Linux Fundamentals

Introduction:-

Linux: A Big contribution of Linus Torvalds since 1991.

Philosophy:

Linux follows 5 core principles:

Principle	Description
`Everything is a file`	All configuration files for the various services running on the Linux operating system are stored in one or more text files.
`Small, single-purpose programs`	Linux offers many different tools that we will work with, which can be combined to work together.
`Ability to chain programs together to perform complex tasks`	The integration and combination of different tools enable us to carry out many large and complex tasks, such as processing or filtering specific data results.
`Avoid captive user interfaces`	Linux is designed to work mainly with the shell (or terminal), which gives the user greater control over the operating system.
`Configuration data stored in a text file`	An example of such a file is the /etc/passwd file, which stores all users registered on the system.

Components:

Component	Description
`Bootloader`	A piece of code that runs to guide the booting process to start the operating system. Parrot Linux uses the GRUB Bootloader.
`OS Kernel`	The kernel is the main component of an operating system. It manages the resources for I/O devices the system at the hardware level.
`Daemons`	Background services are called "daemons" in Linux. Their purpose is to ensure that key functions such as scheduling, printing, and multimedia are working correctly. These small programs load after we booted or log into the computer.
`OS Shell`	The operating system shell or the command language interpreter (also known as the command line) is the interface between the OS and the user. This interface allows the user to tell the OS what to do. The most commonly used shells are Bash, Tcsh/Csh, Ksh, Zsh, and Fish.

Component	Description
`Graphics server`	This provides a graphical sub-system (server) called "X" or "X-server" that allows graphical programs to run locally or remotely on the X-windowing system.
`Window Manager`	Also known as a graphical user interface (GUI). Many options include GNOME, KDE, MATE, Unity, and Cinnamon. A desktop environment usually has several applications, including file and web browsers. These allow the user to access and manage an operating system's essential and frequently accessed features and services.
`Utilities`	Applications or utilities are programs that perform particular functions for the user or another program.

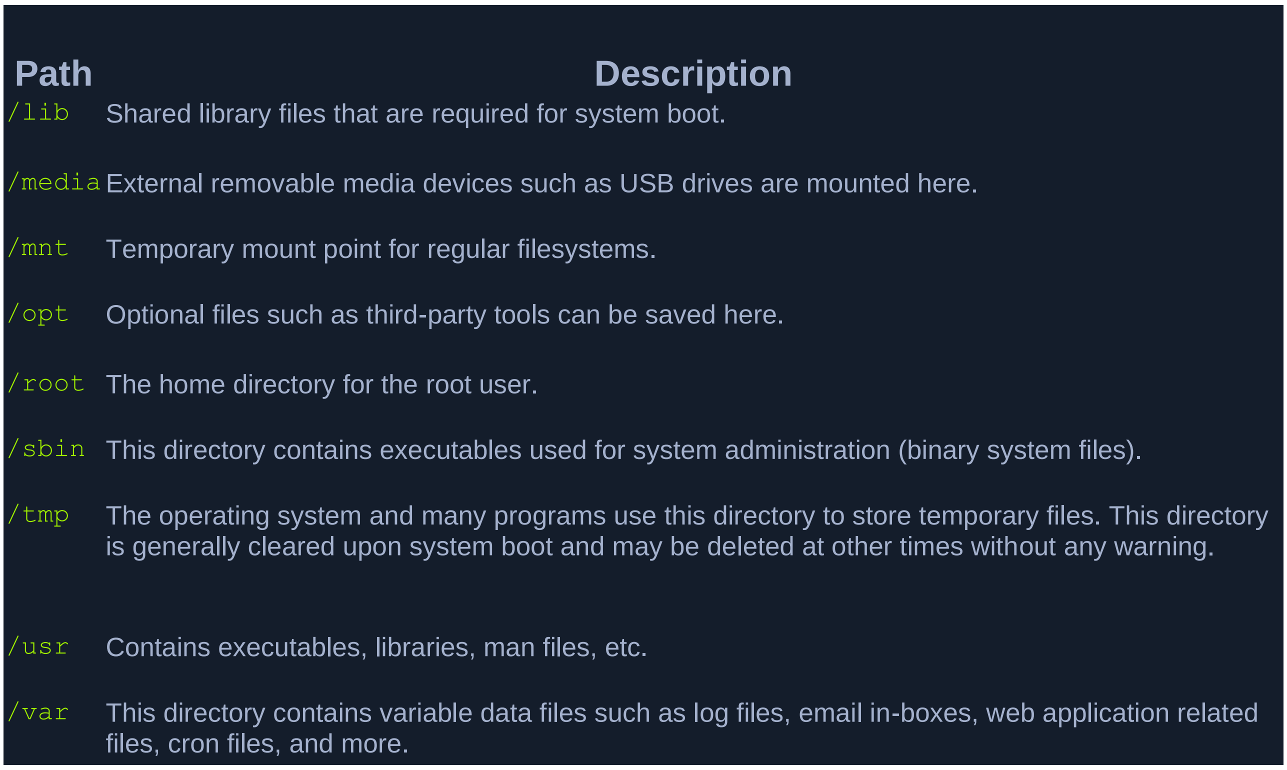
Linux Architecture:

Linux OS can be broken down into layers:

Layer	Description
`Hardware`	Peripheral devices such as the system's RAM, hard drive, CPU, and others.
`Kernel`	The core of the Linux operating system whose function is to virtualize and control common computer hardware resources like CPU, allocated memory, accessed data, and others. The kernel gives each process its virtual resources and prevents/mitigates conflicts between different processes.
`Shell`	A command-line interface (**CLI**), is also known as a shell that a user can enter commands into to execute the kernel's functions.
`System Utility`	Makes available to the user all of the operating system's functionality.

File System Hierarchy:

Path	Description
`/`	The top-level directory is the root filesystem and contains all of the files required to boot the operating system before other filesystems are mounted as well as the files required to boot the other filesystems. After boot, all of the other filesystems are mounted at standard mount points as subdirectories of the root.
`/bin`	Contains essential command binaries.
`/boot`	Consists of the static bootloader, kernel executable, and files required to boot the Linux OS.
`/dev`	Contains device files to facilitate access to every hardware device attached to the system.
`/etc`	Local system configuration files. Configuration files for installed applications may be saved here as well.
`/home`	Each user on the system has a subdirectory here for storage.



Path	Description
`/lib`	Shared library files that are required for system boot.
`/media`	External removable media devices such as USB drives are mounted here.
`/mnt`	Temporary mount point for regular filesystems.
`/opt`	Optional files such as third-party tools can be saved here.
`/root`	The home directory for the root user.
`/sbin`	This directory contains executables used for system administration (binary system files).
`/tmp`	The operating system and many programs use this directory to store temporary files. This directory is generally cleared upon system boot and may be deleted at other times without any warning.
`/usr`	Contains executables, libraries, man files, etc.
`/var`	This directory contains variable data files such as log files, email in-boxes, web application related files, cron files, and more.

