Securing Internet communications

Francesco PALMIERI
Università di Salerno

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

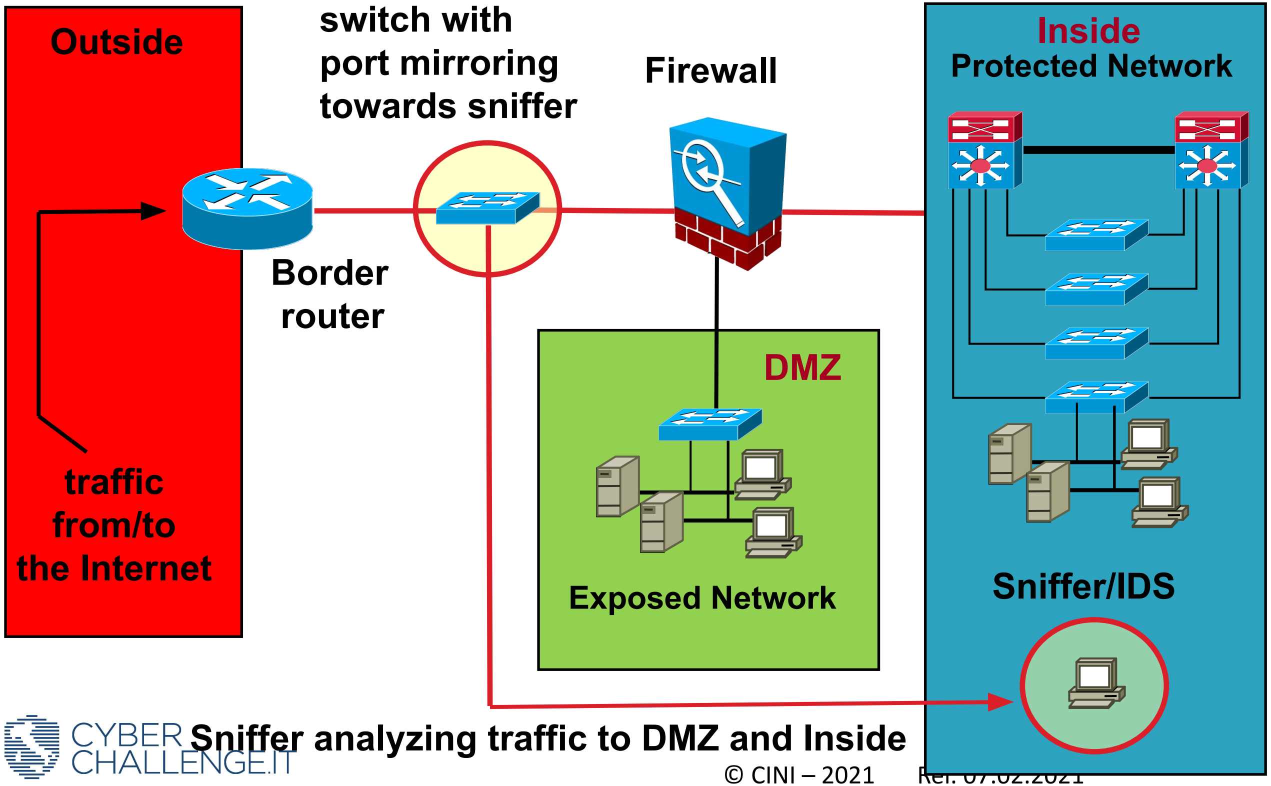
Topics

- Security enforcement devices
- Traffic filtering strategies and policies
- Network access control: techniques and tools
- Implementing simple network access control policies

Current Topic

- Security enforcement devices
- Traffic filtering strategies and policies
- Network access control: techniques and tools
- Implementing simple network access control policies

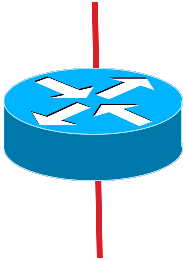
Basic security architecture

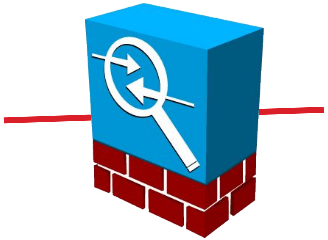


- In a common network architecture there are at least three domains:
  - **Outside** (all the world outside - the Internet): trust degree 0
  - **Inside** (the internal organization to be protected and hidden): degree of trust 100
  - **DMZ** (the set of internal machines that expose services outside): degree of trust 0 <x <100

Security enforcement devices

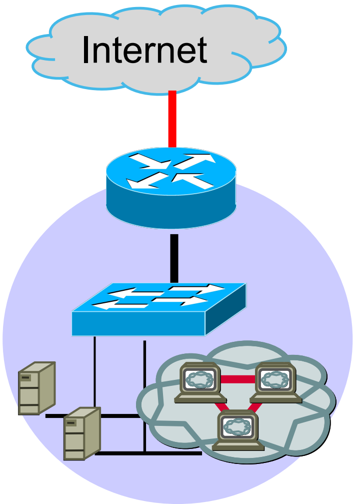
- In a common security architecture in order to implement security policies we can rely on two perimeter control devices:
  - **Routers** (typically those located on the network border)
  - **Firewalls**





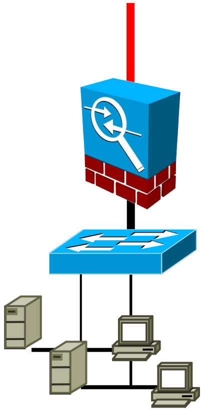
Border Router

- The border router is the first barrier protecting its internal network
  - difficult to circumvent by malicious end-users
- It allows the centralization of a good number of security checks
- Its protection is fundamental
  - a compromise may open access to the internal network
  - an inadequate filtering policy can expose the internal network to attacks
  - corruption of routing tables can cause disruptions and unauthorized access to data
- A properly configured router can minimize effects from internal sites compromised by attacks or hostile activities (insider threats)



Firewall

- Firewall is an english term with original meaning of a fire isolation barrier
- It is the main passive perimeter defense component
- It has security enforcement tasks, in the broadest sense of the term, with the aim of controlling traffic between two or more networks:
  - allowing only what is specifically authorized by the security policy
  - detecting and reporting any attempts to violate the security policy
  - possibly carrying out additional auditing and accounting functions
  - it can also connect at the link or network layer two or more network segments



Why installing a firewall

- To implement a security policy:
  - Able to allow controlled access to systems or services of a protected network:
    - Authorized users only
    - Only to authorized systems
  - Able to allow users and systems of a protected network to access the systems and services of an outside (untrusted) network in a controlled way:
    - only if the risk is acceptable
    - recording all their activities

Firewall: pros

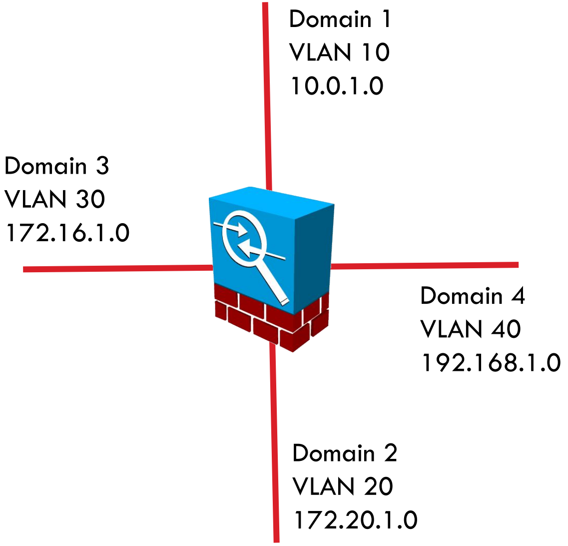
- Centralization of security policies
  - Can result in a Single point of failure (can be a disadvantage)
- Relying on a special purpose solution able to optimize traffic filtering operations (through appropriate HW)
- Ability to inspect traffic from data link to application layers
- Stateful control of sessions

