LINUX NETWORK TOOLS

Let's see some real traffic…

tcpdump

wireshark

`tcpdump`
the command line network analizer

For documentation:
- `man tcpdump` (program usage)
- http://danielmiessler.com/study/tcpdump/ (nice tutorial)

Essentials:
- Capture all packets on all interfaces and dump the entire packet:
  ```
  tcpdump –i any -X
  ```
- Capture all packets on all interfaces and don't convert addresses to names:
  ```
  tcpdump –i any -n
  ```
- Capture all packets on eth0 and save the trace on file (the whole packets…):
  ```
  tcpdump –i eth0 –w file –s0
  ```
- Capture 10 packets on eth0 to/from `$ADDR`:
  ```
  tcpdump –i eth0 –c 10 host $ADDR
  ```
- Capture all TCP packets to/from port 80 on eth0:
  ```
  tcpdump –i eth0 tcp port 80
  ```
- Capture all packets with destination or source address != $ADDR and port in the range [10000:20000]:
  ```
  tcpdump –i eth0 host not $ADDR portrange 10000-20000
  ```

`tcpdump`
output format

Normal output

```
0 packets dropped by kernel
root@marlon-vmxbn:/home/marlon/Src/netgroup# tcpdump -ni eth0 host 8.8.8.8
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:50:44.913843 IP 172.16.166.152 > 8.8.8.8: ICMP echo request, id 25220, seq 1, length 64
15:50:44.936668 IP 8.8.8.8 > 172.16.166.152: ICMP echo reply, id 25220, seq 1, length 64
```

Verbose output

```
0 packets dropped by kernel
root@marlon-vmxbn:/home/marlon/Src/netgroup# tcpdump -nvvvi eth0 host 8.8.8.8
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:51:05.529625 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length 84)
    172.16.166.152 > 8.8.8.8: ICMP echo request, id 25250, seq 1, length 64
15:51:05.554011 IP (tos 0x0, ttl 128, id 745, offset 0, flags [none], proto ICMP (1), length 84)
    8.8.8.8 > 172.16.166.152: ICMP echo reply, id 25250, seq 1, length 64
```
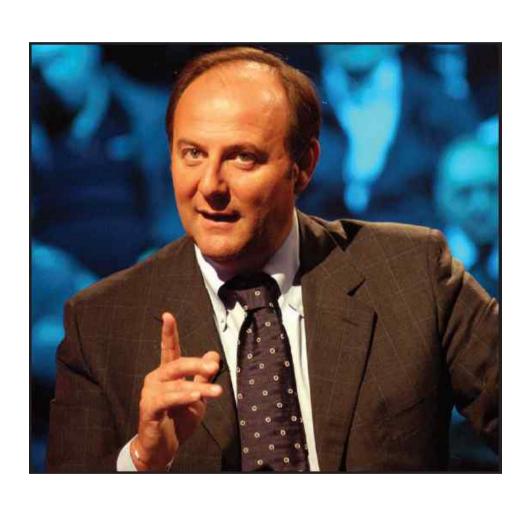
`tcpdump`
output format

Packet content in HEX and ASCII

```
root@marlon-vmxbn:/home/marlon/Src/netgroup# tcpdump -nXi eth0 host 8.8.8.8
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
15:51:28.311102 IP 172.16.166.152 > 8.8.8.8: ICMP echo request, id 25254, seq 1, length 64
        0x0000:  4500 0054 0000 4000 4001 d7f0 ac10 a698  E..T..@.@.......
        0x0010:  0808 0808 0800 57ba 62a6 0001 f08c 4f4f  ......W.b.....OO
        0x0020:  0ebf 0400 0809 0a0b 0c0d 0e0f 1011 1213  ................
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
15:51:28.335982 IP 8.8.8.8 > 172.16.166.152: ICMP echo reply, id 25254, seq 1, length 64
        0x0000:  4500 0054 02eb 0000 8001 d505 0808 0808  E..T............
        0x0010:  ac10 a698 0000 5fba 62a6 0001 f08c 4f4f  ......_.b.....OO
        0x0020:  0ebf 0400 0809 0a0b 0c0d 0e0f 1011 1213  ................
        0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223  .............!"#
        0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233  $%&'()*+,-./0123
        0x0050:  3435 3637                                4567
```

tcpdump advanced filtering

- `man pcap-filter` (filter syntax details)
- pcap filter primitives include
  - `host, dst host, src host`
  - `port, dst port, src port`
  - `ether host, ether dst, ether src`
  - `net, dst net, src net`
  - `portrange, dst portrange, src portrange`
  - `less, greater`
  - `ip proto, ip6 proto, ether proto`
  - `ip broadcast, ip multicast`
  - `ip, ip6, arp, tcp, udp, icmp`
  - `ifname`
  - `proto [ expr : size ]`
    - `ip[16:4] = 0xffffffff` → DEST BROADCAST IP PACKET

- Example:
  - `tcpdump -ni any "ip[12:4] = 0xac10a69c"`

NERD QUIZ

What do they mean?

