

A project completed as part of the
requirements for the
B.Sc. (Hons) Computer Science
entitled

Web Application Security Principles

Designing Secure Web Based Enterprise Solutions

by

Johann REHBERGER

in the years 2002/2003

ABSTRACT

This project is a research about software development security principles in web applications. Security vulnerabilities of web applications are researched and discussed in detail. The work examines existing security principles for application development and recommendations for implementing the researched principles in web applications are given. In addition an investigation on the design elements of enterprise web applications is made and the core components of web applications are defined.

For demonstration purposes a web application has been implemented to show how a web application that is vulnerable to researched issues looks like. This report gives detailed information how the code should be implemented to prevent those vulnerabilities. The web application can be found on the attached compact disc.

The project comes to the conclusion that “Input Validation” is the most important security principle for web applications. By applying this principle most vulnerabilities can be prevented.

ACKNOWLEDGMENTS

I would like to thank Mr. Dipl.-Ing. Johann Preissl for his guidance and assistance as supervisor for this Final Year Project. In addition I would like to thank Ms. Dipl.-Ing. Dr. Elke Stangl for her advisories and contributions to this Final Year Project.

During the research of this project I had the chance to read a lot of interesting and exciting material from different authors that help me to see things in different ways. Therefore I would like to thank those people.

Last but not least I would like to say thank you to Maria, Leopold, Hubert, Rita and last but not least Andreas. L. Zeiner who supported me to make this project reality.

TABLE OF CONTENTS

| | |
|----------------------------------|----|
| ABSTRACT | 2 |
| ACKNOWLEDGMENTS..... | 3 |
| TABLE OF CONTENTS | 4 |
| LIST OF FIGURES..... | 8 |
| LIST OF TABLES..... | 8 |
| 1 INTRODUCTION | 9 |
| 1.1 Generally..... | 9 |
| 1.2 Aims of the report..... | 11 |
| 1.3 Aims of the sample code..... | 11 |
| 1.4 Approach..... | 12 |
| 2 SOFTWARE SECURITY | 14 |
| 2.1 What is security?..... | 14 |
| 2.2 Security Threats..... | 20 |
| 3 SECURITY VULNERABILITIES | 23 |
| 3.1 Vulnerability defined..... | 23 |
| 3.2 Buffer Overflow | 24 |
| 3.3 Code Injections | 30 |
| 3.3.1 <i>SQL Injection</i> | 30 |

| | | |
|--------|--|-----------|
| 3.4 | Cross Site Scripting..... | 36 |
| 3.5 | URL Traversal..... | 41 |
| 4 | SECURITY VERSUS USABILITY | 44 |
| 4.1 | Generally..... | 44 |
| 4.2 | Server | 45 |
| 4.3 | Client..... | 48 |
| 5 | PRINCIPLES OF SECURE DEVELOPMENT | 50 |
| 5.1 | Generally..... | 50 |
| 5.2 | Existing Principles | 51 |
| 5.2.1 | <i>Validate Input.....</i> | <i>54</i> |
| 5.2.2 | <i>Validate Output</i> | <i>56</i> |
| 5.2.3 | <i>Fail Securely</i> | <i>59</i> |
| 5.2.4 | <i>Keep it simple</i> | <i>59</i> |
| 5.2.5 | <i>Use and Reuse Trusted Components / Use your community resources</i> | <i>60</i> |
| 5.2.6 | <i>Practice Defence in Depth</i> | <i>61</i> |
| 5.2.7 | <i>Secure the weakest link</i> | <i>62</i> |
| 5.2.8 | <i>Security by Obscurity, Transparency, Ease of Use.....</i> | <i>62</i> |
| 5.2.9 | <i>Principle of Least Privilege.....</i> | <i>63</i> |
| 5.2.10 | <i>Compartmentalization, Segmentation</i> | <i>64</i> |
| 5.2.11 | <i>Promote privacy / Reduce surface area</i> | <i>65</i> |

