledge of SSH commands, and we hope this SSH cheat sheet is a good starter or refresher for you.

To learn more about SSH and secure network administration, check out the following courses from us:

- The Complete Cyber Security Course! Volume 3: Anonymous Browsing
  - https://courses.stationx.net/p/the-complete-cyber-security-course-anonymous-browsing
- Secure Shell Fundamentals - Learn SSH By Configuring It
  - https://courses.stationx.net/p/secure-shell-fundamentals-learn-ssh-by-configuring-it
- Linux Security and Hardening, The Practical Security Guide
  - https://courses.stationx.net/p/linux-security-and-hardening-the-practical-security-guide