(1) ether[0] & 1 != 0
(2) ip[0] & 0xf != 5
(3) ip[6:2] & 0x1fff = 0

Are you sure?
Shall we light them?

Solutions


```
ether[0] & 1 != 0          (ethernet multicast/broadcast packet)
ip[0] & 0xf != 5           (ip packets with option)
ip[6:2]  & 0x1fff = 0  (ip un-fragmented packets or frag0)
```

Wireshark

- Wireshark is a graphical packet analyzer
- Like `tcpdump` can analyze live streams or files
- It's compatible with `tcpdump` (pcap format) traces
- It provides additional features:
  - Better protocol parsing
  - Statistics tool
  - Exporting
  - Better Filtering (different syntax)
  - Can be extended to understand proprietary protocol

Wireshark

Wireshark

Trace export

Filtering

Protocol hierarchy

I/O graphs

Wireshark and NETKIT

- Can I use wireshark to capture traffic on a NETKIT VM?
  - No! But I can use wireshark to open a trace captured with `tcpdump`
  - It's only a matter of copying the file from the VM to the HOST machine (let's use the hosthome directory)
  - Second option: copy the file with `nc, scp or rsync` (later on…)

ping

- `ping` is one of the oldest IP utilities around
- `ping` asks another host if it is alive, and records the round-trip time between the request and the reply
- `ping` relies on ICMP echo-request and echo-reply packets (next slide..)
- **warning**: in some cases ICMP traffic is dropped by firewalls. We can not assume that all machines are down if they don't reply to a ping…

ICMP basics

- The **Internet Control Message Protocol** is one of the core protocols of the IP Suite
- ICMP packets are mainly used for diagnostic (ping, traceroute, timestamp request) and error notification (routing anomalies, unreachability, TTL expired, etc…)
- It goes directly on top of IP (but it can't be seen as a transport protocol)
  - IP.proto = 1
- We will focus on ICMP Echo Request/Reply. We'll see (and force the transmission) of other ICMP messages later on…

ICMP header

ICMP Message Types

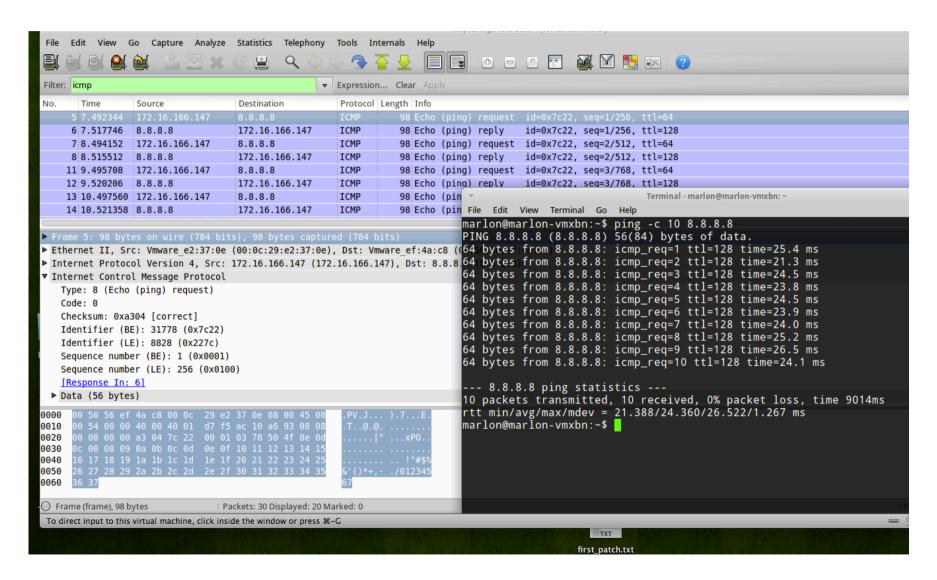| Type	Code/Name	Type	Code/Name	Type	Code/Name
0	Echo Reply	3	Destination Unreachable (continued)	11	Time Exceded
3	Destination Unreachable		12 Host Unreachable for TOS		0 TTL Exceeded
	0 Net Unreachable		13 Communication Administratively Prohibited		1 Fragment Reassembly Time Exceeded
	1 Host Unreachable	4	Source Quench	12	Parameter Problem
	2 Protocol Unreachable	5	Redirect		0 Pointer Problem
	3 Port Unreachable		0 Redirect Datagram for the Network		1 Missing a Required Operand
	4 Fragmentation required, and DF set		1 Redirect Datagram for the Host		2 Bad Length
	5 Source Route Failed		2 Redirect Datagram for the TOS & Network	13	Timestamp
	6 Destination Network Unknown		3 Redirect Datagram for the TOS & Host	14	Timestamp Reply
	7 Destination Host Unknown	8	Echo	15	Information Request
	8 Source Host Isolated	9	Router Advertisement	16	Information Reply
	9 Network Administratively Prohibited	10	Router Selection	17	Address Mask Request
	10 Host Administratively Prohibited			18	Address Mask Reply
	11 Network Unreachable for TOS			30	Traceroute