| | | |
|--------|--|----|
| 5.2.12 | <i>Remember that hiding secrets is hard</i> | 65 |
| 5.2.13 | <i>Be reluctant to trust</i> | 66 |
| 5.2.14 | <i>Use secure defaults</i> | 67 |
| 5.2.15 | <i>Check at the gate</i> | 67 |
| 5.2.16 | <i>Assume external systems are insecure</i> | 68 |
| 5.2.17 | <i>If you don't use it, disable it</i> | 69 |
| 6 | ELEMENTS OF A SECURE DESIGN | 70 |
| 6.1 | Web Based Enterprise Solutions | 70 |
| 6.2 | Authentication | 71 |
| 6.2.1 | <i>Anonymous Access</i> | 71 |
| 6.2.2 | <i>HTTP Basic Authentication [RFC 2617]</i> | 72 |
| 6.2.3 | <i>HTTP Digest Authentication [RFC 2617]</i> | 72 |
| 6.2.4 | <i>Forms Authentication</i> | 73 |
| 6.2.5 | <i>Integrated Windows authentication</i> | 73 |
| 6.2.6 | <i>Digital Certificates</i> | 75 |
| 6.3 | Authorization | 76 |
| 6.4 | Auditing | 78 |
| 6.5 | Privacy | 80 |
| 6.6 | Integrity | 82 |
| 6.7 | Availability | 82 |
| 6.8 | Nonrepudiation..... | 83 |

| | | |
|------|--------------------------------------|-----|
| 7 | DIFFERENT DESIGN APPROACHES | 84 |
| 7.1 | About the programming language | 84 |
| 7.2 | Intranet..... | 86 |
| 7.3 | Internet..... | 87 |
| 7.4 | Web Services | 88 |
| 8 | CONCLUSION..... | 90 |
| 9 | CRITICAL EVALUATION..... | 95 |
| 10 | BIBLIOGRAPHY | 97 |
| 11 | APPENDIX..... | 107 |
| 11.1 | Proposal | 108 |
| 11.2 | Meeting Protocols..... | 109 |
| 11.3 | Project Plan | 118 |
| 11.4 | Interim Report..... | 119 |

LIST OF FIGURES

| | |
|---|----|
| Figure 2 The stack | 26 |
| Figure 3 Overridden buffer | 27 |
| Figure 6 Login mask [the application can be found on the CD]..... | 31 |
| Figure 7 Input that evaluates always to true | 33 |
| Figure 8 Dropping a table from the login mask | 34 |
| Figure 9 Message Forum..... | 37 |
| Figure 10 Output..... | 38 |
| Figure 11 Displaying the cookies | 39 |
| Figure 12 Usability vs. Security [Source: Howard, Levy, Waymire 2000, p. 6].... | 44 |
| Figure 13 Existing security principles for software development..... | 53 |

LIST OF TABLES

| | |
|---|----|
| Table 1 Vulnerabilities reported from 1995-2002 [Source: CERT 2003] | 9 |
| Table 2 Top Ten Vulnerabilities of 1st Quarter 2002 [Source: SecurityFocus 2002]..... | 10 |
| Table 3 Security Terminology [Source: Sommerville 2000, p. 368] | 14 |

1 INTRODUCTION

1.1 Generally

Over the last years vulnerabilities in software products have increased tremendously. According to the Carnegie Mellon University [CERT 2003] the total number of vulnerabilities reported between 1995 – 2002 is 9,162. Nearly the half (4129 which are 45%) were reported in 2002.¹

| Year | 1995 | 1996 | 1997 | 1998 | 1999 | 2000 | 2001 | 2002 |
|-----------------|------|------|------|------|------|-------|-------|-------|
| Vulnerabilities | 171 | 345 | 311 | 262 | 417 | 1,090 | 2,437 | 4,129 |

Table 1 Vulnerabilities reported from 1995-2002

[Source: CERT 2003]

The word vulnerability is defined in detail in Chapter 3.

¹ The most recent statistics can be found under <http://www.cert.org>

According to [SecurityFocus 2002] the top 10 vulnerabilities of the 1st Quarter 2002 were:

| Number | Name of Vulnerabilities | Date Released |
|---------------|---|----------------------|
| 1. | Multiple Vendor SNMP Implementation Vulnerabilities | February 12, 2002 |
| 2. | PHP Post File Upload Buffer Overflow Vulnerabilities | February 26, 2002 |
| 3. | OpenSSH Channel Code Off-By-One Vulnerability | March 7, 2002 |
| 4. | Multiple Oracle 9i Remote Command Execution Vulnerabilities | February 6, 2002 |
| 5. | Multiple Vendor Java Virtual Machine Bytecode Verifier Vulnerability | March 19, 2002 |
| 6. | Microsoft VBScript Same Origin Policy Violation Vulnerability | February 21, 2002 |
| 7. | Gator Insecure ActiveX Control Vulnerability | February 20, 2002 |
| 8. | ZLib Compression Library Heap Corruption Vulnerability | March 11, 2002 |
| 9. | Microsoft Commerce Server 2000 ISAPI Buffer Overflow Vulnerability | February 21, 2002 |
| 10. | Internet Security Systems BlackICE and RealSecure Buffer Overflow Vulnerability | February 11, 2002 |

