System and Network Administration

Introduction to Linux

- Free and open-source Operating System. Has many distros:
  - **Free:** Debian, Ubuntu, CentOS, OpenSUSE, Mint, Gentoo, ...
  - **Enterprise:** Red Hat, SUSE, Oracle, ...
- Used as a server, desktop (alternative to Windows), in super-computers, and in many devices such as smartphones, routers, switches, firewalls, gaming consoles.
- **Recall:**
  - Kernel space vs User-space
  - Shell (`bash`, `sh`, `csh`, `ksh`)
  - Common shell commands and programs.
  - `ssh` and OpenSSH, PuTTy
  - Users and groups (`uid`, `gid`); `id` command to show them.
  - Permission system (**types:** read, write, execute, **by:** owner, group, all)
    - `useradd`, `adduser`, `passwd`, `su`, `groupadd`, `groupdel`
  - Ubuntu's Advanced Packaging Tool (APT) and `apt-get`, how to install software by adding repositories and/or compiling from source, `make`.
  - Red Hat Package Manager (`rpm`) and Debian packages `.deb`
  - `ls -lah` **output :**

    ```
    -rw-rw-r--  1 user group 3.4K Jul 24 10:27  file.txt
    ```

    1. **File permissions**
       - 1st bit can be `d` for directory, `-` for file, `s` for socket `p` for pipe, `D` for Door, `l` for symlinks, `c` for character special files.
       - 1st triple: owner permissions, 2nd: group, 3rd: everyone.
       - Numerical representation: `1=x, 2=w, 4=r`, thus: `3=-wx, 5=r-x, 6=rw-, 7=rws` which can be used with `chmod` alternative to `chmod +|-` syntax.
    2. **Number of links**
    3. **User**, 4. **Group**, 5. **file size**, 6. **last modified**, 7. **file name.**

Linux Directory hierarchy

Read more here

- `/` is the root directory, parent to all others
- `/bin`: binaries (e.g., `ls`, `cp`, ...)
- `/boot`: static files for boot loader.
- `/dev`: contains files for external devices attached to the system, (USBs, display, keyboard) as well as `/dev/null`

- `/etc`: config files local to the machine.
- `/home`: contains a subfolder for each normal (non-system) user with their name, containing user desktop files, documents, etc.
- `/lib`: software libraries shared by programs (e.g., `ld*`, `lib`, `*.so.*`)
- `/media`: mounted file systems of removable media (e.g., flash memory)
- `/mnt`: temporarily mounted file systems (user can mount FSs there)
- `/proc`: contains information about running processes, their memory and CPU utilization.
- `/root`: home directory for root, hidden from user, contains system files.
- `/var`: dynamic files; constantly being written to or changed (websites are typically stored there)
- `/srv` site-specific data served by the file system.
- `/sbin` binaries used by root (e.g., `fsck`)
- `/tmp` temporary files, cleaned regularly on boot or by a job
- `/usr` contains user application, typically stored under `/usr/local`
- `/opt` for optional application addons.

Highlighted bash commands, programs, and utilities

- `man` `echo`, `cp`, `mv`, `rm`, `cd`, `ls`, `cat`, `top`, `uname`, `more`, `less`, `df`, `du`, `free` `head`, `tail`, `cut`, `sort`, `uniq`, `awk`
- `service <name> start|stop|restart|status`
- `gcc`, `htop`, `grep`, `curl`, `wget`, `dpkg`, `make`
- Text editors: `vim`, `nano`, `emacs`
- Extracting `gzip` and `bzip` archives with `tar` (`-c` to compress, `-x` to extract)
- `sudo` can grant temporary root command execution to `/etc/sudoers`
  - `$` prefixes limited user names, `#` prefixes root user (`uid=gid=0`).
- `systemd` and `cron`
  - `crontab` syntax (`-e` to edit jobs, `-l` to list them).
- `iptables` rule-based firewall. Inserting `-I`, Listing `-L` , Deleting `-D` rules.
- System logs (general, kernel, authentication, mail, http, boot, database) under `/var/log`
  - Trick to empty logs: `> /path/to/large/logfile`
- **Some special characters in bash.**

```
~   Home directory
#   Comment
$   Variable expression
&   Background job
*   String wildcard
(   Start subshell
)   End subshell
\   Escape next character
|   Pipe (send output of left as input to right)
```

```
[   Start character-set wildcard
]   End character-set wildcard
{   Start command block
}   End command block
;   Shell command separator
'   Strong quote (quotes everyting until next ')
"   Weak quote ($ is evaluated inside)
<   Input redirect
>   Output redirect
2>  Stderr redirect
>>  Append to file
/   Pathname directory separator
?   Single-character wildcard
&&  logical AND
||  logical OR
!   logical NOT
.   This directory
..  Parent directory
```

- **Special $ shell variables**

```
$1, $2,...  are the positional parameters.
$@  is an array-like construct of all positional parameters.
$*  is the IFS expansion of all positional parameters.
$#  is the number of positional parameters.
$-  current options set for the shell.
$$  pid of the current shell (not subshell).
$_  most recent parameter (or the abs path of the command to start the
current shell immediately after startup).
$IFS    is the (input) field separator.
$?  is the most recent foreground pipeline exit status.
$!  is the PID of the most recent background command.
$0  is the name of the shell or shell script.
```