Checksum

Checksum of ICMP header

RFC 792

Please refer to RFC 792 for the Internet Control Message protocol (ICMP) specification.

source: http://nmap.org/book/tcpip-ref.html

`ping` output and ICMP packets

ping usage

- For a complete doc: `man ping`
- Essentials
  - Don't convert IP addresses to names (`-n`)

    `ping -n 160.80.103.147`
  - Specify the number of packets (`-c`) and display only the summary line (`-q`)

    `ping -q -c 10 160.80.103.147`
  - Specify the source address of the packets (`-I`)

    `ping —I 10.0.0.12 160.80.103.147`
  - Stress the network (flood `-f`) and specify the size of the packet (`-s`)

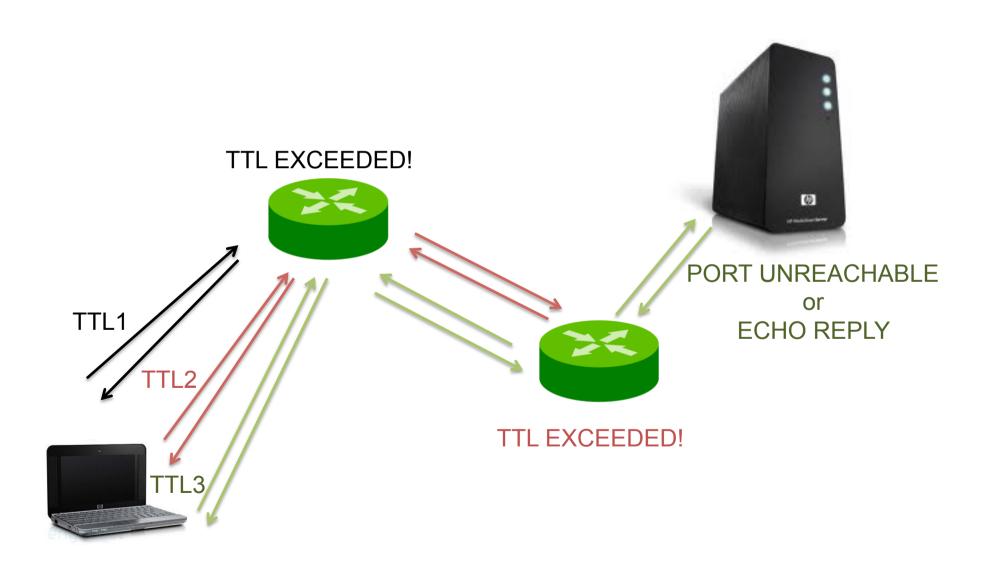   `ping -c 5000 -s 512 -f 160.80.103.14`
  - Record the network route (many hosts ignore the ROUTE RECORD option. Let's use `traceroute` for that)

    `ping -R 160.80.103.14`

`traceroute`

- A computer network diagnostic tool for displaying the route and measuring transit delays of packets across an IP network
- `traceroute` sends a sequence of packets to the destination
- `traceroute` works by increasing the TTL value of each successive (set of) packet(s)
- `traceroute` reconstructs the path to the destination by receiving the ICMP TTL Exceeded message by each router traversed by the packet
- Implementations on Unix-like OSs use UDP with ports from 33434 to 33534. Others use ICMP Echo Request
- For UDP version, `traceroute` ends when a port unreachable is received from the destination
- For ICMP version, `traceroute` ends when a ICMP Echo Reply is received for the destination

How does `traceroute` work?

TTL EXCEEDED!

TTL1

TTL2

TTL3

TTL EXCEEDED!

PORT UNREACHABLE
or
ECHO REPLY

traceroute

RTT

```
marlon@MarlonMAC:~$ traceroute -q 1 -v 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 64 hops max, 52 byte packets
 1   192.168.100.1 (192.168.100.1) 36 bytes to 192.168.100.63  6.710 ms
 2   10.192.0.1 (10.192.0.1) 36 bytes to 192.168.100.63  6.519 ms
 3   10.0.253.45 (10.0.253.45) 36 bytes to 192.168.100.63  5.579 ms
 4   10.0.253.30 (10.0.253.30) 36 bytes to 192.168.100.63  4.812 ms
 5   *
 6   rt-rm2-rt-mi2.mi2.garr.net (193.206.134.229) 36 bytes to 192.168.100.63  14.180 ms
 7   193.206.129.134 (193.206.129.134) 36 bytes to 192.168.100.63  11.496 ms
 8   216.239.47.128 (216.239.47.128) 36 bytes to 192.168.100.63  12.231 ms
 9   72.14.232.78 (72.14.232.78) 148 bytes to 192.168.100.63  22.366 ms
10   209.85.254.112 (209.85.254.112) 36 bytes to 192.168.100.63  21.627 ms
11   *
12   google-public-dns-a.google.com (8.8.8.8) 36 bytes to 192.168.100.63  24.035 ms
marlon@MarlonMAC:~$
```