Shell:-

Shell: A Linux terminal.

Most commonly used shell: BASH (Bourne-Again Shell) – part of GNU project

Other shells: Tcsh/Csh, Ksh, Zsh, Fish.

Terminal Emulators: use text-based programs with a graphical user interface.

Famous terminal emulators… GNOME, Terminal, Xterm, XFCE4 terminal

Terminal Multiplexers: Additional terminals in one terminal. Tmux, GNU screen.

Manual pages and Help:

Find detailed manual with detailed explanation for any particular tool using, **$man <tool>**

Look at the optional parameters of particular tool without browsing full documentation using, **$<tool> --help** or **$<tool> -h**

Another way to search for short descriptions from the available manual page, “Apropos”. Apropos search the description for instances of a given keyword. Using, **$apropos <keyword>** …… For example, $apropos sudo

I'm not able to understand long commands >>> use https://explainshell.com/

System information:

It covers information about the system, its processes, network configuration, users, directories, user settings and corresponding params.

Command	Description
`whoami`	Displays current username.
`id`	Returns users identity
`hostname`	Sets or prints the name of the current host system.
`uname`	Prints basic information about the operating system name and system hardware.
`PWD`	Returns working directory name.
`ifconfig`	The ifconfig utility is used to assign or to view an address to a network interface and/or configure network interface parameters.
`IP`	Ip is a utility to show or manipulate routing, network devices, interfaces and tunnels.
`netstat`	It Shows network status.
`ss`	Another utility to investigate sockets.
`ps`	Shows process status.
`who`	Displays who is logged in.
`env`	Prints environment or sets and executes the command.
`lsblk`	Lists block devices.
`lsusb`	Lists USB devices
`lsof`	Lists opened files.
`lspci`	Lists PCI devices.

System management:-

User Management:

An essential part of Linux administration.

- ➔ create new users, add other users to specific groups.
- ➔ execute commands as a different user.
- ➔ Cheatsheet for user management

Command	Description
`sudo`	Execute command as a different user.

Command	Description
`su`	The `su` utility requests appropriate user credentials via PAM and switches to that user ID (the default user is the superuser). A shell is then executed.
`useradd`	Creates a new user or update default new user information.
`userdel`	Deletes a user account and related files.
`usermod`	Modifies a user account.
`addgroup`	Adds a group to the system.
`delgroup`	Removes a group from the system.
`passwd`	Changes user password.

Package Management:

Packages are archives that contain binaries of software, config files, info about dependencies and track of updates and upgrades.

Features that most package management systems provide:

- ➔ Package downloading
- ➔ Dependency resolution
- ➔ A standard binary package format
- ➔ Common installation and config locations
- ➔ Additional system related configuration and functionality
- ➔ Quality control

examples: .deb, .rpm, etc etc…

Various package management programs:

Command	Description
`dpkg`	The `dpkg` is a tool to install, build, remove, and manage Debian packages. The primary and more user-friendly front-end for `dpkg` is aptitude.
`apt`	Apt provides a high-level command-line interface for the package management system.
`aptitude`	Aptitude is an alternative to apt and is a high-level interface to the package manager.
`snap`	Install, configure, refresh, and remove snap packages. Snaps enable the secure distribution of the latest apps and utilities for the cloud, servers, desktops, and the internet of things.
`gem`	Gem is the front-end to RubyGems, the standard package manager for Ruby.
`pip`	Pip is a Python package installer recommended for installing Python packages that are not available in the Debian archive. It can work with version control repositories (currently only Git, Mercurial, and Bazaar repositories), logs output extensively, and prevents partial installs by downloading all requirements before starting installation.
`git`	Git is a fast, scalable, distributed revision control system with an unusually rich command set that provides both high-level operations and full access to internals.

APT (Advanced Package Manager):

Debian based Linux distributions use apt. It is an archive file containing multiple ".deb" files. Many software has a lot more dependencies and requirements, If we gonna install single single ".deb" files of software then it contains issues in dependencies. So, apt make it easier & more efficient by packaging together all of the dependencies needed to install the program.

Repositories are labelled as stable, testing or unstable. We can check by viewing the content of /etc/apt/sources.list file.

apt uses a database called apt-cache. Apt cache provides info about packages installed on our system offline.

For example, Command to find all impacket related packages

```
$ apt-cache search impacket
```

Additional info about packages:

```
$ apt-cache show impacket-scripts
```

List all installed packages:

```
$ apt list --installed
```

Install missing packages:

```
$ sudo apt install impacket-scripts -y
```

Git:

Used to download useful tools from GitHub.

Command: git clone <url>

DPKG:

Download programs and tools separately.

Command:
```
$ sudo dpkg -i strace_4.21-1ubuntu1_amd64.deb
```

Service and Process management:

Service management:

two types of services: internal & system services.

services that are required at system startup are called system services. which perform hardware related tasks, and services that are installed by the user… include all server services.

These services are running in the background without the user's interaction. also called **"daemons"**. For ex. sshd, system etc…

Daemons is an init process started first and thus has the PID (Process ID) = 1. Daemon's work is to monitor and take care of orderly started services. All processes have PID, which we can see under /proc/ with PID number. Some processes have PPID (Parent Process ID) ... these are the child processes.

Used tools to manage System services: systemctl

To manage SysV init script links: update-rc.d

Systemctl usage:

To start system service:

```
$ systemctl start <service_name>
```

To check the status of service:

```
$ systemctl status <service_name>
```

To add the service to SysV script to tell the system that "run service after startup" OR (in short) enable the service after every startup:

```
$ systemctl enable <service_name>
```

To list all services:

```
$ systemctl list-units --type=service
```

To view the logs of the service (If an error occurred and the service isn't able to start): use command journalctl

```
$ journalctl -u ssh.service --no-pager
```

< it shows the logs of the ssh service >

Process Management:

To kill a process:

4 states of the process-

Running

Waiting

Stopped

Zombie (stopped but still has an entry in process table)

Control the processes using, **kill**, **pkill**, **pgrep**, **killall**.

Umm… How to interact with process?? >>> Send a signal

Which type of signal? >>> `$ kill -l  //It will list all signals.`

Most commonly used signals…

Signal	Description
1	`SIGHUP` - This is sent to a process when the terminal that controls it is closed.
2	`SIGINT` - Sent when a user presses `[Ctrl] + C` in the controlling terminal to interrupt a process.
3	`SIGQUIT` - Sent when a user presses `[Ctrl] + D` to quit.
9	`SIGKILL` - Immediately kill a process with no clean-up operations.
15	`SIGTERM` - Program termination.
19	`SIGSTOP` - Stop the program. It cannot be handled anymore.
20	`SIGTSTP` - Sent when a user presses `[Ctrl] + Z` to request for a service to suspend. The user can handle it afterwards.