- **Some CTRL shortcuts**

```
CTRL+A  Move to the beginnging of the line
CTRL+E  Move to the beginnging of the line
CTRL+C  Interrup (SIGINT) the currently running command
CTRL+L  Clear screen keeping the last line
CTRL+R  Reverse shell (search for command in history)
CTRL+X  Lists possible filename completions
CTRL+Z  Suspend (SIGSTP) executing command, resume with fg|bg <PID>
```

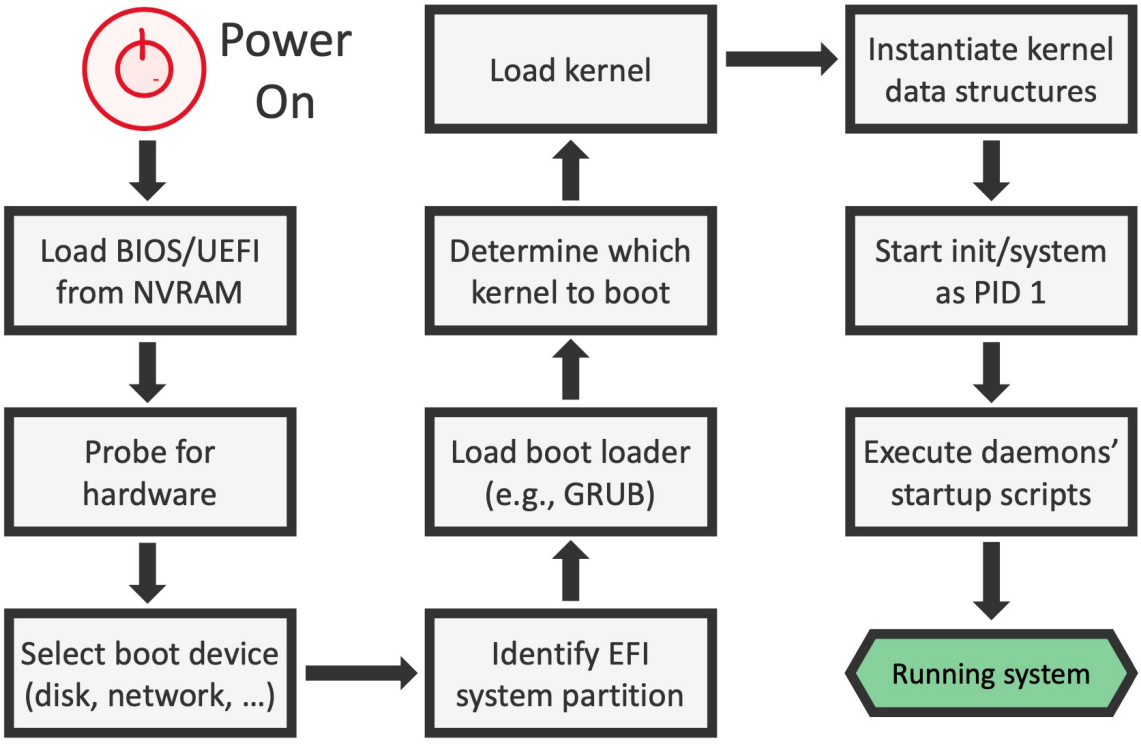
Networking in Linux

- Network interface naming standard (from thr output of `ip` or `ifconfig`)
  - `lo` for loopback interface (localhost)
  - `eth0` for Ethernet interfaces [check]
  - `wlo` for wireless network interface
- `netstat` displays active TCP connections routing tables and other network statistics.
- `nc` for reading and writing to network connections using TCP/UDP.

- `lsof` to list open files and the processes that are using them.
- `route` to view and manipulate routing tables.
  - `ip route add|list|delete`
- `ping` for testing reachability and measure RTT.
  - `hping`: packet generator and analyzer.
- `nmap`: a security scanner to discover hosts and services on networks.
  - **Host discovery** (e.g., listing the hosts that respond to TCP and/or ICMP requests or have a particular port open).
  - **Port scanning**: enumerating open ports on target host(s)
  - **Version detection**: determine the running server application name and version number.
  - **OS detection**: determine OS and hardware characteristics of network devices.
- Common Linux server applications

> **HTTP:** Apache `httpd`, `nginx`
>
> **SQL:** Mysql `mysqld`, SQLite, `postgresql`
>
> **FTP:** Proftpd, PureFTPd, vsFTPd, Filezilla.
>
> **DNS:** Bind.
>
> **SMTP:** postfix, qmail, sendmail.
>
> **POP3 / IMAP:** Dovecot, Courier
>
> **Remote access server:** OpenSSH