Basis usage:
```
traceroute [options] $DEST_HOST
```

Useful options:
```
-q <num_queries>: number of queries
-i <iface_name>: source interface
-s <addr>: source address
-M <ttl>: initial TTL
-m <ttl>: maximum TTL
-w <time>: wait time for a probe response
```
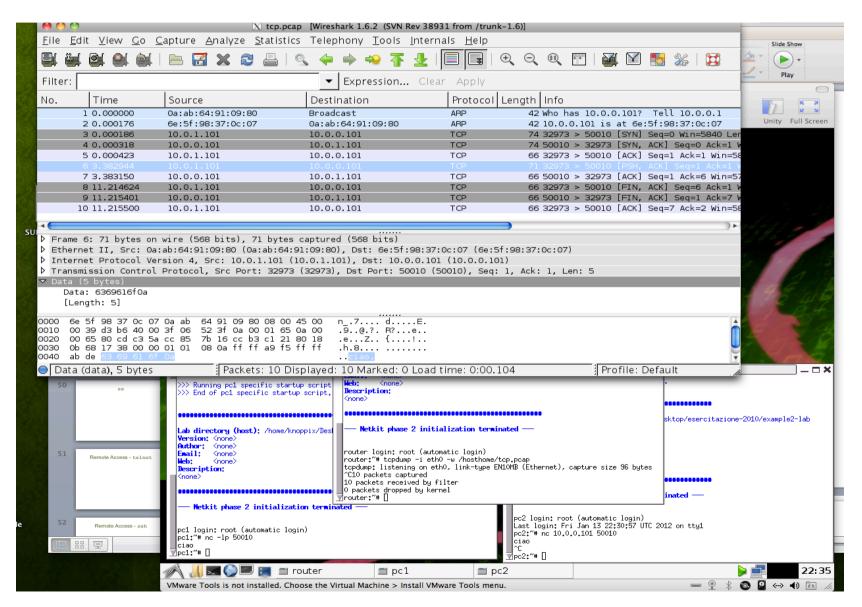
netcat

- Utility that reads and writes data through IP transport session, either TCP or UDP
- It can create TCP or UDP socket in listening
  - `nc –l 9000` (open a TCP socket listening on port 9000)
  - `nc –lu 9000` (open a UDP socket listening on port 9000)
- It can connect a TCP socket
  - `nc 160.80.103.147 9000`
- It can create a UDP socket for sending packets
  - `nc –u 160.80.103.147 9000`

- **NOTE:** there are 2 versions of nc. One is the GNU version. The other one is the BSD porting. These 2 versions have a slightly different syntax and options. For example (that's the case of `nc` on the NETKIT VM), you might have to use the following syntax for listening sockets:
  - `# nc –l –p 9000`

Exercise: TCP connection

- Let's get back to Lab0
- On PC1 create a listening TCP socket on port 9999
- On PC2 connect a TCP socket to PC1:9999
- Write something and press CTRL+C to close
- Sniff the entire TCP flow on router (connection, data, close – use `tcpdump` and write to a file)
- Display the trace with wireshark

Exercise: TCP connection

Advanced use of `netcat`

- We already saw `nc` as a chat ☺

- We can also transfer files:
  - `server:# nc -l 9000 > received_file`
  - `client:# cat file_to_send | nc $server 9000`

- Get a web page (like `wget`)
  - `client:# printf "GET / HTTP/1.0\\r\\n\\r\\n\n" | nc 160.80.103.147 80`

- Remote shell (dangerous – removed from bsd porting)
  - `server:# nc -l 9000 -e /bin/bash`
  - `client:# nc $server 9000`

- Perform a port scan (`-z` option)
  - `client# nc -v -z $target 7-1023`

ss

- Utility to investigate sockets
- All TCP sockets, all UDP sockets, all established ssh / ftp / http / https connections, all local processes connected to X server, etc…
- Basic usage: # `ss [options] [filter]`

```
-s: display summary
-a: display both listening and non-listening
-l: display listening socket
-t: display TCP sockets
-u: display UDP sockets
-p: display processes using sockets

And many more…
```