For example, A program was to freeze we could force to kill it. Then,

```
$ kill 9 <PID>
```

Background a process:

Background processes don't require user interaction. So, we can use the same shell without waiting until the process finish. Once the process finishes, we'll get notified by a terminal.

Press [ctrl] + z to suspend & then type "bg". Now your process is running in the background.

How to display all background processes? >>> $jobs

Automatically set the process as a background process: & sign at the end of the command.

For example: `$ ping -c 10 <url> &`

Foreground a process:

If we want to get the background process into the foreground and want to interact with the process then, use the $fg <id> command.

For example, `$ fg 1`

Execute multiple commands in a single line:

use

- ➔ ; (semicolon) OR && (double ampersand) OR | (pipe)
- ➔ `$ command1; command2; command3`
- ➔ In the above scenario, command1's output doesn't matter for command2. If command1 gives an error, still command2 & 3 are going to execute.
- ➔ `$ command1 && command2 && command3`
- ➔ In the above scenario, every command's output is matters for the next command. If command1 gives an error, command 2 & 3 will not execute.
- ➔ `$ command1 | command2 | command3`
- ➔ Execute command1, the output will be redirected to command2, and their output will be redirected to command3. Pipe not only depend on the correct or error-free operation of previous processes.

Web services:

using Apache webserver. Install Apache using apt…

Widely used command when we talk about web services: curl

cURL is the tool that allows us to transfer files from the shell over HTTP, HTTPS, FTP, SFTP, FTPS, SCP.

```
$ curl http://localhost
```

Second most used command: wget (alternative to curl)

Download a webpage file remotely on your system over HTTP or FTP.

```
$ wget http://localhost
```

When comes to the data transfer, another tool is: python3

When you execute a command, it will start the server in that particular directory.

```
$ python3 -m HTTP.server <port>
```

(default port 8000)

ps: you can do it not from just python3, use PHP, npm or other services also.

Workflow:-

Commands: pwd, cd, ls,

Jump in a directory where we last in… cd -

get index number of listing directories files using… ls -i

List hidden dirs. and files using… ls -la

Working with files and dirs:

create, move, copy, delete:

```
create empty file: $ touch <filename>
create empty directory: $ mkdir <dirname> (-p to add parent dir/)
Look whole structure in particular dir: $ tree .
Move file/dir:  $ mv <file/directory> <renamed file/directory>
Copy file: $ cp <filename> <path-to-dir>/<filename>
List : $ ls (check flags for filters using help or man)
```

Editing files:

Text editors like vim or vi, nano, gedit, mousepad.

Vim:

Improved clone of vi. A powerful editor follows the Unix principle. It provides an interface to external programs like grep, sed, awk etc…

Vim can distinguish between text and command input.

Modes of vim:

Mode	Description
`Normal`	In normal mode, all inputs are considered as editor commands. So there is no insertion of the entered characters into the editor buffer, as is the case with most other editors. After starting the editor, we are usually in the normal mode.
`Insert`	With a few exceptions, all entered characters are inserted into the buffer.
`Visual`	The visual mode is used to mark a contiguous part of the text, which will be visually highlighted. By positioning the cursor, we change the selected area. The highlighted area can then be edited in various ways, such as deleting, copying, or replacing it.
`Command`	It allows us to enter single-line commands at the bottom of the editor. This can be used for sorting, replacing text sections, or deleting them, for example.
`Replace`	In replace mode, the newly entered text will overwrite existing text characters unless there are no more old characters at the current cursor position. Then the newly entered text will be added.

Don't familiar with vim? >>> use vimtutor, practice on it and then use vim.

Find files & directories:

(focus more on this, your weakness ☹)

Tools: which, find, locate

- Which:
  - ➢ Returns path to the file or directory that should be executed.
  - ➢ Allow us to determine specific programs like curl, nc, wget, python...

Command:
```
$ which python
```

- Find:
  - ➢ Use to find files or folders using filters like date, size etc…

syntax:
```
$ find <location> <options>
```

```
eg: $ find / -type f -name *.conf -user root -size +20k -newermt
2020-03-03 -exec ls -al {} \; 2>/dev/null
```

Explanation: find from / , -type ‘f’ = file & ‘d’ = directory, -name filename (* = wildcard & .ext = extension of file), -user username, -size Greater_than_20k, -newermt files_which_are_newer_than_$date , -exec execute_command

Option	Description
`-type f`	Hereby, we define the type of the searched object. In this case, '`f`' stands for '`file`'.
`-name *.conf`	With '`-name`', we indicate the name of the file we are looking for. The asterisk (`*`) stands for 'all' files with the '`.conf`' extension.
`-user root`	This option filters all files whose owner is the root user.
`-size +20k`	We can then filter all the located files and specify that we only want to see the files that are larger than 20 KiB.
`-newermt 2020-03-03`	With this option, we set the date. Only files newer than the specified date will be presented.
`-exec ls -al {} \;`	This option executes the specified command, using the curly brackets as placeholders for each result. The backslash escapes the next character from being interpreted by the shell because otherwise, the semicolon would terminate the command and not reach the redirection.
`2>/dev/null`	This is a `STDERR` redirection to the '`null device`', which we will come back to in the next section. This redirection ensures that no errors are displayed in the terminal. This redirection must `not` be an option of the 'find' command.

- Locate:
  - ➢ It will take less time than find command. But don’t have many filters like find command.
  - ➢ It works with a local database and contains all info about files & dirs.
  - ➢ update database using $sudo updated nd then run command

```
command: $ locate *.conf
```

File Descriptors and Redirectors:

Descriptors:

Indicator of connection maintained by kernel to perform I/O operations.

In windows, it’s called filehandle.

It’s a connection from OS to perform I/O operations (Input/Output of bytes).

First three file descriptors:

Datastream for input: STDIN – 0

Datastream for output: STDOUT – 1

Datastream for output error: STDERR – 2

Example: redirect to standard output and standard error to 2 different files.

```
$ find /etc/ -name shadow 2> stderr.txt 1> stdout.txt
```

Redirect STDIN: it will redirect with < sign.

```
$ cat < stdout.txt - (redirect stdin & display content)
$ cat > stdout.txt - (if file no exist-create, exist-overwrite)
$ cat >> stdout.txt – (append STDOUT in existing file)
$ cat << EOF > stream.txt (Redirect STDIN stream to File –
EOF tells End-Of-the-File)
```

Pipes: Another way to redirect STDOUT files.

More information about pipes is discussed above.

Filter contents:

Fundamental pagers: more & less to read the content of the file.

More: Content read using cat and redirect it to more. Output remains in the terminal.