Linux boot process



Processes in Linux

- `/sbin/init` is the parent to all other processes with `PID=1`
- The only way to get new processes in Linux is to `fork` other processes, parent and child start from the same state (with different PIDs) and may diverge afterwards.
    - When child terminates, it becomes a `zombie` (maintain an entry in the process table, still reserve it's PID), after the parent reads it's exit state, it's reaped (wiped, removed completely) by the system.
    - If parent terminated before child, child becomes `orphan`, and is adopted by `init`, after child terminates it becomes a `zombie`, a process that is a `zombie` and an `orphan` is automatically reaped by the system.
- **Background processes**: have no GUI and doesn't require interactive user input.
- **Worker:** a process that runs in the background to execute a specific task, terminates when done, typically returning results to its parent.
- **Deamon:** a software program or a process always running in the background waiting for client(s) requests, it forwards synchronous client requests to workers asynchronously.
    - Client waits for daemon to return a response, while daemon doesn't wait for worker, the worker notifies the daemon (calls-it-back) when it finishes.
- **Service:** a contract between clients and a daemon (e.g., `sshd` daemon that is always running and listening at port 22 offers clients the service of secure shell communication through `ssh` protocol)
- **Highlighted commands**
    - `pstree` shows the tree of processes (with `init` at root) in a text format.
    - `pidof <process_name>` gives the PID of process.
    - `ps` shows a static snapshot of the currently running processes associated with current user and terminal, it displays Process ID, terminal name, cumulative CPU time, process state, and the command executed.
        - `-e` or `-A` to show every (all) process in the system.
        - `-f` to make the output formatted
    - `ps aux` is commonly used (different from `ps -aux`)
        - a = show processes for all users.
        - u = display the process's user/owner.
        - x = also show processes not attached to a terminal.
    - `telnet [-I user] host port` (terminal over network) is used to connect to a remote host using telnet protocol.
    - `cat /etc/services` contain the list of services installed on the machine, ports they listen on and protocol tcp/udp, optional aliases can be specified.
    - `apropos daemon` shows command-line tools related to daemons.
- **Common daemons**

> Managing other daemons (`xinetd`, `inetd`)
> Remote access (`sshd`)
> Domain name resolution (`named`)
> Authentication (`ldap`, `ldaps`)
> Mail daemons (`smtpd`, `imapd`)
> Web servers (`httpd`, `nginx`)
> File services (`ftpd`, `nfsd`, `mountd`, `lockd`, `statd`)
> Cloud computing (xen server)

> Microservices (`dockerd`, `containerd`)
> Logging daemons (`klogd`, `syslogd`)

Domain Name System (DNS)

- A globally distributed database with information about different network resources, stored as Resource Records (RRs)

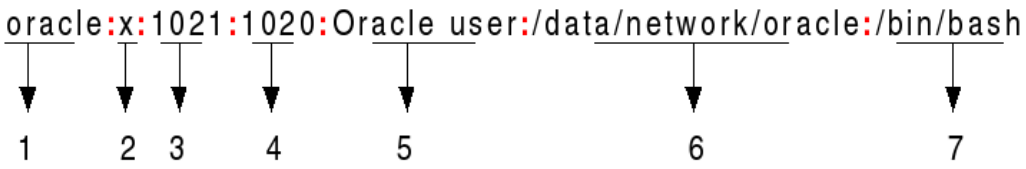
Type	*Mnemonic*	*Description*
1	A	**Address.** A 32-bit IPv4 address. It converts a domain name to an address.
2	NS	**Name server.** It identifies the authoritative servers for a zone.
5	CNAME	**Canonical name**. It defines an alias for the official name of a host.
6	SOA	**Start of authority**. It marks the beginning of a zone.
11	WKS	**Well-known services.** It defines the network services that a host provides.
12	PTR	**Pointer.** It is used to convert an IP address to a domain name.
13	HINFO	**Host information.** It defines the hardware and operating system.
15	MX	**Mail exchange**. It redirects mail to a mail server.
28	AAAA	**Address.** An IPv6 address (see Chapter 26).
252	AXFR	A request for the transfer of the entire zone.
255	ANY	A request for all records.

- **RR syntax:** `<NAME>, <CLASS>, <TYPE>, <TTL>, <RD Length>, <RDATA>`, some fields can be omitted, example `google.com IN A 216.239.38.120` (IN for Internet records)
- **DNS structure**
  - **DNS server = Name server:**
    - Storing RRs (e.g., Google 8.8.8.8, CloudFlare 1.1.1.1)
    - Answers queries from client (e.g., hostname to IP address), caches answers for performance, and communicates with other name servers to keep DNS data synchronized.
    - **Authoritative DNS server** has the latest, trustworthy records for a specific resource it's responsible for.
      - There can be one **master (primary)** authoritative server that propagate changes to many authoritative **slaves (secondary servers)**.
      - Or the RRs can be **distributed** among authoritative servers.
    - **Non-authoritative DNS server** forwards requests to authoritative servers, it may cache records to provide (non-authoritative) answers quickly later.
    - **A recursive DNS server** - when queried - will keep asking other servers - possibly gets redirected - and returns the final results to the querier.
    - **A non-recursive DNS server** - when queried - will either provide the results directly to the querier, or will redirect him to another server to ask.
      - Non-recursive DNS servers should not be used in `resolv.conf` (see below) as they can simply answer "unknown" and it will be accepted.
  - **DNS client = resolver:** client side of the DNS, responsible for initiating and sequencing the appropriate queries to the name server, leading to the full hostname to IP translation.
    - DNS daemon at client communicates with DNS server through port 53, it uses udp for requests and tcp for zone transfer.
- **Questions:**