Firewall: cons

- Difficulty in coping with non-trivial protocols
- Performance / throughput
  - It can turn into a bottleneck
  - user perception can be negative due to service limitation
- Complex management
  - Configuration requires specialization
  - Verification and analysis of logs is not straightforward
- Excessive sense of trust and internal insecurity
- High costs for performance beyond Gigabit

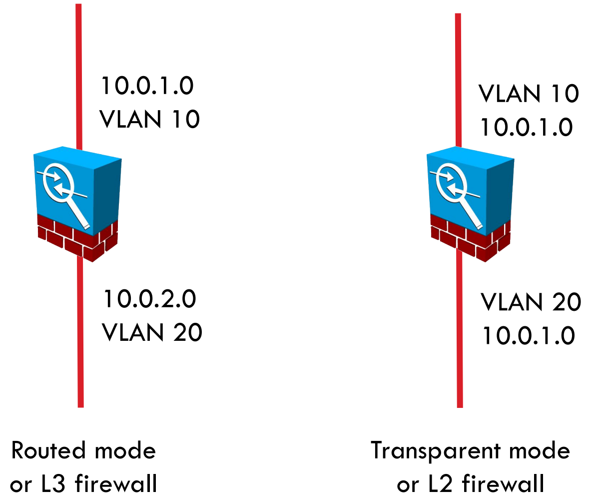
Implementation and basic functions

- Network device with at least 2 network interfaces
  - Each interface identifies a **separate security domain** on a different network segment (VLAN)
- Can remap IP addresses (NAT)
- Filters traffic between different zones/domains through predefined rules (access control policies)
- It can mediate access to specific applications for control and inspection purposes:
  - Proxy service access
  - Content filtering (selective content filtering)
  - Deep packet inspection and traffic analysis
  - Enforce bandwidth limitations on specific traffic types



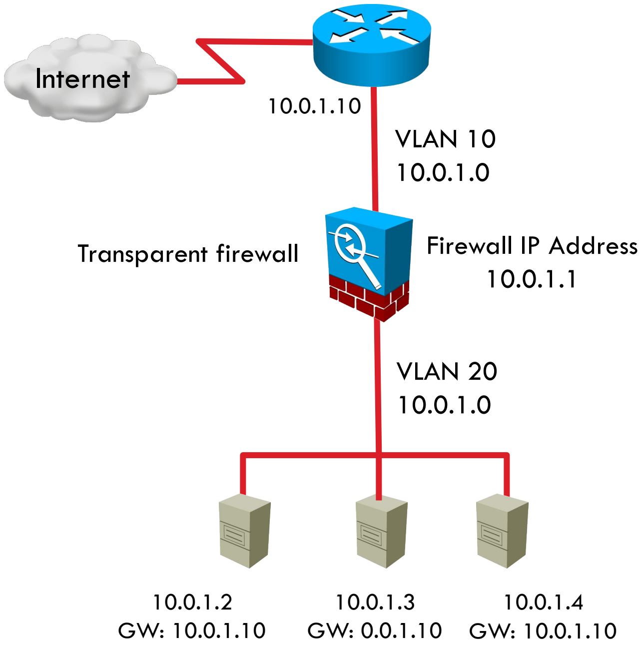
Firewall: operating modes

- A firewall can operate in two ways:
  - **Routed**: Operates at level 3, segmenting different networks based on IP addresses
  - **Transparent**: Operates at level 2, segmenting on MAC address basis
- A routed firewall looks like a layer 3 device and needs an IP address/network on each interface associated to a segment
  - Routes IP/IPv6 traffic between the various interfaces
  - Supports the most common routing protocols



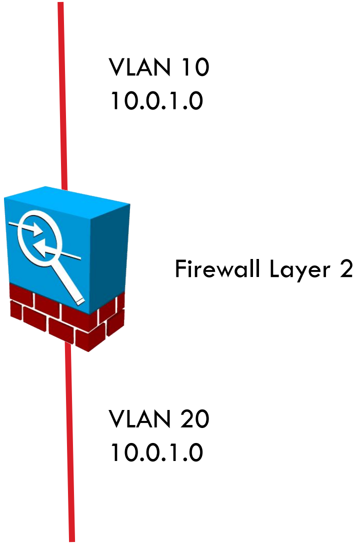
Transparent mode operations

- Layer 3 traffic must be explicitly allowed to pass through the firewall
  - However it performs packet screening/filtering from network to application layers
- The segments connected to the interfaces must be on the same layer 3 subnet
- The firewall IP address must not be configured as the default gateway for connected devices
  - Devices must point to the router ahead of the firewall (passed through transparently)
  - Each interface identifies a different segment/VLAN even if associated with the same IP network (the firewall bridges different segments)



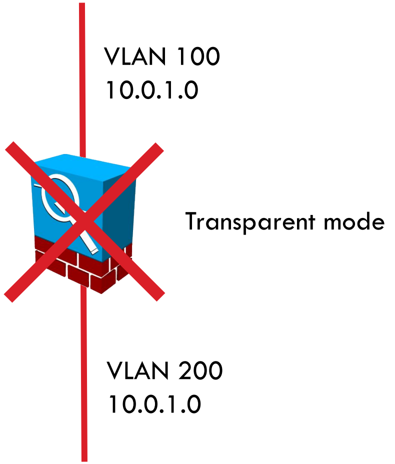
Transparent mode benefits

- Flexible, integrated and easy to manage:
  - IP-level redirection not required
  - No NAT to configure
  - Routing and redirection problems cannot occur (it does not perform routing)
- Totally invisible from the outside
- Greater robustness



Transparent mode unsupported features

- The following features are typically not supported by a firewall in transparent mode:
  - NAT
  - Routing protocols (e.g. OSPF, RIP, BGP)
  - IP / IPv6
  - DHCP relay
  - QoS
  - Multicast
  - VPN termination

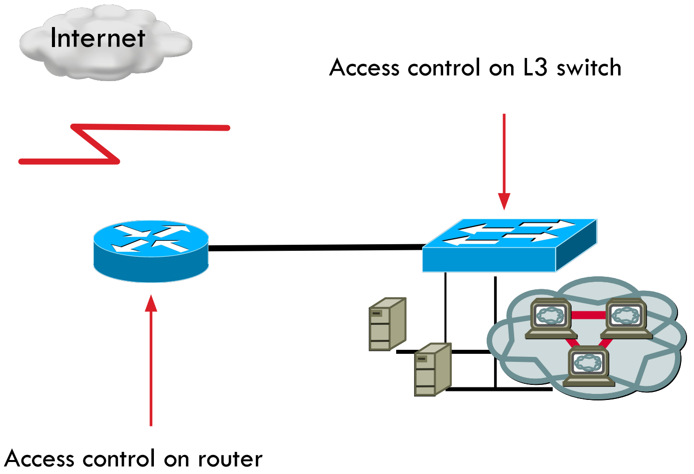


Current Topic

- Security enforcement devices
- Traffic filtering strategies and policies
- Network access control: techniques and tools
- Implementing simple network access control policies

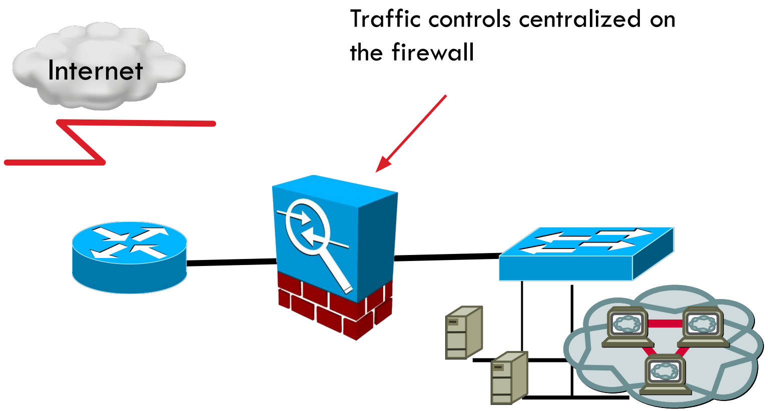
Traffic filtering: putting controls on border router