- Documentation: `/usr/share/doc/iproute-doc/ss.html`

ss output

```
marlon@marlon-vmxbn:~$ ss -ap
State      Recv-Q Send-Q      Local Address:Port          Peer Address:Port
LISTEN     0      4           127.0.0.1:5037                         *:*
LISTEN     0      128                 *:www                          *:*
LISTEN     0      128                 *:webmin                       *:*
LISTEN     0      5              :::domain                        :::*
LISTEN     0      5               *:domain                         *:*
LISTEN     0      128                 *:ssh                          *:*
LISTEN     0      128              :::ssh                         :::*
LISTEN     0      128         127.0.0.1:ipp                          *:*
LISTEN     0      128             ::1:ipp                         :::*
ESTAB      200    0           172.16.166.147:33756          172.16.166.1:netbios-s
sn   users:(("gvfsd-smb-brows",29392,9))
ESTAB      0      0           172.16.166.147:43615          199.7.59.72:www
users:(("chromium-browse",2235,82))
ESTAB      0      0           172.16.166.147:42840          173.194.35.51:https
users:(("chromium-browse",2235,71))
```

```
marlon@marlon-vmxbn:~$ ss -s
Total: 512 (kernel 0)
TCP:    16 (estab 6, closed 1, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total      IP         IPv6
*         0          -          -
RAW       0          0          0
UDP       9          6          3
TCP       15         12         3
INET      24         18         6
FRAG      0          0          0
```

Remote access - `telnet`

- Telnet protocol provides a fairly general, bi-directional, eight-bit byte oriented communications facility
- A telnet connection is a Transmission Control Protocol (TCP – listening port 23) connection used to transmit data with interspersed telnet control information
  - Data: 1st bit 0 (ASCII character)
  - Commands: 1st bit 1
- Nice article describing the protocol:
  - http://support.microsoft.com/kb/231866
- Typical use: remote shell
- Example PCAP trace:
  - http://stud.netgroup.uniroma2.it/cgrl/traces/telnet.pcap
- Client/server implementation for virtually all OSs!
  - On linux: `telnet/telnetd`
  - daemon usually not installed `(apt-get install telnetd)`
- Due to several security aspects it has been "abandoned" in favor of SSH

Remote Access - SSH

- Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network

- RFCs define 3 major components:
  - The Transport Layer Protocol (RFC4252)
  - The User Authentication Protocol (RFC4253)
  - The Connection Protocol (RFC4254)

- OpenSSH (client/server implementation):
  - Encryption, Authentication, Data integrity
  - Secure file transfer (`scp`)
  - X session forwarding
  - Port forwarding
  - SOCKS4|5 proxy
  - Public Key authentication

- We won't take a look at the protocol, but we'll focus on some practical uses

OpenSSH installation and configuration (DEBIAN)

- `openssh-client` present in almost all Linux distribution (DEBIAN included)
- `openssh-server` usually not included
  - `apt-get install openssh-server`
- Configuration file in:
  - `Server: /etc/ssh/sshd_config`
  - `Client: /etc/ssh/ssh_config`
- Documentation:
  - `man (ssh_config|sshd_config)`
- Useful configuration parameters (server, except ServerAliveInterval):
  - `Protocol (1|2)`
  - `PermitRootLogin (yrs|no)`
  - `PasswordAuthentication (yes|no)`
  - `X11Forwarding` (yes|no)
  - `ServerAliveInterval` <seconds>
  - `DenyUsers` <user list> and `DenyGroups` <group list>
  - `UseDNS no`

- Remember to restart ssh to apply any changes in the configuration file
  - `/etc/init.d/ssh restart`

OpenSSH basic usage

To connect to a ssh server just type

`ssh user@server`

- The server send it's public key fingerprint
- The program asks you to verify the authenticity of the key
- Once the host is recognized, the server address is put in the file `~/.ssh/known_host`
  - To check fingerprints on server: `ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key`
- What if the key fingerprint doesn't match the one stored in `~/.ssh/known_host`? See the next slide…

SSH key authentication failure


```
marlon@MarlonMAC:~$ ssh 172.16.166.147
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@     WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
9e:32:f0:94:09:84:6e:d9:6c:dd:01:f5:33:bb:82:88.
Please contact your system administrator.
Add correct host key in /Users/marlon/.ssh/known_hosts to get rid of this message.
Offending key in /Users/marlon/.ssh/known_hosts:3
RSA host key for 172.16.166.147 has changed and you have requested strict checking.
Host key verification failed.
marlon@MarlonMAC:~$
```

Not necessarily something nasty is happening!
E.g.: ssh has been reinstalled or a big update has request the generation of a new key (pair)

SSH public key authentication

- It might happen that a sysadmin doesn't trust the strength of a user password
- Users' account violation can lead to apocalyptic scenarios (sudoers users…)
- Public key authentication is a stronger auth method
- Users are requested to generate a public/private key
- The public key is manually (and over a secure channel) installed on the server
- The user is not authenticated via user/password verification, but via a "safer" cryptographically challenge/response mechanism (later on…)