- **What DNS servers to use?**
  - Each host joining a network is provided (statically or through DHCP) with the IP address of a primary and secondary DNS server (if the primary fails).
  - Prioritized list of DNS nameservers to use is stored at `/etc/resolv.conf`
- **How to use the DNS servers?**
  - The application requesting DNS will communicate with the recursive DNS resolver (through a stub resolver) which will query the DNS server and returns results.
  - **DNS tools**
    - `host`, `dig` , `nslookup` are used for performing DNS lookups (hostnames to IP addresses)
    - `dig` is preferred for debugging (it has more options and output is a raw record), while `host` and `nslookup` are more user-friendly.
    - Examples: `host example.com 8.8.8.8`. `dig @8.8.8.8 example.com in A`, `nslookup example.com`
  - `/etc/resolv.conf`
    - Nameservers under the file can be modified directly, although not recommended.
    - A `search <suffix>` entry can be added to the `resolv.conf`; Then, if the DNS can't resolve some request like `ping <host>`, it will try `ping <host><suffix>`
    - This is helpful if the LAN environment has it's own DNS server (e.g., querying `university` will automatically resolve to `university.innopolis.ru` if there is an entry `search innopolis.ru` in `resolv.conf`)
    - For changes to take effect, the DNS server should be restarted (e.g., `sudo service bind9 restart`)
- **When to use DNS servers?**
  - For translation between hostnames and IP address, host/mail server aliasing, or load distribution.
  - `/etc/nsswitch.conf`
    - A file used by C and other applications to decide where to get **name-service information** (from which database/file, or whether to query DNS/NIS/LDAP) and in which order.
      - **Directory/name service** maps names of network resources to their respective network addresses (like a phonebook)
      - In other words, it tells the system to look for X in [Y] where
        - **X can be** a specific user/group name/password, hostname, network, protocol, service, etc.
        - **Y may include (in any order):** local database, OS files, DNS, LDAP (check below), NIS with rules on what to do if the resource was not found.
    - **Example:** `hosts: dns [!NOTFOUND=return] files` will try DNS lookup first to get RRs from hostnames, if not found, it will check files (e.g., `/etc/hosts`)
- **When to use /etc/hosts?**
  - Hosts file stores (IP, hostname) pairs that override the ones provided by the DNS (unless configured otherwise in `/etc/nsswitch.conf`)

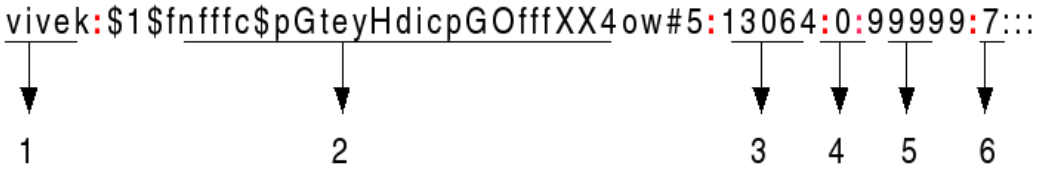
Identity and Access Management

- **Concepts:**
  - **Identification:** user claims an identity (by providing a username, email, etc.)
  - **Authentication:** user proves the claimed identity (password, fingerprint, etc.)
  - **Authorization:** user is given access to a specific set of privileges or resources based on their proven identity.
  - **Single Sign-On (SSO):** technology allowing users to login once, get a TTL-limited token (saved in browser), and use it to access many services without having to login to each one.
  - **Lightweight Directory Access Protocol**
    - LDAP can be considered a lightweight database, often stored as a tree-like structure, optimized for small number of writes and many reads.
    - It contains structured information about users, systems, sub-networks, services, and/or applications in the network.
    - LDAP is commonly used as a central place of usernames, their passwords, and privileges, allowing applications or services to connect to LDAP and authenticate/authorize users.
    - **To implement such LDAP authentication system, one needs:**
      - A directory service software (e.g., Microsoft active directory)
      - A tool for sysadmin to add entries, etc. (e.g., phpLDAPadmin)
      - A mechanism for authenticating users
        - Users directly submit their credentials to LDAP for authorization.
        - Or using a 3rd party authentication protocol (e.g., Kerberos) for secure authentication over an non-secure network.
- **Highlighted files:**
  - `/etc/passwd` entry example



    1. User name
    2. x means that password stored as a hash in `/etc/shadow`
    3. User ID (root has 0)
    4. Group ID (root has 0)
    5. User Info (GECOS): fullname, phone number, etc.
    6. Home directory for the user.
    7. Absolute path of login shell to use, a system user like root can use a placeholder `/sbin/nologin`