Command: `$ more file_name`

Less: Output doesn’t remain in the terminal. Have more functions than more, look at `$man less`. Presentation is almost the same as more.

Read-only beginning or end content of the file: head & tail.

Head: Read the first 10 lines of a file if not specified or otherwise.

Tail: Read the last part of the file or result.

Sort desired results alphabetically or numerically: sort

```
Command: $ cat file_name | sort
```

Find exact match strings from your files: grep

```
command: $ cat /etc/passwd | grep "/bin/bash"
```

use -v with grep to exclude specific results.

```
ex. $ cat /etc/passwd | grep -v "false\|nologin"
```

For specific results separated with delimiters: cut

```
command: $ cat /etc/passwd | grep -v "false\|nologin" | cut -d":" -f1
```

Replace certain characters with another defined char: tr

```
Command: $ cat /etc/passwd | grep -v "false\|nologin" | tr ":" " "
```

Display results in tubular form: column

```
command: $ cat /etc/passwd | grep -v "false\|nologin" | tr ":" " " |
column -t
```

To display first ($1) and last ($NF) result of the line: awk

```
command: $ cat /etc/passwd | grep -v "false\|nologin" | tr ":" " " |
awk '{print $1, $NF}'
```

Stream editor to change specific word/pattern in whole file or stdin: sed

Sed looks for the pattern we have defined in the form of regular expressions and replaces them with another pattern.

```
command: $ cat /etc/passwd | grep -v "false\|nologin" | tr ":" " " |
awk '{print $1, $NF}' | sed 's/bin/aaa/g'
```

s: substitute command

bin: pattern we want to replace

aaa: pattern we want to use as a replacement

g: stands for replacing all matches.

For count words or lines in particular file: wc

```
command: $ cat /etc/passwd | grep -v "false\|nologin" | tr ":" " " |
awk '{print $1, $NF}' | wc -l (-l count lines)
```

$ netstat -tunleep4 |grep -v "127.0.0" | grep "LISTEN" | wc -l

$ ps aux | grep "proftpd"

$curl https://www.inlanefreight.com/ | grep -Po 'https://www.inlanefreight.com/\K[^"\x27]+' | sort -u | wc -l ➔ 33

curl https://www.inlanefreight.com --insecure > ilf

cat ilf | grep "https://www.inlanefreight.com" > ilf.1

cat ilf.1 | tr " " "\n" | sort | grep "inlanefreight.com" | cut -d'"' -f2 | sort | cut -d"'" -f2 | sort | uniq -c > ilf.2

cat ilf.2 | wc -l

$> 34

Permission Management:

Permissions: Read, Write, Execute

Assigned to: Owner, Groups, Others

The whole permission system is based on the octal number system.

```
Overview: $ ls -l /etc/passwd

- rwx rw- r--   1 root root 1641 May  4 23:42 /etc/passwd
- --- --- ---   |  |    |    |   |__________|
|  |   |   |    |  |    |    |        |_ Date
|  |   |   |    |  |    |    |__________ File Size
|  |   |   |    |  |    |_______________ Group
|  |   |   |    |  |____________________ User
|  |   |   |    |_______________________ Number of hard links
|  |   |   |_ Permission of others (read)
|  |   |_____ Permissions of the group (read, write)
|  |_________ Permissions of the owner (read, write, execute)
|____________ File type (- = File, d = Directory, l = Link, ... )
```

Change Permissions:

the command used: chmod

u- user, g- group, o- other, a- all

add or remove permissions: using + or –

example: apply read permission for all users.

```
$ chmod a+r file/dir_name
$ chmod 754 file/dir_name   (same as above)
```

```
Binary Notation:                4 2 1  |  4 2 1  |  4 2 1
----------------------------------------------------------
Binary Representation:          1 1 1  |  1 0 1  |  1 0 0
----------------------------------------------------------
Octal Value:                      7    |    5    |    4
----------------------------------------------------------
Permission Representation:      r w x  |  r - x  |  r - -
```

Change Owner:

command using: chown

```
syntax: $ chown <user>:<group> <file/directory>
```

SUID & GUID:

Configure special permissions for files and directories using: SUID (Set User ID) & GUID (Set Group ID)

SUID/GUID bits allow users to run the program with the rights of another user.

Admin also can give their users some special rights for certain apps or files.

In this case, the letter ‘s’ is used instead of ‘x’.

When we execute these programs, at that time file owner’s SUID/GUID is being used.

Notes: Don’t do these things as an admin, if you are not familiar.

Tips & Tricks:-

Shortcuts:

Autocomplete: tab

Cursor movement:

ctrl + A: move at the beginning of a current line

ctrl + E: move at the end of the current line

ctrl + ←/→ : jump at the beginning of current/previous words.

alt + B/F: jump backward/forward one word.

Erase the current line:

ctrl + U: erase everything from the current position to the beginning.

ctrl + K: erase everything from the current position to the end.

ctrl + W: erase the word preceding the cursor position.

Paste erased content: ctrl + Y

End task: ctrl + C

End-of-File (EOF): ctrl + D (close STDIN pipe aka EOF)

Clear terminal: ctrl + L (shortcut of `$clear` command.)

Background a process: ctrl + Z

search through command history: ctrl + R

Type previous/next command: ⬆/⬇

Switch between apps: alt + tab

Zoom in: ctrl + [+]

Zoom out: ctrl + [-]

Linux Security:

One of the most security measures of Linux OS: Keep the OS up to date

Command:
```
$ apt update && apt dist-upgrade
```

If firewall rules are not set at the network level, we can use a Linux firewall or iptables to restrict the traffic.

If SSH opens on the server, disallow password login & disallow root user logging via SSH.

Another common protection method: fail2ban (count number of failed login attempts)

Issues that lead to privileged escalation:

Out of date kernel, user permission issues, world-writable files, misconfigured cron jobs or misconfigured services.

Kernel security modules: SELinux (Security-Enhanced Linux), AppArmor

→ used for security access control policies.

Some other Linux security Tools:

Snort, chkrootkit, rkhunter, Lynis

In addition, some security settings should be made… such as:

- Removing or disabling all unnecessary services and software
- Removing all services that rely on unencrypted authentication mechanisms
- Ensure NTP is enabled and Syslog is running
- Ensure that each user has their account
- Enforce the use of strong passwords
- Set up password aging and restrict the use of previous passwords
- Locking user accounts after login failures
- Disable all unwanted SUID/SGID binaries

Web requests

Introduction:

HTTP (HyperText Transfer Protocol):

Application-level protocol used to access resources over WWW.
Hypertext: Text containing links to other resources.
Communication consists of client & server, where client request to the server for resource && server process the request and provide the resource.
Default port: 80

Fully Qualified Domain Name (FQDN): www.example.com

Resources over HTTP are accessed via URL.