- Router and Switch Layer 3 devices provide simple access control mechanisms
  - Based on stateless traffic filtering
  - Filters based on IP address and TCP/UDP Ports
- The use of complex controls with a significant number of filtering clauses, entails a certain increase in CPU load in the forwarding activity.
  - can be acceptable if the Router or L3 Switch provide hardware implementation of access control mechanisms
- The truly advantage is that such devices are already present in any network, partitioning it in a natural way



Traffic filtering: putting controls on firewalls

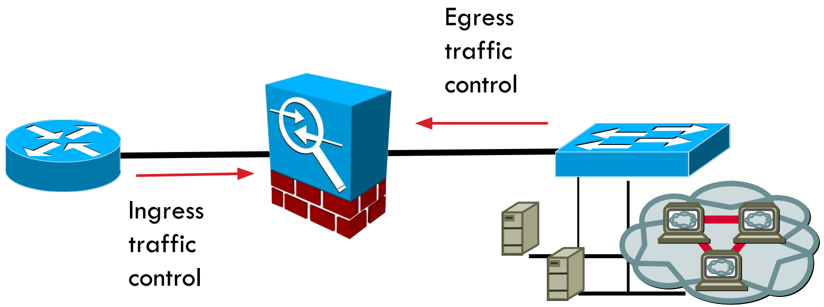
- The introduction of a firewall reduces the CPU burden associated to traffic control/filtering activity on routers or L3 switches
- The centralization of control policies on the firewall constitutes a significant advantage from the management perspective:
  - reduces the configuration complexity
  - centralizes the management of logics and filtering problems
  - It allows you to simultaneously protect thousands of machines



- This policy does not scale in the presence of large traffic volumes and becomes a performance bottleneck that can be exploited to create DoS

Choosing the right filtering location

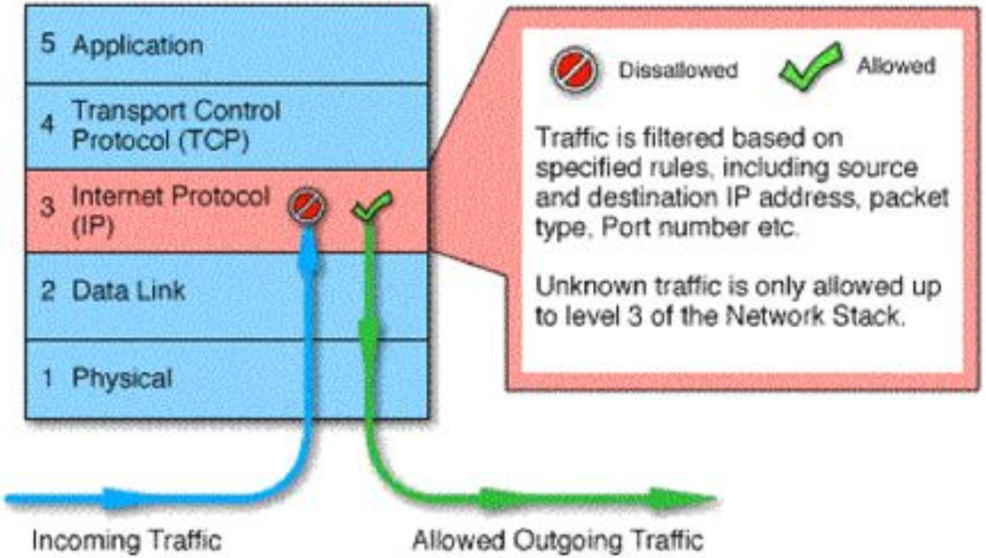
- **Ingress** filtering:
  - natively knows from which interface packets are coming in
  - provides protection of internal networks
- **Egress** filtering:
  - enables checking also the locally generated traffic exiting the domain
  - blocks what should not come out



- The controls should be located **as close as possible** to the traffic origin

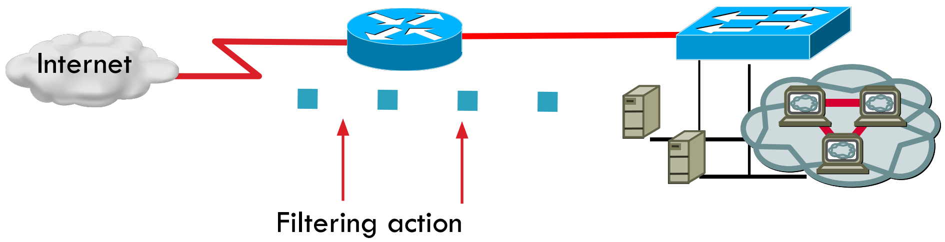
Filtering Parameters

- IP Header
  - source address
  - destination address
  - protocol
  - flags, options (source routing)
- TCP/UDP Header
  - source port
  - destination ports
  - flags TCP (SYN, ACK)



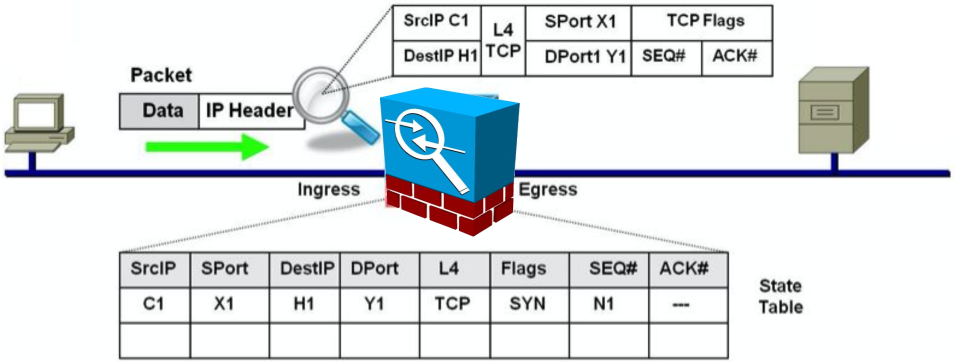
Stateless filtering (router)

- Based only on IP addresses, TCP / UDP ports (source and destination), and protocol
- Controls are carried out independently one packet at a time (no memory/state)
- There is no perception of the flow of packets belonging to an end to end connection
- Packets can also come from interfaces other than the one on which they exit (asymmetric forwarding phenomena are tolerated)



Stateful filtering (firewall)

- When a new connection is established, if the filtering rules do not block it, then the related information are used to add an entry (**session**) of a **connection status table**.
- Subsequent incoming packets will be handled according to their belonging to one of the active connections (or data flow sessions) whose status is saved in the table.
- When the connection is terminated, the corresponding entry in the table is deleted
- The table contains:
  - Unique session ID
  - Connection status (*handshaking*, *established*, *closing*)
  - Packet sequencing information
  - Source and destination addresses/ports
  - Network interfaces used



Filtering (access control) policies

- Before defining any filtering policy aimed at performing access control, a careful preliminary assessment must be made, by considering:
  - Who needs access?
  - When and how?
  - From where?
  - At which time/date?
  - What services does it need?
  - What protocols does it use?
  - What QoS (e.g bandwidth) does it require?