  - `/etc/shadow` entry example



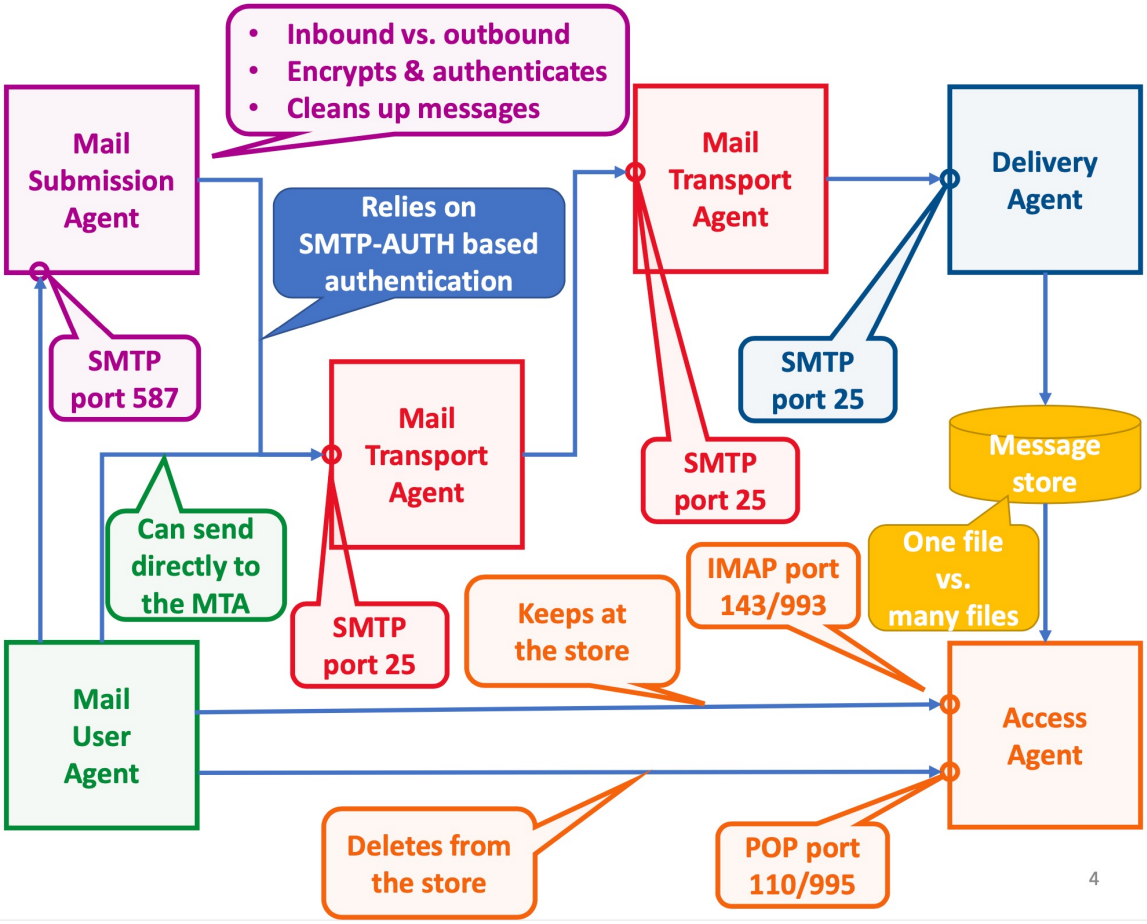
> 1. **Username**
> 2. Encrypted **Password**, format: `$id$salt$hashed` where `$1$` is MD5, `$2a$` is Blowfish, `$2y$` is Blowfish, `$5$` is SHA-256, and `$6$` is SHA-512
> 3. **Last changed** timestamp (days since epoch 01.01.1970)
> 4. **Minimum** delay required between password changes
> 5. **Maximum** number of days the password is valid (after which the user is forced to change password)
> 6. **Warn:** password expiration reminder (7 means a week before expiry).
> 7. **Inactive:** number of days after which an account with expired password is disabled.
> 8. **Expire:** account expiry timestamp (days since epoch)

  - `/etc/gpasswd` entry: `<group_name>:x:<gid>:<list_of_members>`
  - `/etc/gshadow`: same syntax as `/etc/shadow` for groups.

- **Highlighted commands:**
  - `passwd`: to change user password
  - `chfn`: change user fullname (and GECOS)
  - `su <uname>`: switch to execute commands as another user.
  - `id`: get user or group id (`-u <uname>`)
  - `strace`: trace system calls and signals
  - Default in Ubuntu: `ll` is an alias to `ls -al`

- **Highlighted C functions:**
  - `getpwuid(uid)` returns entry for used with `uid` from `/etc/passwd` as a `struct passwd`
  - `getpwnam(name)` same as `getpwuid(uid)` but takes username as a parameter.
  - `getpwent`: returns the next entry from passwd file (e.g., the local password file `/etc/passwd`, NIS, or LDAP)
  - `setuid(uid)`, `setgid(gid)`: used to change the effective id of the calling process (execute commands as other users or groups, without actually being logged in as them)
    - **Example:** for a C script to be able to restart `bind9` DNS server service, it has to be executed by bind system user, so we use `setuid` with the number we get from `id -u bind`
  - `seteuid()`, `setegid()`: same as the above ones, but if root (or root) wants to get lower privileges temporarily and then regain root privileges, it should use `seteuid(), setegid()`
  - `setreuid(ruid, euid)`, `setregid(rgid, egid)`: sets real and/or effective ids at ones.
  - `getuid()` returns the real user id of the calling process.
  - `geteuid()` returns the effective used id of the calling process.