Public key authentication with OpenSSH

```
pippo@marlon-vmxbn:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/pippo/.ssh/id_rsa):
Created directory '/home/pippo/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/pippo/.ssh/id_rsa.
Your public key has been saved in /home/pippo/.ssh/id_rsa.pub.
The key fingerprint is:
3c:55:18:b3:fb:ce:b2:c2:c0:a9:4a:f9:9a:07:c8:63 pippo@marlon-vmxbn
The key's randomart image is:
+--[ RSA 2048]----+
|           oo.   |
|          .+     |
|          o      |
|        . . .    |
|..    . .S .     |
|.E.. +  . .      |
|. +. . o    .    |
| . oo  o .o      |
| ++.    ..oo     |
+-----------------+
```

Public key authentication with OpenSSH

- The client generates the key pair

  ```
  ssh-keygen -t (rsa|dsa)
  ```
- By default, the public key is stored in:

  ```
  ~/.ssh/id_rsa.pub
            or
  ~/.ssh/id_dsa.pub
  ```
- The public key has to be appended to the file `~/.ssh/authorized_keys` in the home of the authorized user
- 1st way, assuming that `id_rsa.pub` has been securely copied on the remote machine

  ```
  cat id_rsa.pub >> ~/.ssh/authorized_keys
  ```
- 2nd way, with a tool provided by OpenSSH (from the client)

  ```
  ssh-copy-id user@server
  ```

Exercise

- Back to Lab0-interfaces
- Install SSH server on router (if needed)
- Force public key authentication
- Configure public key authentication for user@router

Secure file transfer over SSH

- Basic usage
  - `scp [-r] [[user@]host1:]file1 ... [[user@]host2:]file2`
- Examples
  ```
  1) scp file1 marlon@example.org:
  2) scp marlon@example.org:file2 /home/marlon/dir/
  3) scp —r dir/ marlon@example.org:/home/marlon/dir_target
  ```
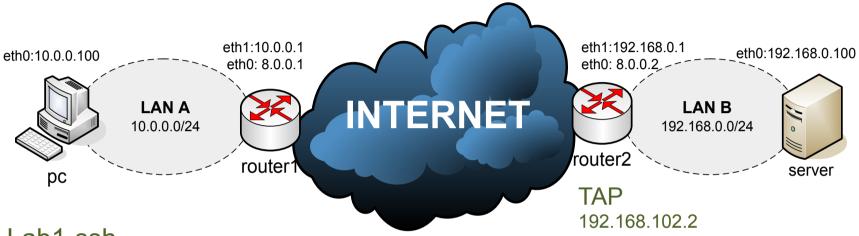
  Where:
  1) `file1` is copied in marlon's home on the remote host
  2) `file2` (in marlon's remote home) is copied in the specified local path with the same name
  3) The local directory `dir` is recursively copied into the specified remote path

OpenSSH advanced usage

- Running commands over ssh
  - `ssh username@server "command"`
- Forward X session
  - `ssh -X username@server`
- Local Port forward
  - `ssh -L lport:remote_addr:rport username@server`
- Remote port forward
  - `ssh -R rport:local_addr:lport username@server`
- Socks5 proxy
  - `ssh -ND 9999 username@server`
- Remote filesystem with sshfs
  - `sshfs  user@host: mountpoint`

- Nice tutorials:
  - http://www.subhashdasyam.com/2011/05/25-best-ssh-commands-tricks.html

Local Port Forwarding example

eth0:10.0.0.100

eth1:10.0.0.1
eth0: 8.0.0.1

LAN A
10.0.0.0/24

INTERNET

eth1:192.168.0.1
eth0: 8.0.0.2

LAN B
192.168.0.0/24

eth0:192.168.0.100

pc

router1

router2

server

TAP
192.168.102.2

Lab1-ssh

Problem: router1 doesn't have the route to 192.168.0.0/24 (as in real world topologies…)

(Note: router1 and router2 on the same lan is not a real topology… let's pretend they reach each other through the internet…)

Goal: connect pc to server:2024 with nc trough a "SSH tunnel"

Preliminaries:

Install openssh-server on router2 (if not already installed)

Create a guest account (user) for ssh login on router2 (set the password for "user" account)

To reach server from pc:

1)   Put server:2024 in listening on port 2024
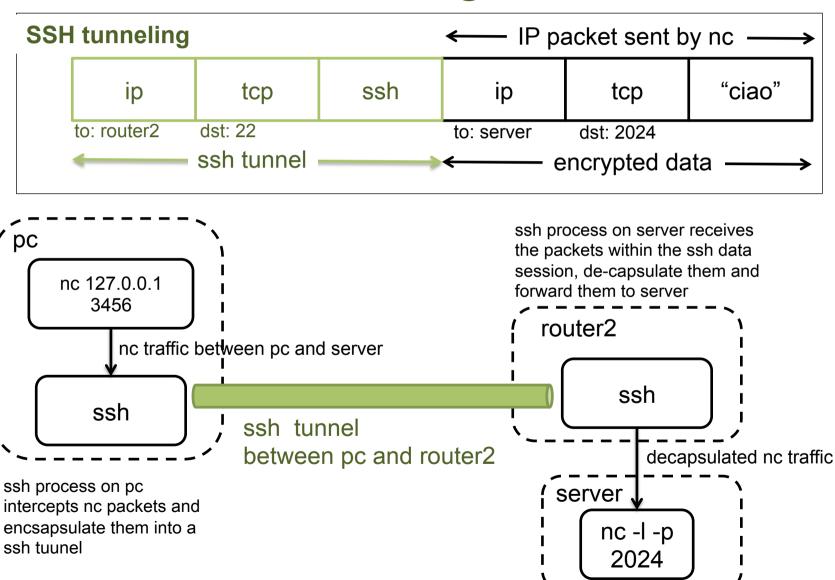
```
server# nc -l -p 2024
```