Filtering (access control) policies

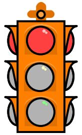
A firewall (or router) can operate in two diametrically opposite ways:

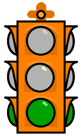
- **Deny All**: Anything that isn't specifically allowed is denied. **High security**
  - Block all traffic and each service must be implemented on a case-by-case basis
  - More conservative policy in terms of protection
  - the number of choices available to the user is limited

- **Allow All**: Anything that is not specifically denied is allowed. **Ease of management**
  - Forward all traffic and each malicious service must be closed on a case-by-case basis
  - Increasing difficulties in guaranteeing security as the network grows.
  - Rarely used in security schemes, however it is may cover several specific cases

Selective traffic filtering

Service	Port	Protocol
echo	7	TCP/UDP
discard	9	TCP/UDP
systat	11	TCP/UDP
daytime	13	TCP/UDP
netstat	15	TCP
quotd	17	TCP/UDP
chargen	19	TCP/UDP
ftp-data	20	TCP
ftp	21	TCP
ssh	22	TCP/UDP
telnet	23	TCP
smtp	25	TCP
time	37	TCP/UDP
rlp	39	TCP/UDP
whois	43	TCP/UDP
tacacs	49	TCP/UDP
domain	53	TCP
whois++	63	TCP/UDP
bootp	67-68	UDP
tftp	69	UDP
gopher	70	TCP
finger	79	TCP
http	80	TCP
link	87	TCP
supdup	95	TCP
pop2	109	TCP
pop3	110	TCP
sunrpc	111	TCP/UDP
auth	113	TCP/UDP
nntp	119	TCP
ntp	123	TCP/UDP
nbios-ns	137	TCP/UDP

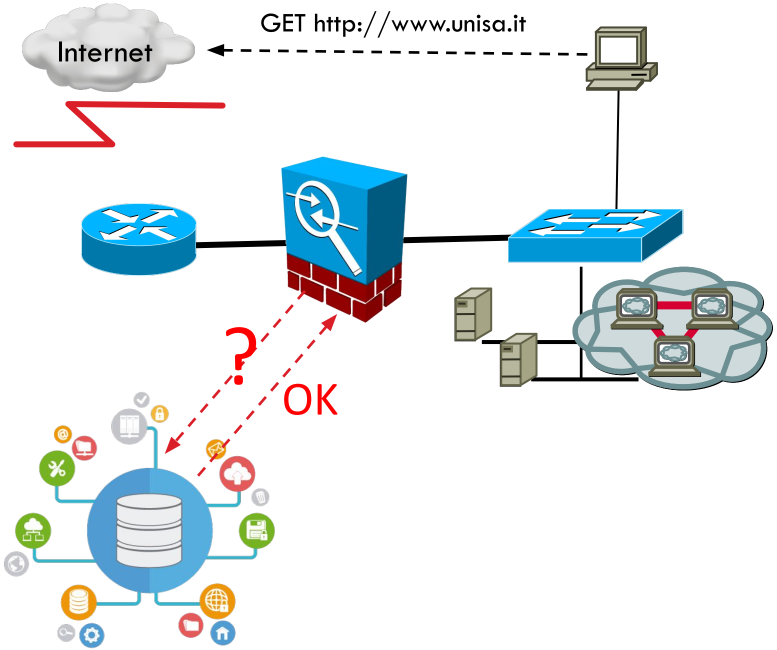




Service	Port	Protocol
nbios-dgm	138	TCP/UDP
nbios-ssn	139	TCP/UDP
imap	143	TCP
NeWS	144	TCP
snmp	161	UDP
snmptrap	162	UDP
xdmcp	177	UDP
irc	194	TCP/UDP
wais/Z39.50	210	TCP
imap3	220	TCP
ldap	389	TCP/UDP
netware-ip	396	TCP/UDP
rmt	411	TCP
https	443	TCP
exec	512	TCP
biff	512	UDP
login	513	TCP
who	513	UDP
shell	514	TCP
syslog	514	UDP
printer	515	TCP/UDP
talk/ntalk	517-518	TCP/UDP
route	520	UDP
timed	525	TCP/UDP
uucp	540-541	TCP
mountd	635	TCP/UDP
wins	1512	TCP/UDP
radius-old	1645-1646	UDP
radius	1812-1813	UDP
openwin	2000	TCP
NFS	2049	TCP/UDP
X11	6000-6063	TCP

- It can be a good practice blocking or selectively filtering potentially dangerous services
- Only allow access to an extremely limited number of services (e-mail, www, ftp) provided by specific and possibly controlled hosts

Content filtering



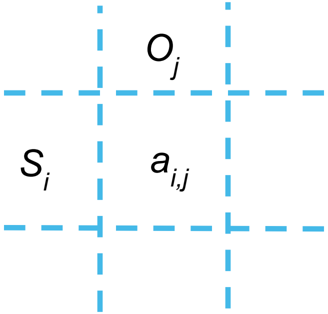
- Filtering unwanted, objectionable, and harmful content through URL inspection
- Requires the use of third party and always up-to-date knowledge bases
  - Resource classification DB
  - Categorization engines
- The firewall performs payload inspection and before admitting the session checks the content type against local policies
  - e.g. block gambling, drugs, crime-related URLs

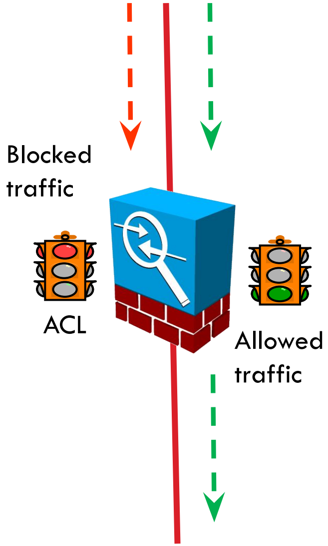
Current Topic

- Security enforcement devices
- Traffic filtering strategies and policies
- Network access control: techniques and tools
- Implementing simple network access control policies

Traffic filtering mechanisms: ACL

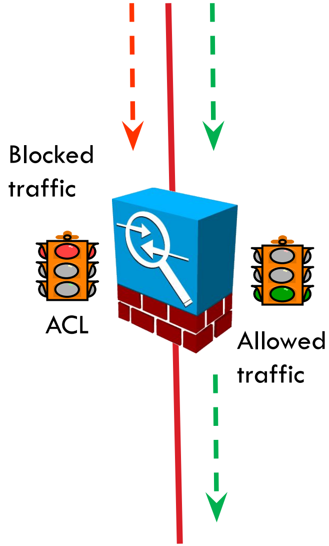
- The simplest and most immediate way to implement security schemes and policies is traffic filtering (packet filtering)
- Routers, switches and firewalls support lists of filtering (or access control) rules known as ACLs: **Access control Lists**
- An ACL represents a column in the Lampson's **access control matrix**, where:
  - S_i: j-th subject to be controlled (e.g. a netblock)
  - O_j: j-th object to be protected (e.g. an interface)
  - $a_{i,j}$: access rigths of S_i on O_j (e.g. permit or deny)





Traffic filtering mechanisms: ACL

- Filtering rules can be applied at:
  - **data link** layer (based on MAC addresses)
  - **network** layer (based on IP addresses)
  - **trasport** layer (based on ports or protocol)
- Additional elements may be checked:
  - Date and time of application
  - Session flags or status (established, closing etc.)
- Each packet received is compared with each rule, in the order in which it appears on the list, to decide if it has to be forwarded or dropped
  - The application of controls takes place on an interface basis
  - Eligible actions are **permit** (or allow) and **deny** (or drop)
  - The direction of application of the controls (**inbound** or **outbound**) is significant and defines the origin of the traffic concerned

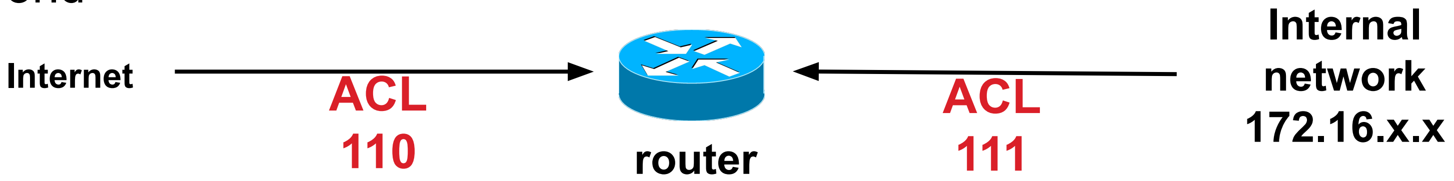


ACL Syntax

- Only one ACL can be applied to an interface in each specific direction (inbound or outbound):

```
interface ethernet 0
  ip access-group 110 in
  ip access-group 111 out
```

- In the example below, the ACL 110 and 111 are applied respectively to the input and output on the border interface that connects a router to the outside world



ACL Syntax

- An ACL is made up of rules scanned in sequence until the first match

list identifier protocol destination address/port

```
access-list 110 permit tcp any gt 1024 172.16.0.0 0.0.255.255 eq 23
```

action src addr/port

- The masks associated with the addresses are in "reverse dotted mask" format or “/msklen” in format (e.g. `0.0.0.255` is equal to `/24`)

ACL Syntax

- **Long search**: the search is carried out until there is a matching (rule with permission or denial found) or until the list is finished;
- Efficiency depends on the **order**: the most frequent matching element should be the first in the list
- Removing a permission may be without effect
- The **any** option replaces 0.0.0.0 as the IP address and 255.255.255.255 as the wildcard mask. It results in a matching with any compared address.
- The **host** option replaces 0.0.0.0 as a mask. This mask requires that all the bits of the address match. Compare exactly one address.