E-Mail

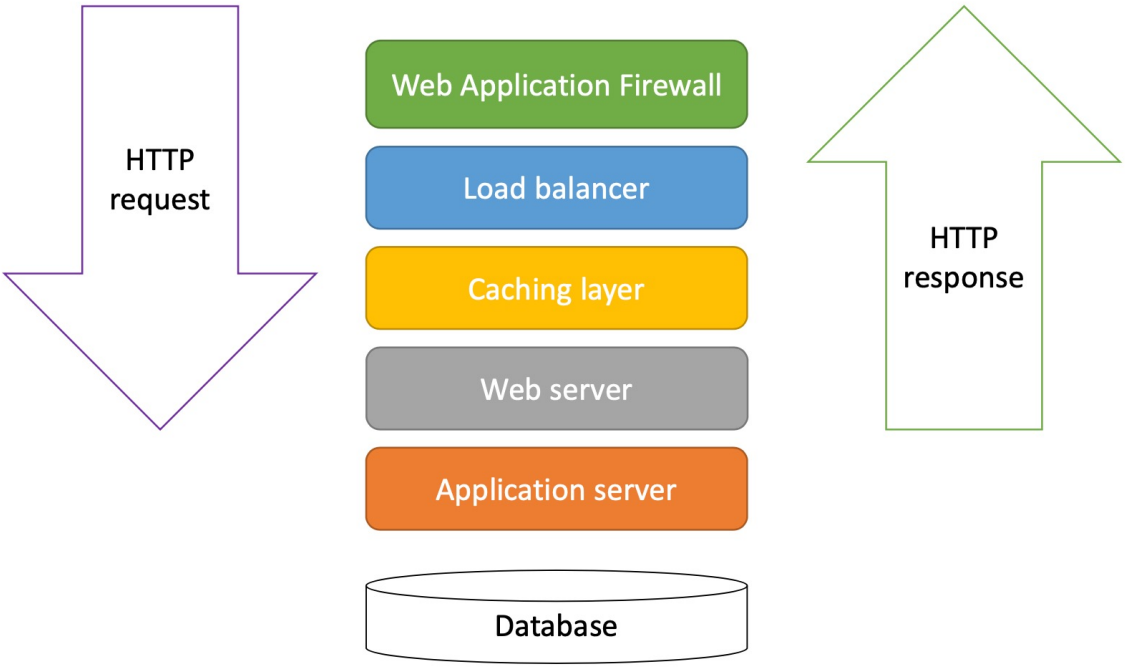


- **Mail User Agent (MUA):** software to read and compose emails.
  - `/bin/mail`, Thunderbird, Outlook, etc.
- **Mail Submission Agent (MSA):** interface between MUA and MTA.
- **Mail Transport Agent (MTA):** program to route emails among machines.
  - `sendmail`, `exim`, `exchange`, `postfix`
- **Delivery Agent (DA):** sorts incoming mail into a local **store**, filters out spam, scans for viruses, etc.
  - `procmail`, `maildrop`
- **Access Agent (AA):** connects MUA to the **store** (IMAP/POP server)
  - `cyrus`, UW IMAP

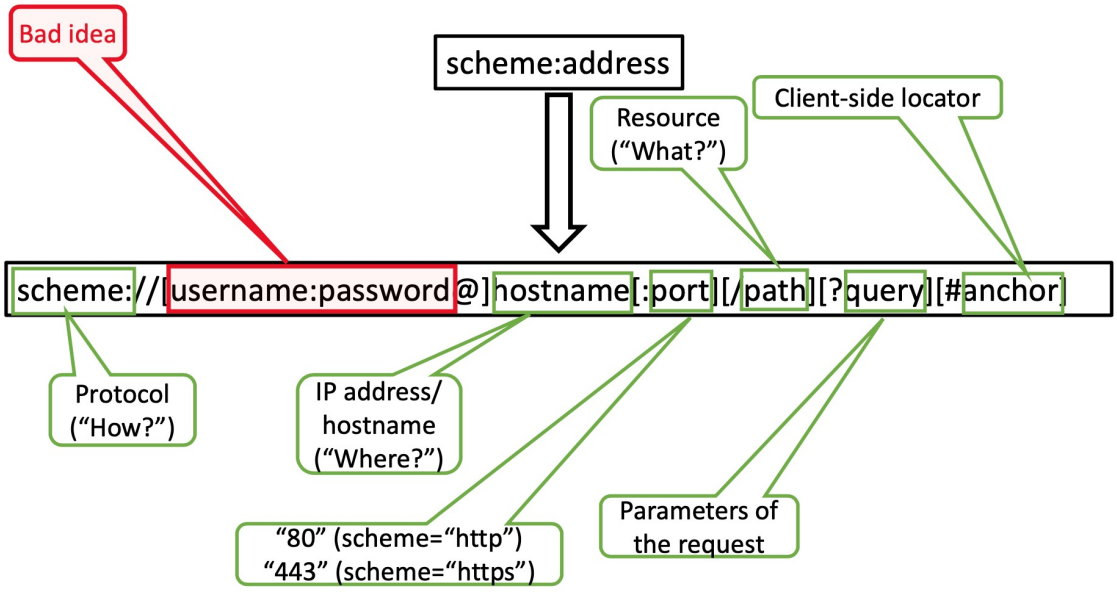
Hyper-Text Transfer Protocol (HTTP)

- Main protocol for retrieving and viewing web pages.