2)   Run ssh port forwarding command on pc

```
pc# ssh -NL 3456:192.168.0.100:2024 user@8.0.0.2
```

3)   Connect nc to server

```
pc# nc 127.0.0.1 3456
```

Local Port Forwarding: how it works

SSH tunneling

IP packet sent by nc →

ip	tcp	ssh	ip	tcp	"ciao"

to: router2   dst: 22   to: server   dst: 2024

← ssh tunnel →   ← encrypted data →

pc

nc 127.0.0.1
3456

nc traffic between pc and server

ssh

ssh tunnel
between pc and router2

ssh process on server receives
the packets within the ssh data
session, de-capsulate them and
forward them to server

router2

ssh

decapsulated nc traffic

ssh process on pc
intercepts nc packets and
encsapsulate them into a
ssh tuunel

server

nc -l -p
2024

SSH remote port forwarding

- Remote port fowarding
  - `ssh -NR r_port:local_addr:l_port user@server`
- In the previous example, we want to connect a tcp socket port 3000 from router2 to pc:2000
  - `pc# ssh  -NR 3000:10.0.0.100:2000 user@8.0.0.2`
- We put nc in linstening on pc
  - `pc# nc -l -p 2000`
- We connect nc from router2
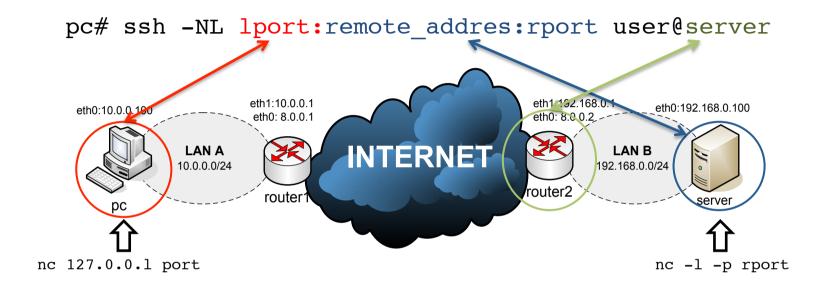 - `router2# nc 127.0.0.1 3000`
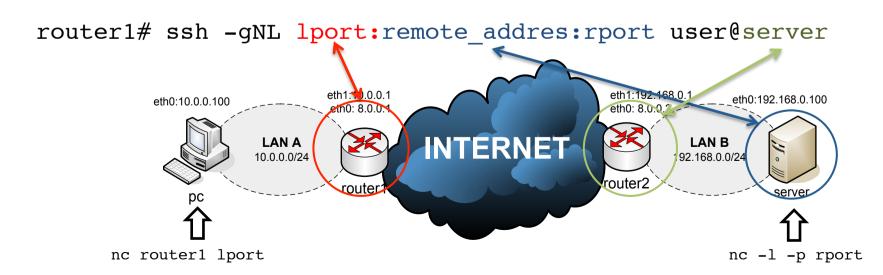
SSH port forwarding "for everyone"

- We can also set up a gateway that forwards ports for all hosts in a LAN

- For example, we can run ssh local port forwaring on router1 for all hosts in LAN A
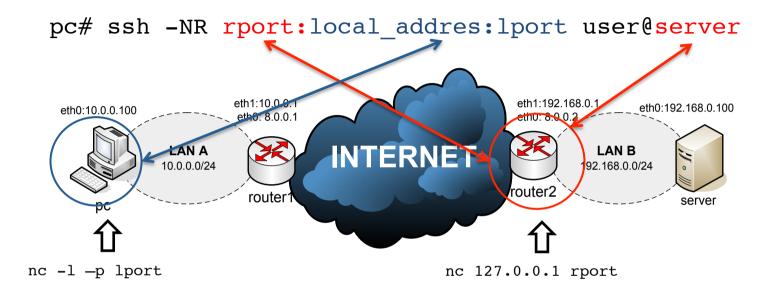 - `router1# ssh -NL 3456:192.168.0.100:2024 user@router2 -g`