ACL Syntax

- Every ACL terminates with an implicit "`deny any any`" clause
- It is possible to use relational operators in ACLs: `eq neq, gt, lt:`

```
access-list 110 deny tcp 192.168.1.0 0.0.0.255 any eq www
access-list 110 deny tcp any eq ftp 192.168.1.25
```

- ACLs can be assigned logical names

```
ip access-list extended allowt permit tcp host 192.132.34.17  any eq 23
```

- It is possible to define rules in ACL that can be activated on a date/time basis, specifying a "time-range" of validity and a periodic or absolute scope

```
time-range no-http periodic weekdays 8:00 to 18:00
access-list 110 deny tcp any any eq http time-range no-http
```

ACL Syntax

- The "established" clause at the end of a rule identifies all TCP connections that have passed the setup phase (3 way handshake)

```
access-list 110 permit tcp any any established
```

  - allows you to block all incoming traffic from the outside, with the exception of return TCP traffic, due to a TCP session started from the inside.
  - checks, on incoming TCP packets, the presence of the TCP ACK or RST flags:
    - if they are present, traffic is allowed,
    - otherwise it is assumed that the traffic has been generated from the outside and will be blocked.

ACLs on Linux: iptables

- Simple ACLs can be implemented as well under linux with iptables
- Iptables is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel.
- Several different tables may be defined. Each table contains a number of built-in chains and may also contain user-defined chains.
- chain = list of rules which can match a set of packets
  - each rule specifies criteria for a packet and an associated target, namely what to do with a packet that matches the pattern
  - We are interested in the FORWARD built-in chain:
    - packets that have been routed and were not for local delivery will traverse this chain.

ACL Syntax: iptables

- It is possible a new user-defined chain by the given name

```
iptables –N acl111
```

- … and apply on specific inbound or outbound interfaces

```
iptables -A FORWARD -i eth1 -o eth0 -j acl110
```

- Also a default policy for the chain can be specified

```
iptables -P FORWARD DROP
```

- Possible targets are
  - accept = let the packet through
  - drop = drop the packet on the floor

ACL Syntax: iptables

- Syntax is very intuitive and based on traditional shell command-line

list identifier protocol destination address/port

```
iptables -A myacl -p tcp -s 172.16.1.0/24 -d 192.168.1.1 --dport domain -j ACCEPT
```

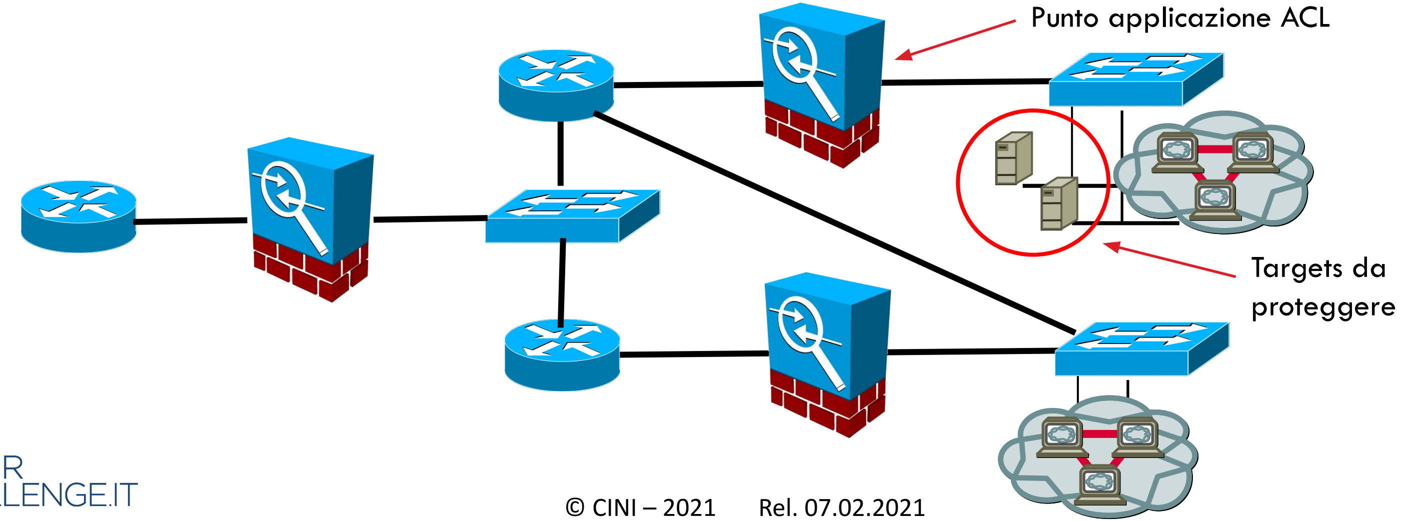
src addr/mask action

- very similar to traditional ACLs and also provides an “established” facility

```
iptables -A acl110 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Applying ACL in the correct places

- ACLs should be placed **as close as possible** to the **target** to be protected
- This allows to **restrict** the **size** of the **security domain** in order to increase the effectiveness of the filtering policies implemented and make the solution more **scalable**

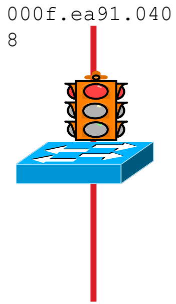


ACL applied on a switch

- Filtering can be applied also at the link layer
  - e.g. by authorizing only one host to traverse an interface

```
mac access-list mac-01          interface eth 1
  permit host 00c0.4f00.0407 any      mac port access-group mac-01 in
```

```
iptables -A FORWARD -i eth1 -m mac --mac-source 00:C0:4F:00:04:07 -j ACCEPT
```



- It can be helpful to completely lock the mac of a compromised host