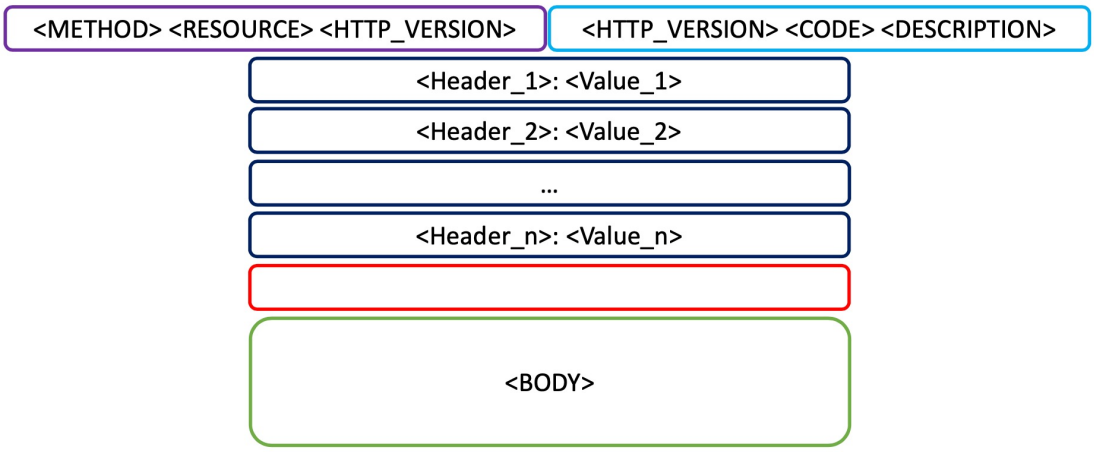
Web application stack



Unified Resource Locators



HTTP message structure



Commonly encountered headers

Name: example value	Direction
Host: www.admin.com	C
Content-Type: application/json	CS
Last-Modified: Wed, Sep 7 2016	S
Cookie: flavor=oatmeal	C
Content-Length: 423	CS
Set-Cookie: flavor=oatmeal	S
User-Agent: curl/7.37.1	C
Server: nginx/1.6.2	S
Upgrade: HTTP/2.0	C
Expires: Sat, 15 Oct 2016 14:02:...	S
Cache-Control: max-age=7200	CS
Referer: https://developer.mozilla.org/en-US/docs/Web/JavaScript	C

- **Some security headers/flags:**
  - `X-XSS-Protection:` when set to 1, stops pages from loading when (Cross-Site Scripting) is detected, now unnecessary due to CSP.
  - `X-Frame-Options`: when set to `SAMEORIGIN`, allows embedded HTML frames only from the same websites.
  - `Cache-Control`: specifies rules on requests/responses caching in user browser (private cache) or Proxies/CDN (shared cache, shouldn't store personal tokens)
    - `Cache-Control: max-age=N` sets a lifetime for a request/response, after which it cannot be reused.

    - `Cache-Control: private|public|no-store` allows the response to be cached only in user machine, anywhere, or not stored at all, respectively.
- `HttpOnly` flag specified on a certain cookie value will allow the cookie to be accessed only through HTTP and not from JavaScript.

HTTP request methods

Method	Safe?	Description
GET		Retrieve the resource
HEAD		GET without the body
DELETE		Delete the resource
POST		Apply the request to the resource
PUT		Replace existing contents
OPTIONS		Show supported methods

HTTP response classes

Code	General indication
1xx	Request received; processing continues
2xx	Success
3xx	Further action needed
4xx	Unsatisfiable request
5xx	Server or environment failure

HTTP server types

Type	Purpose	Examples
Application server	Runs web app code, interfaces to web servers	Unicorn, Tomcat
Cache	Speeds access to frequently requested content	Varnish, Squid
Load balancer	Relays requests to downstream systems	Pound, HAProxy
Web app firewall	Inspects HTTP traffic for common attacks	ModSecurity
Web server	Serves static content, couples to other servers	Apache, NGINX