- For remote port forwarding there's no "-g" option
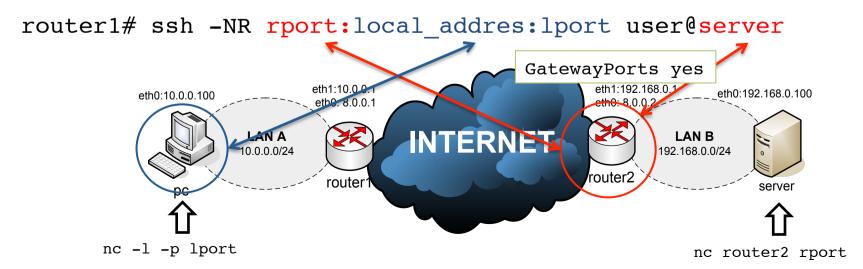 - We have to set the following config option in sshd_config
    - `GatewayPorts yes`

SSH local port forwarding explained

```
pc# ssh -NL lport:remote_addres:rport user@server
```


nc 127.0.0.1 port

nc -l -p rport

```
router1# ssh -gNL lport:remote_addres:rport user@server
```


nc router1 lport

nc -l -p rport

SSH remote port forwarding explained

`pc# ssh -NR rport:local_addres:lport user@server`

nc -l —p lport

nc 127.0.0.1 rport

`router1# ssh -NR rport:local_addres:lport user@server`

GatewayPorts yes

nc -l -p lport

nc router2 rport

Shared screen and X forward with SSH

- Useful trick to share the same remote screen
  1. ssh to my machine marlonmac.local (or let's see what address I have now…) with the user "student" password "student"
  2. Attach to a already attached screen with "screen -r -x"
  3. Have fun!

- To run a graphic application on server
  - Set on server `sshd_config`:
    ```
    X11Forwarding yes
    ```
  - Run ssh on client
    ```
    client# ssh -X user@server
    ```
  - Run a graphic app on client
    ```
    client# xclock
    ```

SSH SOCKS5 proxy

Example: ssh -ND 9999 username@server

SSH SOCKS5 test

- Copy Lab1-ssh/web_page_test/* into server:/var/www
- Configure firefox on the host machine to use a SOCKS5 local proxy
- Use router2 as relay to server

- Start apache in VM "server"
- Open the web page http://192.168.0.100, which is VM "server"

rsync

- Rsync is a fast and versatile file copying tool
- Rsync copies files either to or from a remote host, or locally on the current host
- Delta-transfer Algorithm
  - reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination
- Two modes:
  1. Through a secure shell (ssh, rsh)
  2. Contacting a remote rsync daemon directly via TCP
- Basic usages (man for the options..):
  ```
  rsync -avz --progress foo:src/bar/ /data/tmp
  rsync -av src/ dest/
  rsync -av --delete host::src /dest
  rsync -avd rsync://host:src /dest
  rsync -ravz --exclude="*.o" foo:src/bar /data/tmp
  ```
- Nice tutorial
  - http://www.thegeekstuff.com/2010/09/rsync-command-examples/

Simple backup script with rsync in Lab1-ssh

```sh
#!/bin/sh
LOCAL=/root
REMOTE=/var/backup
HOST=8.0.0.1
LOG=/var/log/backup.log
SYNCLOG=/var/log/backup.synclog

#start log
echo $(date +"%d/%m/%Y") | cat >> $LOG
echo $(date +"%H:%M.%S") backup started... | cat >> $LOG

#Rsync
rsync --delete -azv -e ssh $LOCAL root@$HOST:$REMOTE | cat > $SYNCLOG

#end log
echo $(date +"%H:%M.%S") backup ended! | cat >> $LOG
```

1) Save the script in

    /bin/rsyn_backup.sh

2) Make it executable

    chmod +x /bin/rsyn_backup.sh

3) Add the cron job with the command

    crontab -e

4) Put the following line

    0 4 * * * /usr/local/bin/rsync_backup.sh

wget

- GNU Wget is a free utility for non-interactive download of files from the Web
- It supports HTTP, HTTPS, and FTP protocols, as well as retrieval through HTTP proxies
- Wget is non-interactive, meaning that it can work in the background, while the user is not logged on. This allows you to start a retrieval and disconnect from the system, letting wget finish the work
- Basic usage:
  - `wget` *http://www.example.com/*
- Recursive download (1 folder):
  - `wget -l 1 -r byron.netgroup.uniroma2.it/ ~marlon/RAT`

  (Change 1 → "n" for more levels...)

wget - mirroring

```
wget  --recursive --no-clobber --page-requisites --adjust-
extension --convert-links --restrict-file-names=windows --
domains website.org --no-parent website.org
```

- --recursive: download the entire Web site
- --domains website.org: don't follow links outside website.org
- --no-parent: don't follow links outside the directory tutorials/html/
- --page-requisites: get all the elements that compose the page (images, CSS and so on)
- --adjust-extension: save files with the .html extension
- --convert-links: convert links so that they work locally, off-line
- --restrict-file-names=windows: modify filenames so that they will work in Windows as well
- --no-clobber: don't overwrite any existing files (used in case the download is interrupted and resumed)