Structure of URL:
http://username:password@website.com:80/dashboard.php?login=true#status

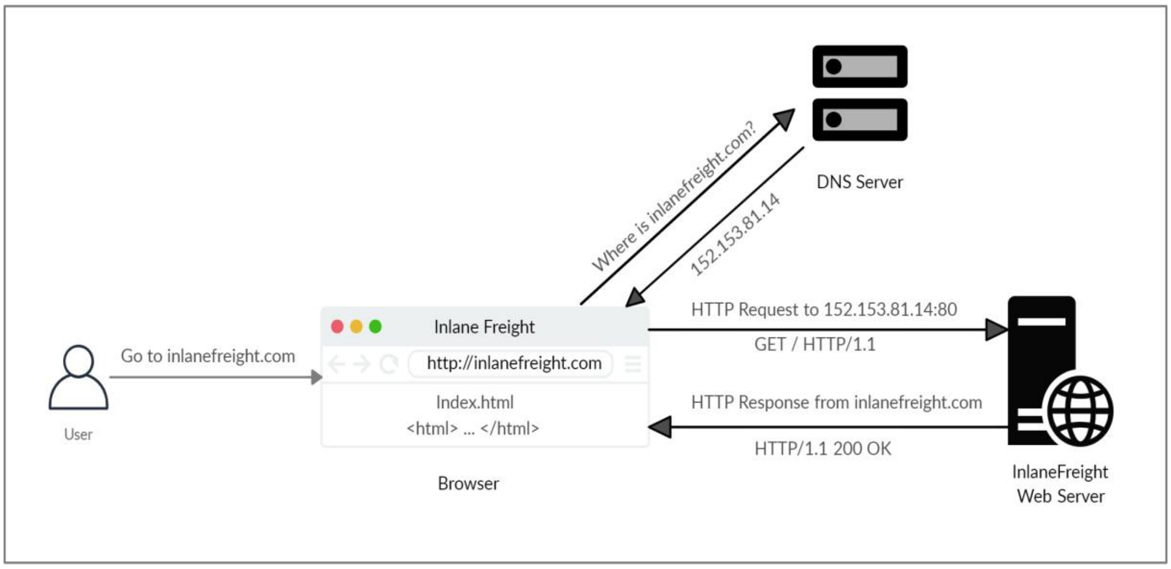
- ➔ scheme :// user_info @ host : port / path ? query_string # fragment

Let's understand it in detail:

Component	Description
Scheme	This is used to identify the protocol being accessed by the client. This is usually `HTTP` or `HTTPS`.
User Info	This is an optional component that contains credentials in the form of `username:password`, which is used to authenticate to the host.
Host	The host signifies the resource location. This can be a hostname or an IP address. A colon separates a host and port.
Port	URLs without a port specified point to the default port 80. If the HTTP server port isn't running on port 80, it can be specified in the URL.
Path	This points to the resource being accessed, which can be a file or a folder. If there is no path specified, the server returns the default index document hosted by it (for example, index.html).
Query String	The query string is preceded by a question mark (?). This is another optional component that is used to pass information to the resource. A query string consists of a parameter and a value. In the example above, the parameter is `login`, and its `value` is true. There can be multiple parameters separated by an ampersand (&).
Fragments	This is processed by browsers on the client-side to locate sections within the primary resource.

Note: Not all components are always required to access resources.

HTTP flow:



HTTPS (HyperText Transfer Protocol secure):

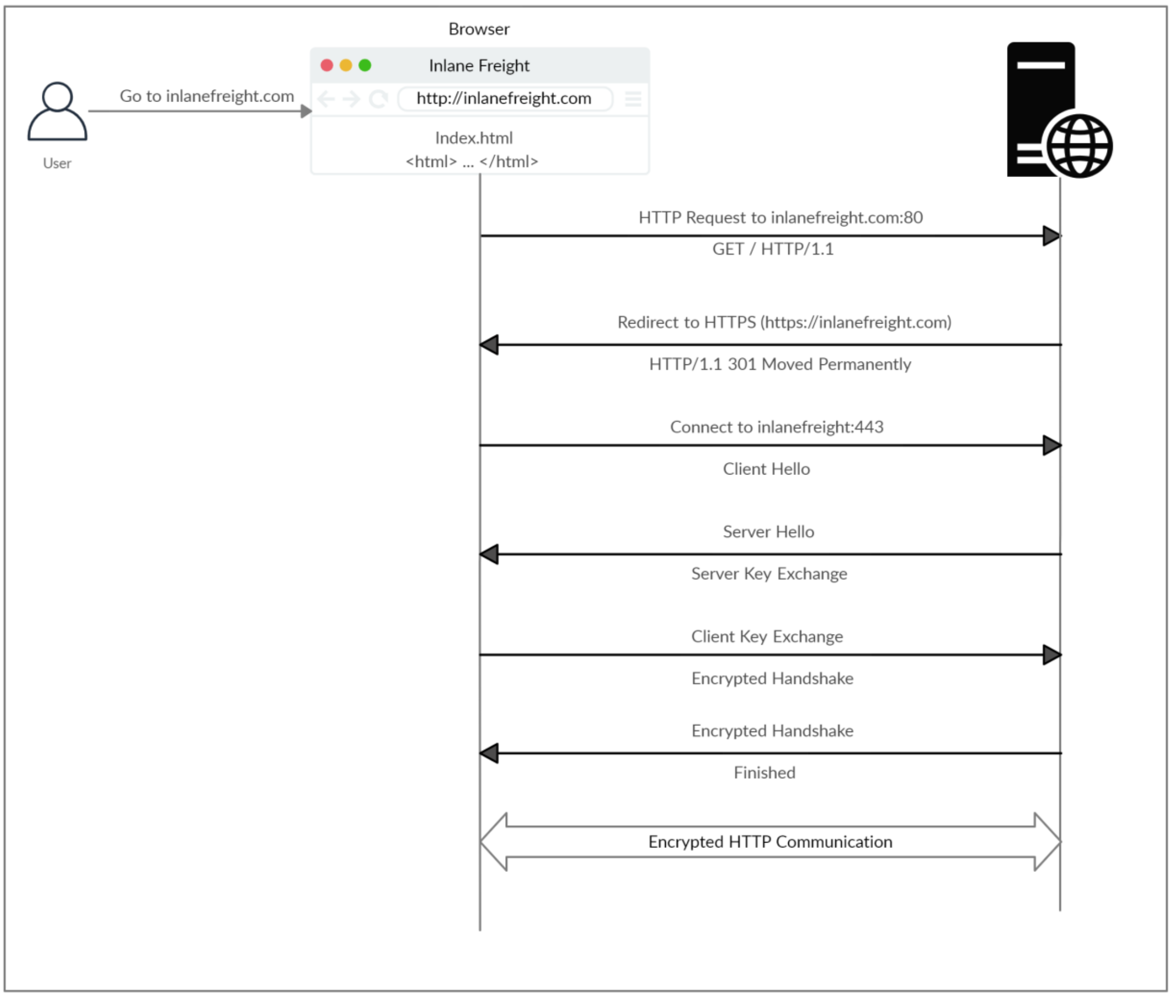
In HTTP all data has been sent in clear text format But in HTTPS All data is in an encrypted format. Encrypted by **TLS/SSL** protocol.