Table 2 Top Ten Vulnerabilities of 1st Quarter 2002

[Source: SecurityFocus 2002]

This list shows that security vulnerabilities exist on a broad basis of systems. Reaching from operating systems like UNIX or Windows to programming languages like PHP and VBScript. Runtime environments like the Java Virtual Machine and tools like firewalls (Black ICE) have also vulnerabilities. The vulnerability on the top position is of interest because it is not just a problem on one platform or implementation. It is a failure that spawns across many different vendors of SNMP enabled systems. [CERT ADV 2002]

If you ask somebody on the street how secure a software product should be, he might say high secure or 100% secure. What is a 100% secure system? Is it possible to create a 100% secure system?

1.2 Aims of the report

This project concentrates on the security of web applications. The work figures out vulnerabilities that exist in web applications and shows how they can be prevented by applying the researched security principles.

1.3 Aims of the sample code

The aim of the sample code is to demonstrate researched vulnerabilities. Most of the sample code is implemented in C# and ASP.NET.

1.4 Approach

- Chapter 2 Defines the notion security from different point of views. Different security threat models are researched.
- Chapter 3 In this chapter security vulnerabilities of web applications are researched and code samples are given. The aim is to show how these vulnerabilities might be exploited.
- Chapter 4 This chapter deals with the impact that security has on the usability of web applications. This is done by discussing server and client side components.
- Chapter 5 This chapter gives an overview about existing software security principles. In addition recommendations on how to apply the researched knowledge to build secure enterprise web applications are given.

- Chapter 6 The main elements of a secure web based enterprise application are researched.
- Chapter 7 Design issues are discussed - including Internet-Applications, Intranet-Applications, Web Services and the choice of the programming language.
- Chapter 8 This chapter includes the critical evaluation of the author. It highlights weaknesses and strengths of this work.
- Generally References and applications are included on the attended CD.

2 SOFTWARE SECURITY

In this chapter “security” is defined from different point of views. It is shown that security is a property of software and therefore has to be considered from the beginning of a project.

2.1 What is security?

According to [Sommerville 2000, p. 545] security is an attribute of the overall software quality. There are some words that are often used in the context of security. The following table gives a definition of these terms:

| | |
|----------------------|--|
| <i>Exposure</i> | <i>Possible loss or harm in a computing system.</i> |
| <i>Vulnerability</i> | <i>A weakness in a computer-based system that may be exploited to cause loss or harm</i> |
| <i>Attack</i> | <i>An exploitation of a system vulnerability</i> |
| <i>Threats</i> | <i>Circumstances that have potential to cause loss or harm</i> |
| <i>Control</i> | <i>A protective measure that reduces a system vulnerability</i> |

Table 3 Security Terminology [Source: Sommerville 2000, p. 368]

“The security of a system is an assessment of the extent that the system protects itself from external attacks that may be accidental or deliberate. Examples of attacks might be viruses, unauthorised use of system services, unauthorised modification of the system or its data, etc.”

[Sommerville 2000, p. 367]

This statement points out that security is not just important because of attacks which are planned but also of accidentals.

“Remember: security is not something that can be isolated in a certain area of the code. Like performance, scalability, manageability and code readability, security awareness is a discipline that every software designer, developer, and tester has to know about.”

[Howard, LeBlanc 2001, p. 22]

[Sommerville 2000, p. 367] states that errors in the development of a system can lead to security loopholes. This means that a bug in a product can lead to a security issue. Software is complex and has bugs. This implies that there might never be a complete secure system.

“In the real world your software will likely never be totally secure.”