```
mac-address-table static 000f.ea91.0408 vlan 1 drop
```

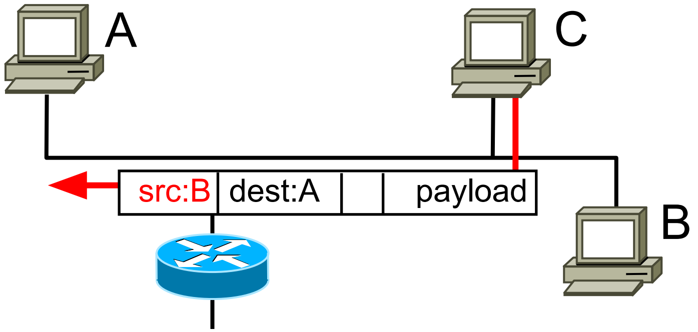
```
iptables -A FORWARD -m mac --mac-source 00:0F:EA:91:04:08 -j DROP
```

Current Topic

- Security enforcement devices
- Traffic filtering strategies and policies
- Network access control: techniques and tools
- Implementing simple network access control policies

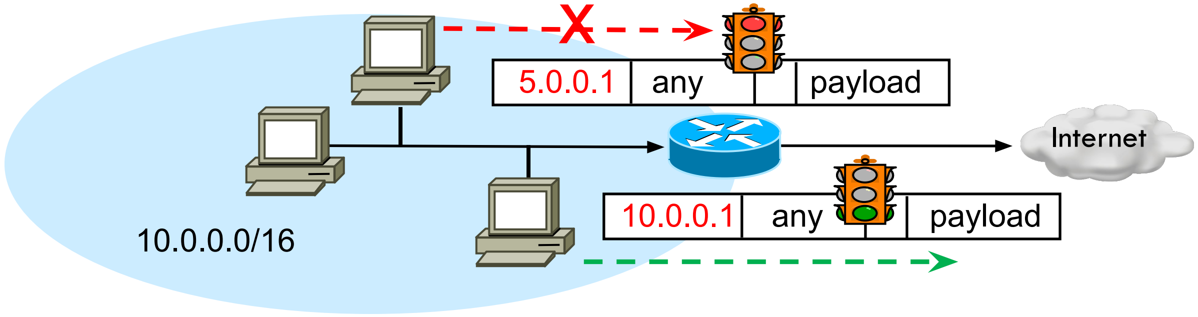
IP address spoofing

- The source IP address is currently the only mechanism for identifying the source available on the Internet
- The falsification/forgery of this data is the basis of most of the attacks and hostile actions
- Spoofing consists in falsifying the source address
  - Any user is able to generate IP packets with any value of the fields provided by the protocol structure
  - Therefore it is immediate to change the source address of the IP packets to prevent any form of identification
  - The result is that C in attacking A assumes the identity of B



Inbound anti-spoofing filtering

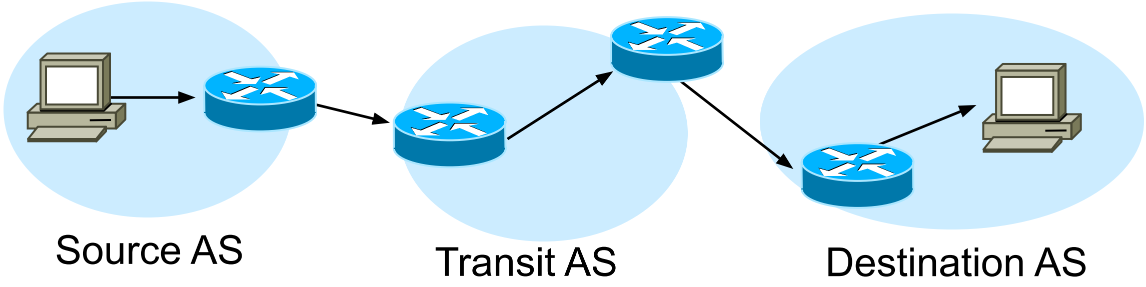
- Solution: checking and enforcing the correctness of the origin of the generated packets



- Inbound filtering policy (RFC 2827, 2000): A border router forwards only packets with legitimate source addresses

Practical implementation problems

- It is necessary that all the organizations involved and the transit ISPs do it
- Everything is based on a collaboration and trust logic working at a global level
  - If 10% of ISPs do not implement it, it is ineffective
- Another solution: enforcing / IP validation of sources at AS peering level



- A packet can only pass if the transit AS validates the source

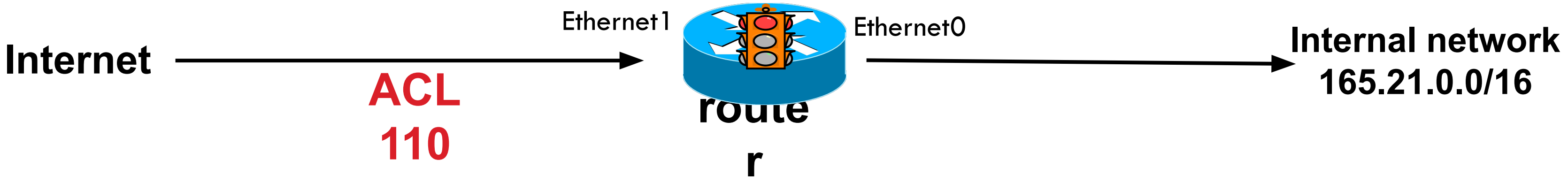
Inbound Anti-Spoofing filters

- The easiest way to protect yourself is to discard all incoming traffic with inadmissible source addresses with respect to the traffic origin

```
interface ethernet 1
ip access-group 110 in
```

```
# blocca traffico spoof entrante da eth1
iptables -A FORWARD -i eth1 …
```

- Block all traffic with source addresses 165.21.0.0/16 if coming from outside (they are my internal addresses!)



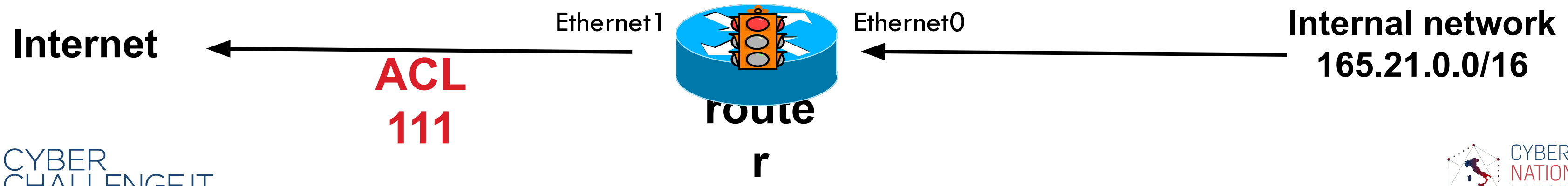
Outbound Anti-Spoofing filters

- To also prevent voluntary or involuntary spoofing from the inside of your network to the outside, similar filtering must be applied to outbound traffic

```
interface ethernet 1
ip access-group 111 out
```

```
# non inoltrare il traffico spoof da eth0
 iptables -A FORWARD -i eth0 …
```

- Block any outgoing packet with source address that does not fall on the network 165.21.0.0/16



Anti-Spoofing ACLs

Inbound Anti spoofing

! Block traffic from the outside with internal source addresses:

```
access-list 110 deny ip 165.21.0.0 0.0.255.255 any log
access-list 110 permit ip any any
```

iptables -A FORWARD -i eth1 -s 165.21.0.0 /16 -j DROP

Outbound Anti spoofing

! Block outgoing traffic with foreign source IPs:

```
access-list 111 permit ip 165.21.0.0 0.0.255.255 any
access-list 111 deny ip any any log
```

iptables -A FORWARD -i eth0 -s ! 165.21.0.0 /16 -j DROP

Docker Fundamentals

Enrico RUSSO
Università di Genova

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

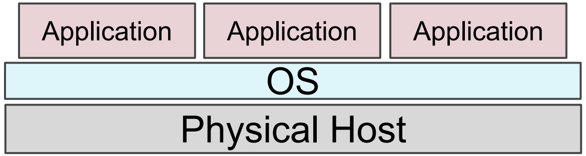
Outline

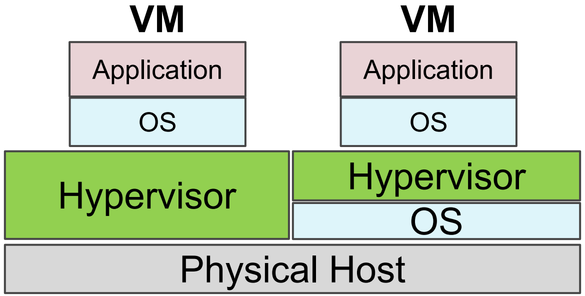
- Docker images and containers
- Docker networking
- Docker compose

Outline

- Docker images and containers
- Docker networking
- Docker compose

Traditional vs Virtualized Deployment





- Physical Hosts run an Operating System (e.g., Windows or Linux).
- Multiple applications run on the shared OS.

- A special software, i.e., the **Hypervisor**, provides Virtual Machines.
- Examples of such technologies are *Virtualbox*[1] or *Linux KVM*[2].
- VM is a full machine running all the components, including its own Operating System, on top of the virtualized hardware

[1]https://www.virtualbox.org/

[2]https://www.linux-kvm.org

Traditional vs Virtualized Deployment

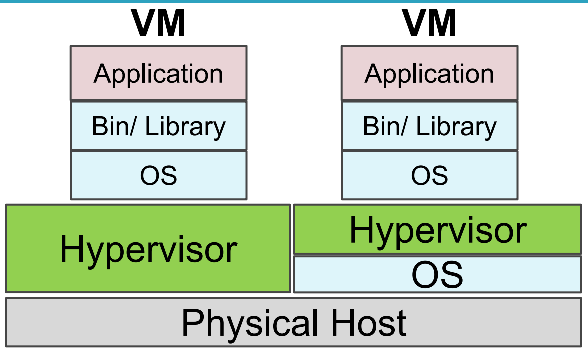
Traditional Deployment

- No way to define resource boundaries for applications.
- Isolating applications requires running them on different physical servers (expensive and resources could be underutilized).

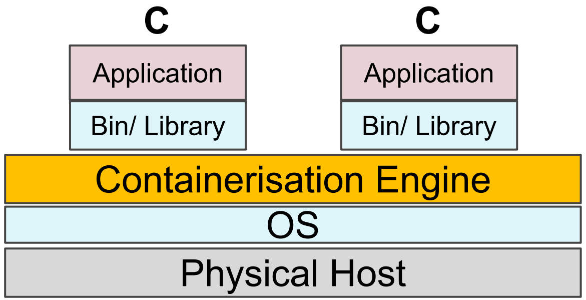
Virtualized Deployment

- Virtualization allows:
  - applications to be isolated between VMs
  - better utilization of resources in a physical server
  - better scalability.

Virtualized vs Container Deployment



- Virtual hardware
  - Each VM has an OS and Application
  - Share hardware resource from the Physical Host



- Virtual Operating Systems
  - Isolated environments, namely **containers**, sharing the same *real* operating system
  - Containers run from a distinct image that provides all files (Bin/, Library) necessary to support them
  - Examples of such technologies is *Docker*[1].

[1]https://www.docker.com/

Virtualized vs Container Deployment

Virtualized Deployment

- Heavyweight: each VM relies on a full copy of an Operating System.
- Provides full isolation.
- Best suited for when you have applications that need to run on *different* Operating System flavors.

Container Deployment

- Lightweight: sharing OS resources significantly reduces the overhead required for running containers.
- Provides a (relaxed) process-level isolation.
- Best suited for when you have applications that need to run over a *single* Operating System kernel.

Docker images and containers

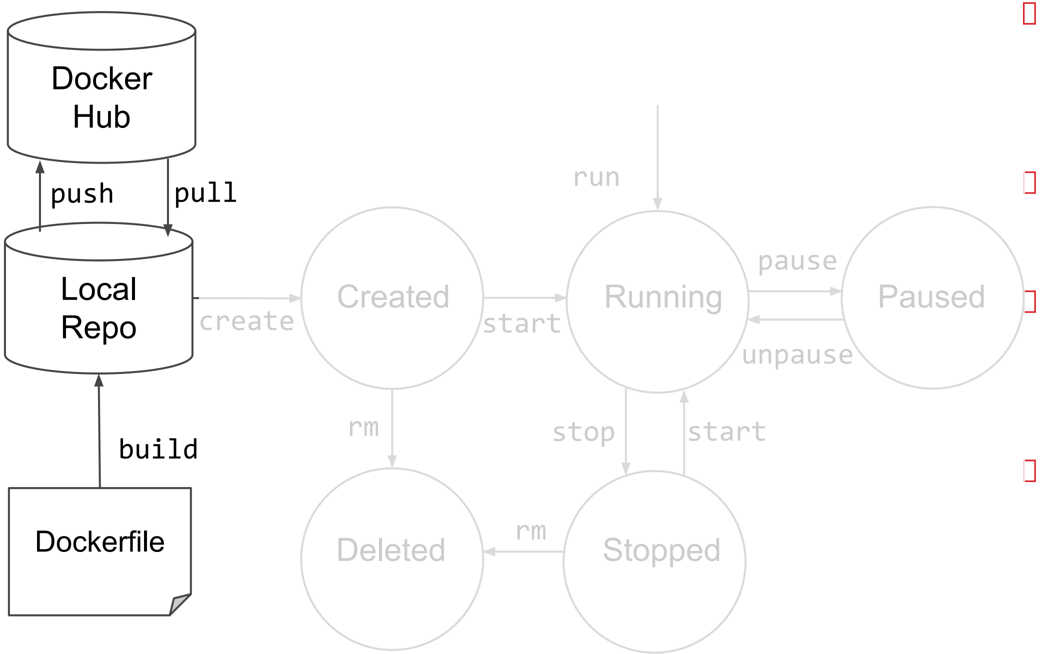
Image

- **Immutable** template for containers
- Includes everything needed to run an application
  - code, runtime, system tools, system libraries, and settings

Container

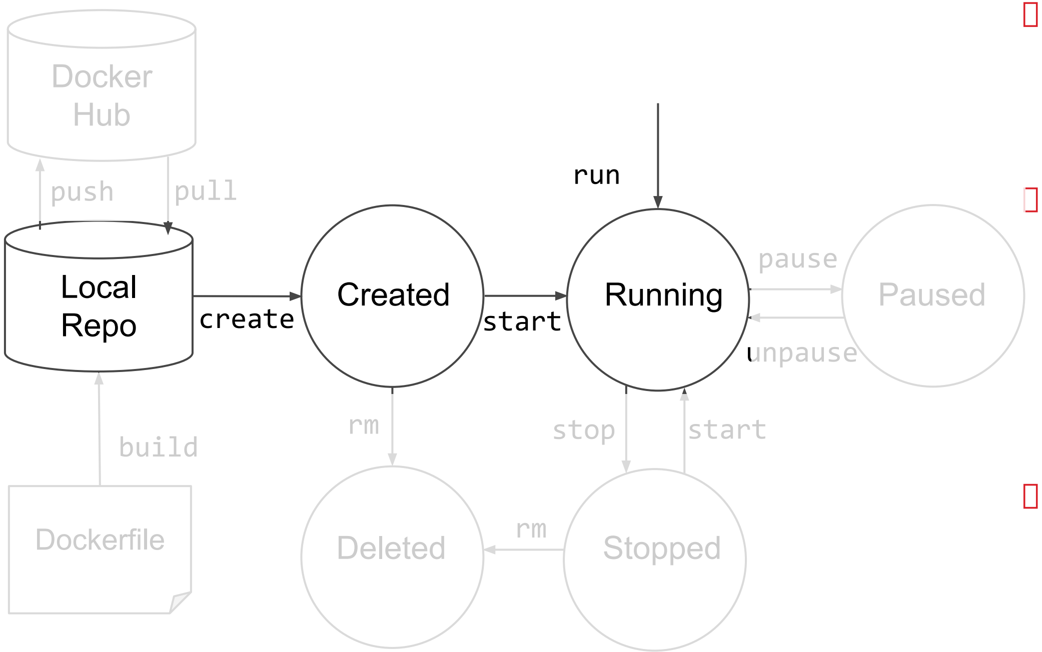
- An instance of an image
- Add a **new writable layer** on top of the underlying image
  - all changes made to the running container (e.g., writing new files or modifying existing files) are written to this writable container layer

Docker container lifecycle



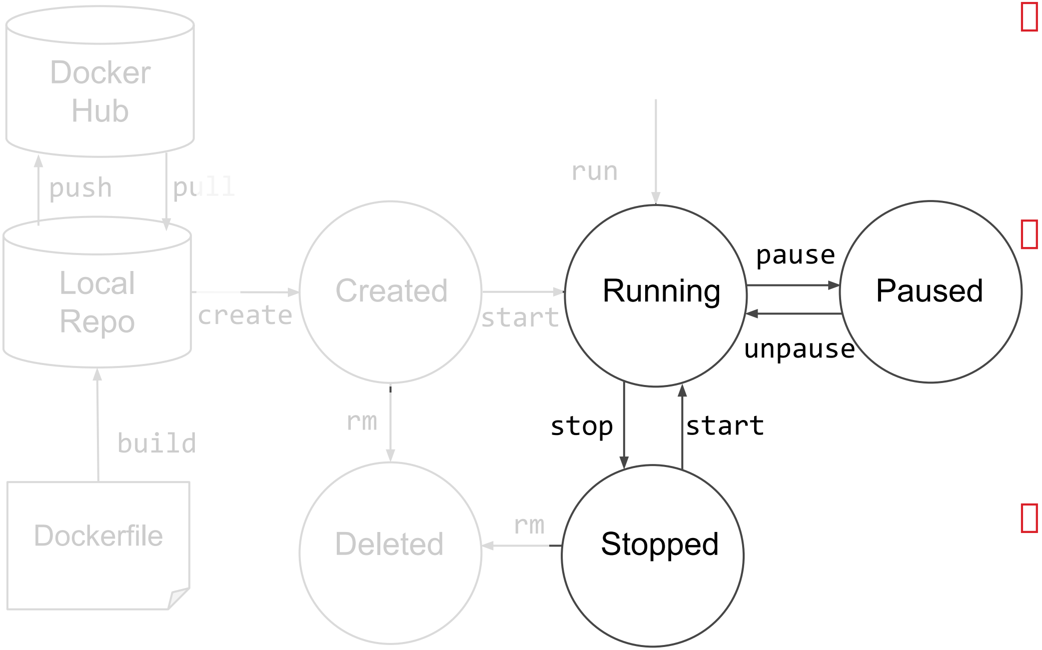
- Docker images can be pulled from a central repository (Docker Hub)
- Pulled images are saved in a local repository
- Custom images can be created and saved starting from a specific configuration file (Dockerfile)
- Custom images can be pulled to the central repository

Docker container lifecycle



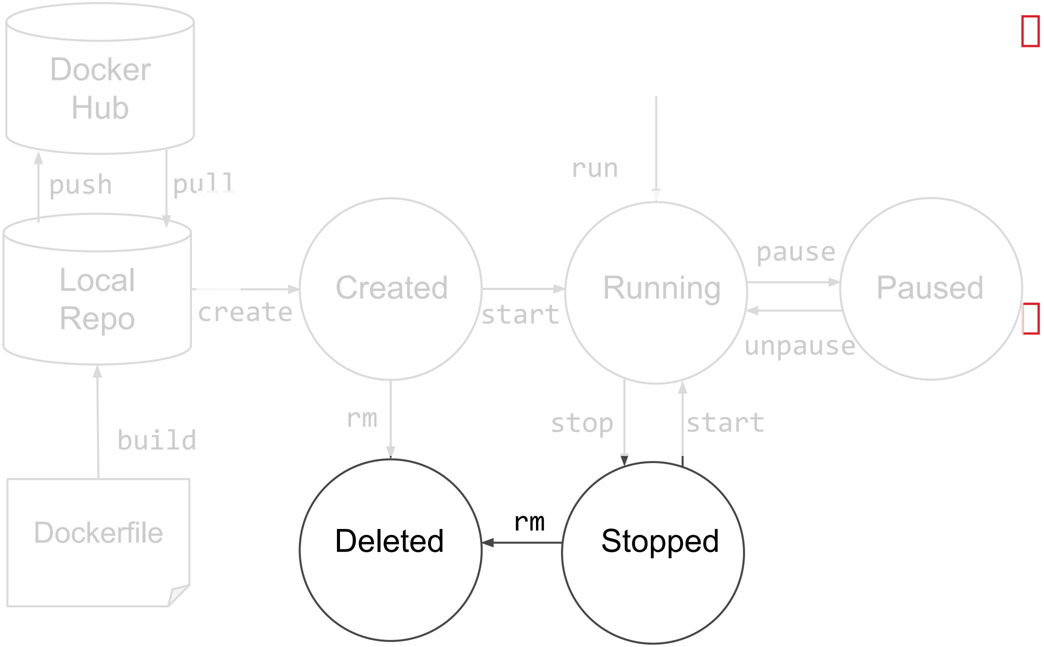
- A saved image can be used for creating a container (a writeable layer is added)
- Starting a container means running a default command contained in the image, namely the entrypoint (can be overridden)
- A container can be created and started using a single run command

Docker container lifecycle



- After executing the entrypoint, the container stops
- A **foreground** process specified as the entrypoint can keep running the container
- A running container can be paused or stopped

Docker container lifecycle



- After a container is stopped, the writeable layer still exists
- Deleting a container permanently removes the associated writable layer

Docker lifecycle example: images

- Pull an image from the central hub
  - docker pull <image>
- Build an image from a Dockerfile
  - docker build -t <image>
- List images saved in the local repository
  - docker images
- Delete an image
  - docker rmi <image>

Docker lifecycle example: containers

- Start a container
  - docker run <image>
- Start a container overriding default entrypoint with <newcmd>
  - docker run --entrypoint <newcmd> <image>
- Execute a command <cmd> (e.g., /bin/bash) in a running container