It uses symmetric and asymmetric **both types of encryption**.

Asymmetric encryption to *establish a connection* & **symmetric encryption** to *transfer data* (which we call session).

Default port: 443

HTTPs flow:



HTTP downgrade attack: Attacker may be able to downgrade HTTPS communication to HTTP. This is done by MITM & proxying.

Requests & Response:

Burpsuite: Tool that acts like a proxy server. Used to examine and modify HTTP requests.

HTTP request:

HTTP request in burp: contains 3 fields.

Field	Description
Method	The first field stands for the HTTP method or verb, which specifies the type of action to perform.
Path	The second field is the path to the resource being accessed. This field can also be suffixed with a query string.
Version	The third and final field is used to denote the HTTP version.

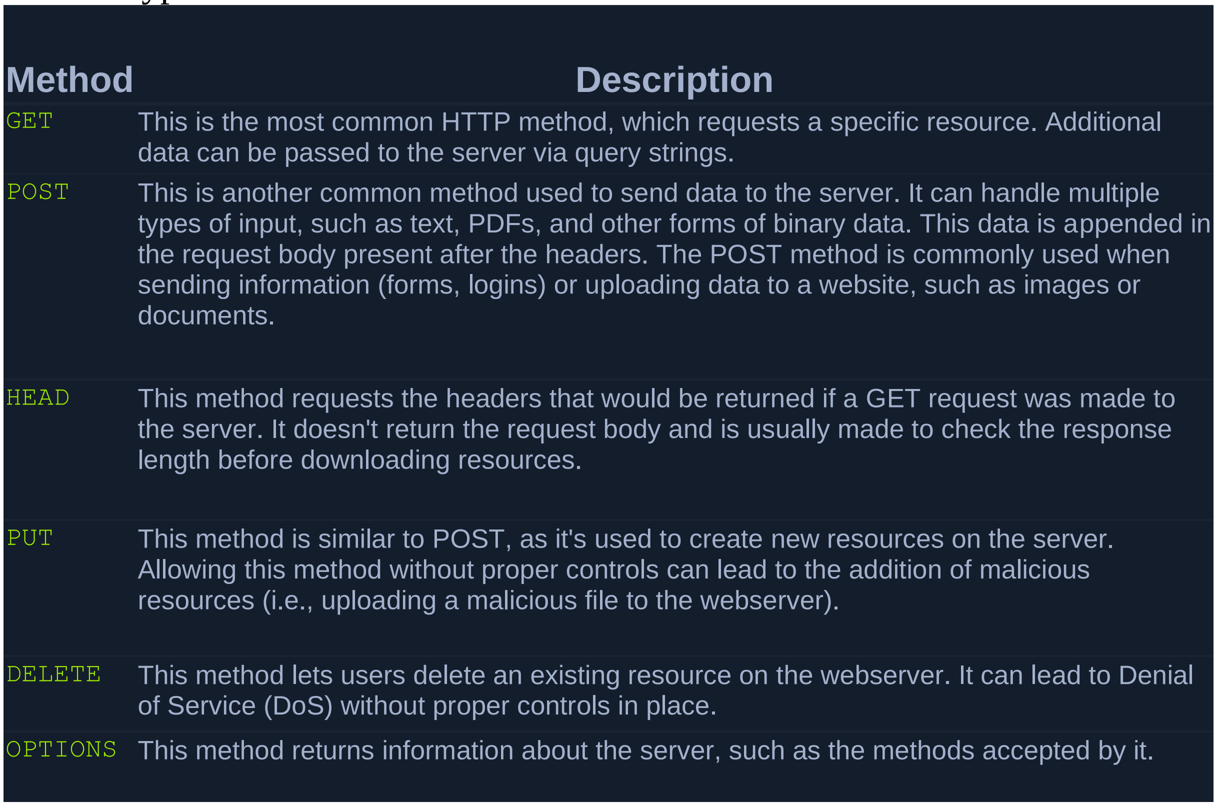
The next of the lines contain HTTP header value pairs. Headers are always terminated with a new line, which is necessary for the server to validate the requests.

HTTP response:
Similar to requests, the response also contains headers.
The First-line contains 2 fields.
HTTP version & Response code.
Response code: used to determine if the request gets succeeded or not.

HTTP Methods and Codes:

Method: Tell the server how to process the request and how to reply.

Various types of methods:



Method	Description
`GET`	This is the most common HTTP method, which requests a specific resource. Additional data can be passed to the server via query strings.
`POST`	This is another common method used to send data to the server. It can handle multiple types of input, such as text, PDFs, and other forms of binary data. This data is appended in the request body present after the headers. The POST method is commonly used when sending information (forms, logins) or uploading data to a website, such as images or documents.
`HEAD`	This method requests the headers that would be returned if a GET request was made to the server. It doesn't return the request body and is usually made to check the response length before downloading resources.
`PUT`	This method is similar to POST, as it's used to create new resources on the server. Allowing this method without proper controls can lead to the addition of malicious resources (i.e., uploading a malicious file to the webserver).
`DELETE`	This method lets users delete an existing resource on the webserver. It can lead to Denial of Service (DoS) without proper controls in place.
`OPTIONS`	This method returns information about the server, such as the methods accepted by it.

Note: Both "PUT" and "DELETE" are usually associated with WebDAV servers.
The availability of a particular method depends on server and app configuration.

Response codes: The server use response codes to tell the client whether the request successfully proceeds or not.

Five types of response codes:

Type	Description
`1xx`	Usually provides information and continues processing the request.
`2xx`	Positive response codes are returned when a request succeeds.
`3xx`	Returned when the server redirects the client.
`4xx`	This class of codes signifies improper requests from the client. For example, requesting a resource that doesn't exist or requesting a bad format.
`5xx`	Returned when there is some problem with the HTTP server itself.

Common HTTP response codes:

Type	Description
`200 OK`	Returned on a successful request, and the response usually contains the requested resource.
`302 Found`	This code redirects the client to another URL. For example, redirecting the user to their dashboard after a successful login.
`400 Bad Request`	Usually returned on encountering malformed requests such as requests with missing line terminators.
`403 Forbidden`	This code signifies that the client doesn't have appropriate access to the resource. It can also be returned when the server detects malicious input from the user.
`404 Not Found`	Returned when the client requests a resource that doesn't exist on the server.
`500 Internal Server Error`	As the name describes, this code is returned when the server cannot process the request.