VirtualBox Network Adapter Settings

- When creating VMs with VirtualBox, we can add up to 4 network interfaces to each guest, an interface will act as a physical NIC with MAC address and can be seen in networks settings in the guest OS, each interface can have one of the following types:
  - **NAT**
    - VirtualBox does **address translation** from guest to host, allowing the guest to use Internet through the host's public IP address (if any).
    - Host and guests **cannot** communicate directly with each other.
  - **NAT Network**
    - Same as NAT, except now the guests form a network (they can communicate with each other, **as well as the Internet** through the host's public IP address).
    - Guests can ping with each other and host, but **host cannot ping VMs**
  - **Bridged adapter**
    - VirtualBox will act as a switch connecting the guest to some interface on host (chosen by the admin).
    - Guests will **act as physical machines** connected to the same network as host.
    - Guests **can also access the Internet** through the host's public IP address.
  - **Internal Network**
    - Connects the VM to an isolated internal network of VMs, each one is identified by a name (set by the admin).
    - Guests are isolated and **cannot** access the internet.
  - **Host-Only Adapter**
    - VMs can communicate with each other and the host only (but they are not part of the host's network).
    - VMs cannot access the Internet.

SSL/TLS Certificates

- **Problem #1**: Communication between client and server cannot happen in plaintext.
  - Intruder can see (eavesdrop), modify (intercept), or add its own messages to the connection (inject).
- **Solution #1:** use a symmetric key algorithm to encrypt and decrypt data.
  - Think of the algorithm as two functions, everyone knows their implementation details.
    - `encrypt(plaintext, key) = ciphertext`
    - `decrypt(ciphertext, key) = plaintext`
  - **Block cipher** means that plaintext is firstly split into blocks of a certain length before encryption, in contrast with **stream ciphers**, that encrypt or decrypt one byte at a time.
  - Keys are just a sequence of bits, there are standards (**AES, DES)** for generating keys of different lengths.
- **Problem #2:** how can client and server know about the same key securely?
  - Key cannot be just sent through the network, since again it can be eavesdropped and used by intruders.
- **Solution #2:** use asymmetric key algorithm to share the symmetric key securely.
  - Think of the algorithm as two functions, everyone knows their implementation details.
    - `encrypt(plaintext, publicKey) = ciphertext`
    - `decrypt(ciphertext, privateKey) = plaintext`
  - **publicKey** and **privateKey**
    - A pair of keys of a certain length generated by a certain algorithm (e.g, RSA-4096)
    - A message encrypted with one of them **can only be decrypted** with the other, therefore, the following is also true:
      - `encrypt(plaintext, privateKey) = ciphertext`
      - `decrypt(ciphertext, publicKey) = plaintext`
    - The convention is that, public key can be shared with everyone (send on the network), while private key (identity) should not be shared with anyone at all.
  - **How it works:**
    1. Server has generated a pair of keys `publicKey`, `privateKey`
    2. Server sends `publicKey` to whoever wants to communicate with it (i.e., client)
    3. Client generates a symmetric key `k`
    4. Client sends `encrypt(k, publicKey)` to the server
    5. Only server can do `k = decrypt(k, privateKey)`
    6. Client and server then use `k` to communicate securely.
- **Problem #3**: intruder can act as a fake server
  - Intruder can send its own public key and communicate with client exactly as if it's the server.
  - Client can never know that the public key it received indeed belongs to the server.
- **Solution #3:** use digital certificates
  - A digital certificate is file with some attributes and values.
  - It proves that a certain entity owns a certain public key.
  - It's Issued by a certain Certification Authority (CA) that also has `CA_publicKey` that is shared with everyone and `CA_privateKey` that is only known by them.
  - **How it works:**
    - Server sends its `publicKey` to the client along with the `certificate` that verifies that it owns the public key.

        - `certificate = encrypt(publicKey, CA_privateKey)`
    - Everyone knows `CA_publicKey`, client can then verify
        - `publicKey == decrypt(certificate, CA_publicKey)`
        - If mismatch happened, server is fake.
        - Otherwise, client can use server's public key safely.
- **Problem #4:** how can the client knows that `CA_publicKey` indeed belongs to the CA?
    - There is no single CA in the world, there are many, intruder can claim to be a CA and fake all this process acting as the server.
- **Solution #4:** root CAs.
    - For a CA to be trusted, it should sign (encrypt) its public key using the private key of another higher-authority CA.
    - A client OS or web browser trusts only a certain number of hard-coded root CAs.

      > As of 24 August 2020, 147 root certificates, representing 52 organizations, are trusted in the Mozilla Firefox web browser, 168 root certificates, representing 60 organizations, are trusted by macOS, and 255 root certificates, representing 101 organizations, are trusted by Microsoft Windows.

    - A client follows this chain of trust verification until it finds a root CA that it trusts by default, or produces an error saying that the user that the server certificate couldn't be trusted.
    - Root CAs are self-signed, meaning that the verification will always be true:
        - `root_ca_cert = encrypt(root_ca_public_key, root_ca_private_key)`
- [Let's Encrypt]
    - A nonprofit Certificate Authority providing TLS certificates to 260 million websites.
    - The objective is to make it possible to set up an HTTPS server and have it automatically obtain a browser-trusted certificate, without any human intervention. This is accomplished by running a certificate management agent on the web server.
    - First, the agent proves to the CA that the web server controls a domain. Then, the agent can request, renew, and revoke certificates for that domain.