  - docker exec -it <containerID> <cmd>
- Stop a container
  - docker stop <containerID>
- Remove a container
  - docker rm <containerID>
- List running and stopped containers
  - docker ps –a

Docker volumes

- Volumes can be used to save (persist) data and to share data between containers
- Volume is unrelated to the container layers: deleting a container does not involve deleting an associated volume
- A volume can be:
  - (anonymous/)named: managed internally by Docker itself
  - host: refers to a filesystem location of the host running Docker

Docker volumes: example

- Create a named volume
  - docker volume create volumename
- Run a container using the named volume
  - docker run –v volumename:/path/in/container_filesystem
- Running a container using a host volume
  - docker run –v /path/on/host_filesystem:/path/in/container_filesystem

Dockerfile

- Docker can build custom images automatically by reading the instructions from a Dockerfile
- Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image
- The *docker build* allows the execution of an automated build of an image starting from a Dockerfile

(Dockerfile reference: https://docs.docker.com/engine/reference/builder/)

Dockerfile example

Command	Description	Example
FROM <image>	Start building from this (base) image	FROM ubuntu
RUN <cmd>	Run the specified command	RUN apt install apache2
COPY <src> <dest>	Copy a file to the image fs	COPY vh.conf /etc/apache2/conf/
CMD ["exec", "param1“, …]	Configure the default command when container starts	CMD ["apache2", "-D", " FOREGROUND "]

Outline

- Docker images and containers
- Docker networking
- Docker compose

Docker networking

Docker supports different configurations, the two main ones being

- *Bridge* (default)
  - isolated layer 3 networks enabling connected containers to communicate
  - (can) allow the access to external networks masquerading connections with the host network configuration
- *Host*
  - containers use the host network
  - listening ports are exposed to the outside world

Docker networking: published ports

- A container connected to bridges is isolated and does not expose any of its ports to the outside world
- A published port can be made available to services running outside the container
- A published port is mapped to a port on the Docker host

Docker networking: examples

- Create network with a configured subnet and gateway address
  - docker network create --driver bridge <networkname> --subnet=<ip/mask> --gateway=<ip/mask>
- Connect a container to a network
  - docker network connect <networkname> <containerid>
- Run a container and expose a port
  - docker run –h <host-name> -p <internal-port>:<exposed-port> --name <container-name> <image-name>

Outline

- Docker images and containers
- Docker networking
- Docker compose

Docker compose

Compose is a tool for defining and running multi-container Docker applications

- A single file for providing configurations
- A single set of commands for configuring, building, and running all the containers

Docker compose configuration

- The Compose file (docker-compose.yaml) uses a standard, human-readable syntax, namely YAML* syntax. It defines
  - Services: configuration that is applied to each container (much like passing command-line parameters to *docker run*)
  - Networks (optional): define configuration of networks to be created
  - Volumes (optional): define configuration of volumes to be created

* https://yaml.org/

Docker compose configuration: example


```yaml
services:
 # frontend container
 app:
  # use a custom image
  build:
   # directory containing Dockerfile
   context: ./app
  # image name
  image: custom_image
  # exposed ports (host:container)
  ports:
    - 8080:80
  # a mapped host volume
  volumes:
   -  ./config/config.json:/etc/config.json
  # connected networks (defined in networks..)
  networks:
   ext:
    ipv4_address: 192.168.100.100
   int:
 # backend container
 db:
  # pull an existing image
  image: mariadb
  # a mapped named volume
  volumes:
   - db-content:/var/lib/mysql
  networks:
   int:

volumes:
 db-content:

networks:
 ext:
  driver: bridge
  ipam:
   driver: default
   config:
     - subnet: 192.168.100.0/24
 int:
  driver: bridge
```

Docker compose: commands example

- (build images and) run services
  - docker-compose up –d
- stop services
  - docker-compose stop
- start services
  - docker-compose start -d
- stop and remove containers and networks
  - docker-compose down
- show logs (entrypoint output)
  - docker-compose logs -f [service_name]