HTTP Headers:

Provide an additional way to pass information between client and server.
Headers can have one or multiple values separated by a colon.
Different categories of Headers:

1) General headers
2) Entity headers
3) Request headers
4) Response headers
5) Security headers

1. General headers

Don’t belong specifically to request or response.

Header	Description
`Date`	The `Date` header holds the date and time at which the message originated. It's preferred to convert the time to the standard UTC zone.
`Connection`	The `Connection` header dictates if the current network connection should stay alive after the request finishes. Two commonly used values for this header are `close` and `keep-alive`. The `close` value from either the client or server means that they would like to terminate the connection, while the `keep-alive` header indicates that the connection should remain open.

2. Entity headers

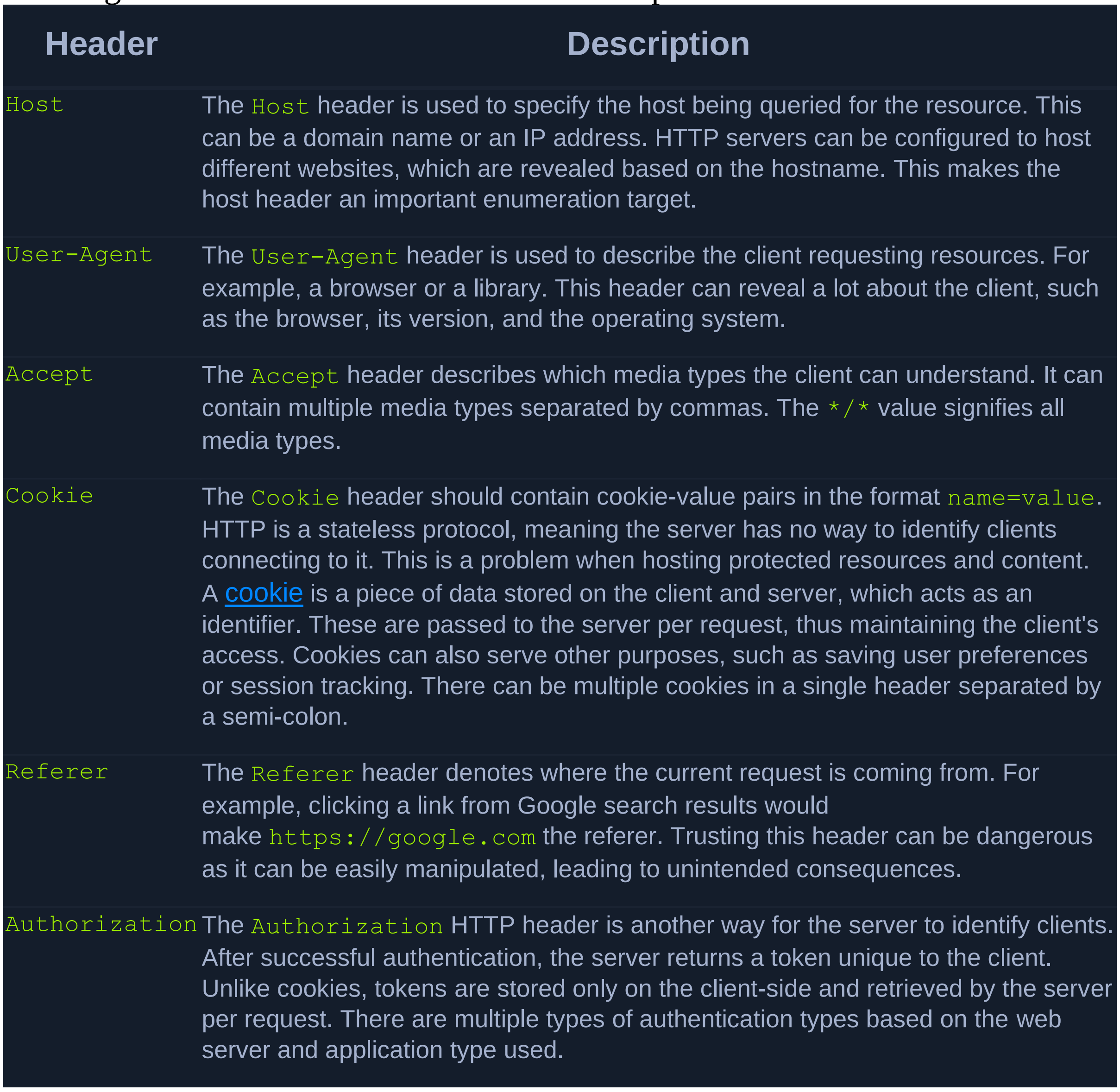
Similar to general headers. Can be common to both request and response.
Used to describe content being transferred by message.
Usually found in POST and PUT requests and responses.

Header	Description
`Content-Type`	This header is used to describe the type of resource being transferred. The value is automatically added by the browsers on the client-side and returned in the server response.
`Media-Type`	The `media-type` describes the data being passed. For example, the media-type for a PDF is `application/pdf`, while the type for a PNG image is `image/png`. This header can play a crucial role in making the server interpret our input. The `charset` field denotes the encoding standard, such as UTF-8.
`Boundary`	The `boundary` directive acts as a maker to separate content when there is more than one in the same message.
`Content-Length`	The `Content-Length` header holds the size of the entity being passed. This header is necessary as the server uses it to read data from the message body.
`Content-Encoding`	Data can undergo multiple transformations before being passed. For example, large amounts of data can be compressed to reduce the message size. The type of encoding being used should be specified using the `Content-Encoding` header.

3. Request headers

The client sends request headers in an HTTP transaction. Defined in RFC 2616.

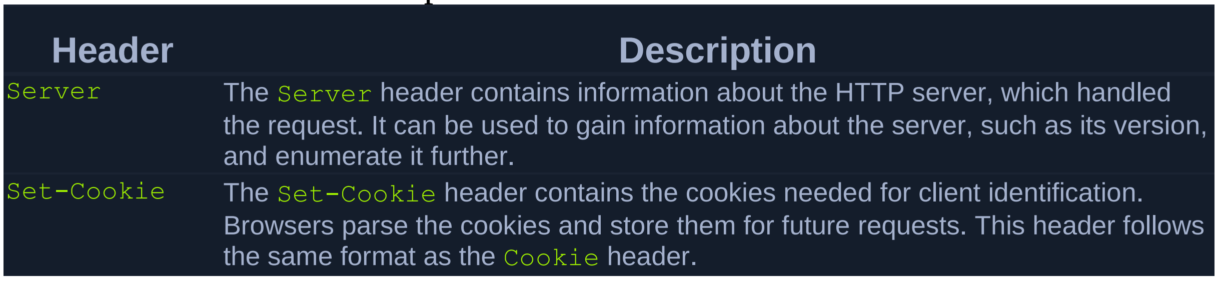
Do not relate to the content of a message.
ex. headers: Accept, Accept-*, IF-* allow for conditional requests. and cookie, User-agent is sent that server can tailor the response.



Header	Description
`Host`	The `Host` header is used to specify the host being queried for the resource. This can be a domain name or an IP address. HTTP servers can be configured to host different websites, which are revealed based on the hostname. This makes the host header an important enumeration target.
`User-Agent`	The `User-Agent` header is used to describe the client requesting resources. For example, a browser or a library. This header can reveal a lot about the client, such as the browser, its version, and the operating system.
`Accept`	The `Accept` header describes which media types the client can understand. It can contain multiple media types separated by commas. The `*/*` value signifies all media types.
`Cookie`	The `Cookie` header should contain cookie-value pairs in the format `name=value`. HTTP is a stateless protocol, meaning the server has no way to identify clients connecting to it. This is a problem when hosting protected resources and content. A cookie is a piece of data stored on the client and server, which acts as an identifier. These are passed to the server per request, thus maintaining the client's access. Cookies can also serve other purposes, such as saving user preferences or session tracking. There can be multiple cookies in a single header separated by a semi-colon.
`Referer`	The `Referer` header denotes where the current request is coming from. For example, clicking a link from Google search results would make `https://google.com` the referer. Trusting this header can be dangerous as it can be easily manipulated, leading to unintended consequences.
`Authorization`	The `Authorization` HTTP header is another way for the server to identify clients. After successful authentication, the server returns a token unique to the client. Unlike cookies, tokens are stored only on the client-side and retrieved by the server per request. There are multiple types of authentication types based on the web server and application type used.

4. Response headers

Used to provide more context about the response.
Detailed information on response headers – RFC 7231.



Header	Description
`Server`	The `Server` header contains information about the HTTP server, which handled the request. It can be used to gain information about the server, such as its version, and enumerate it further.
`Set-Cookie`	The `Set-Cookie` header contains the cookies needed for client identification. Browsers parse the cookies and store them for future requests. This header follows the same format as the `Cookie` header.

Header	Description
`WWW-Authenticate`	The `WWW-Authenticate` header notifies the client about the type of authentication required to access the requested resource.

5. Security headers
HTTP security headers are a class of response headers used to specify certain rules and policies to be followed by the browser while accessing a website.
Detailed information: https://owasp.org/www-project-secure-headers/

Header	Description
`Content-Security-Policy`	The CSP header dictates the website's policy towards externally injected resources. This could be JavaScript code as well as script resources. This header instructs the browser to accept resources only from certain trusted domains, hence preventing attacks such as Cross-site scripting.
`Strict-Transport-Security`	The HTTP Strict Transport Security policy of a website prevents the browser from accessing the website over the plaintext HTTP protocol. All communication is done via the secure HTTPS protocol. This prevents attackers from sniffing web traffic and accessing protected information such as passwords or other sensitive data.
`Referrer-Policy`	This header dictates whether the browser should include the value specified via the `Referrer` header or not. It can help in avoiding disclosing sensitive URLs and information while browsing the website.

Notes: Application can define custom headers based on their requirements. Above discussed all are common headers.
A complete list of standard HTTP headers: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Dive into methods:-

GET Method:

- ➔ most commonly used method.
- ➔ purpose: Request to given resource and retrieve it.
- ➔ Send data: in query parameters (this data should be limited & for long data – we can use POST)
- ➔ Get can be used for basic authentication. Like username:password with the authorization header in the request.
- ➔ For example- authorization: Basic base64encodedvalue=

Authorization types: Basic, Digest, Bearer
Digest: Used to hash the credentials using an MD5 hashing algorithm.
Bearer: Used to send tokens, which identify the client. They are commonly associated with the OAuth authorization framework.

POST Method:

- ➔ Places user parameters in the HTTP request body.
- ➔ Three main benefits:
  1) **Lack of logging:**
  2) **Less encoding requirements:**
  3) **More data can be sent:** The maximum URL length varies between browsers (chrome/firefox/IE), Web Servers (Apache/Nginx/IIS), Content Delivery Networks (Cloudflare/CloudFront/fastly) & URL shorteners (bit.ly/amzn.to). URLs should be kept to below 2000 characters.

Note: Session IDs are almost always hashed. This can be identified by: Containing only hexadecimal characters (a-f) (0-9) & Being divisible by 8.

Every application user has a unique session ID, which maps to either file or database entry on the server, stores information about user login.

Content-type:
It tells a web server what type of content to expect.
For example, content-type: application/x-www-form-urlencoded
Tells the server to expect something like: param1=value1&param2=value2.

Another content type like application/JSON (javascript object notation – simplified way to store and transfer data and easy to intercept and parse)

PUT & DELETE Method:

These methods are usually allowed on Web Distributed Authoring and Versioning (WebDAV) servers.
WebDAV: extension of HTTP and used for remote management of files & dirs.
It is popular CMS and blogging app enabling authors to edit and upload data.
Various types of WebDAV methods: PUT, DELETE --- our main
PROPFIND, PROPPATCH, MKCOL, GET & HEAD & POST (for collections)
, COPY, MOVE, LOCK, UNLOCK
Supported methods can be found by: OPTIONS method in request (useful to enumerate a list of methods allowed by server)

PUT: Allowed to overwrite an existing file on the server or create a new file.
DELETE: Allowed to delete an existing file on the server.

Working with command line

CURL – Client URL

- ➔ A command-line tool that supports HTTP along with many other protocols.
- ➔ Default request: GET
- ➔ It cannot render HTML and run javascript. (Entire response provide us in a row)
- ➔ Increase verbosity to view the row HTTP requests. [-v]
- ➔ It understands the standard URL format. We can specify creds in the URL.

For ex:
```
$curl http://user:password@website.com/ -vvv
```
Another way to specify credentials:
```
$curl -u user:password  http://website.com/ -vvv
```

CURL doesn’t redirect us to a specified location by default. To follow redirections: [-L]
```
$curl -u user:password -L http://website.com/
```

Get request with parameters:
```
$ curl -u user:password 'http://website.com/search.php?id=1'
```

POST method in CURL:
In the POST method of CURL, data can be passed using the -d flag.
Default content-type is application/x-www-form-urlencoded.
```
$ curl -d 'username=user&password=password' -L
http://website.com/login.php -v
```

Specify cookie usage in CURL using --cookie or --cookie-jar.
```
$ curl -d 'username=admin&password=password' -L --cookie-jar
cookies.txt  http://website.com/login.php
```

Header & Method:
To specify headers in CURL, use flag -H.
To specify the request method, we can use flag -X.

File upload:
```
$ curl -X PUT -d @test.txt http://website.com/test.txt -vv
```

========== XXX ==========

It’s not the end. There will be more apart from this.