[Viega, McGraw 2002, Preface xxiv]

We know that systems can not be totally secure and might be broken. Therefore it is important to audit system access. This will not prevent others from entering the system but the unauthorised access attempts are logged. Important information about the methods and strategies of attackers can be gained by analysing the log-files. This information can be used to design more resistant systems, because the strategies of the attackers are better known.

“Know your enemy, know yourself, and in 100 battles you will never be defeated.”

[Tzu 500BC]

There is a project that focuses on these auditing, monitoring and analysing purposes in networks called “The Honeynet Project”.²

² <http://www.project.honeynet.org>

“The HoneyNet Project is about setting up a system that is only there to be attacked. Tracing, auditing and monitoring these attacks will allow you to understand how the ‘blackhat’ community works. Which tactics they use and how they try step by step to break into your system.”

[Spitzner 2002]

Founder of the HoneyNet Project

The information that can be retrieved from a HoneyNet is tremendous. Mr. Spitzner sees security from a military perspective. By observing and analysing the attacks of a HoneyNet a lot of useful information can be achieved. Like scouts in military organisations, HoneyNets are used to identify the enemy, find out how they are acting and which weapons they use.

Security requirements (like others) change over time. It is typical for software that the overall design has to be adapted to fulfil the requirements. A design process that is aware of this agile process like the spiral model [Boehm 1988] allows better integration of security than the *waterfall* approach [Sommerville 2000, p. 367]. Security has to be integrated in all phases of the application development cycle, starting in the early design phases.

Adding security later on might create other risks because it is not completely integrated into the system. This is true not just for software. For instance think of civil aircrafts with pilots wearing guns or having access to guns for safety reasons during flight. This is a good example for security added later on and might bring more risks than it prevents. Possible attackers do not even need to bypass security checks on the ground to bring a weapon on board. The main point is that security cannot be added later on. It has to be included in the overall design process.

According to [Pfleeger 2001, p. 119], a security plan should be part of the overall project plan. This is also often called a “Security Policy”. [BS7799]

“Security involves the protection of assets, where assets are defined as anything with value.”

[Howard, LeBlanc, Waymire 2000, p. 23]

This implies that a non secure system allows access to protected data. But what is an asset with value? Who says that an asset has more value than another? Threats and risks have to be defined and prioritised and a plan has to be defined for the risks with the most impact.

“The fundamental technique is to begin early, know your threats, design for security and subject your design to thorough objective risk analyses and testing.”

[Viega, McGraw 2002]

Security has to be seen in the context of many other concerns. Decisions have to be made, what should be secured to which degree? What will happen if someone breaks into your system because it was not secure enough?

Risk Management has to be undertaken. Storing the credit-card numbers of customers becomes a problem when somebody breaks into your system and steals those numbers. The impact of this scenario would be rather big. After such issues customers may leave and go to competitors. According to [MSF 2002] the top risks have to be defined. This has to be seen in a more general context – security is not the only issue. Resources (money, employees), Time-To-Market or the usability of a system are other important factors. It has to be defined how much effort should be invested into the security of a system.

The process of risk management is a huge topic and not part of this work.

2.2 Security Threats

Nowadays most computer systems are connected together through the Internet. Everybody working on a PC connected to the Internet is a possible victim of an attack. Before the Internet was so widely spread and mainstream the overall damage an attack could make was much more limited. Today the amount of victims has grown tremendously. In addition the details of a specific vulnerability can be distributed easily to a broad audience via the Internet. This means that also people without the explicit knowledge of how the system internals work are able to make serious damage. Those people are often called *script kiddies*.³

According to [Sommerville 2000, p. 167] there are three types of damage that may be caused through an external attack:

1. Denial of service
2. Corruption of programs or data
3. Disclosure of confidential information

Denial of service attacks might lead to inoperable services because the attacked systems cannot be reached. For instance an online web shop might be

³ see [Spitzner 2002, p. 87] and [Viega, McGraw 2002, p. 5]

attacked and therefore the service is not available for customers. This has direct business impact on the company, leading to loss of revenue. The corruption of programs or data means that programs or data might be altered. This can lead to systems acting in unpredictable ways – the reliability of the system is in danger. Confidential information like credit card information might be stolen by attackers leading to privacy loss.

In addition [Howard, Levy, Waymire 2000, p. 23] define the STRIDE model which defines security threats:

- S** *Spoofing user identity*
- T** *Tampering with data (integrity)*
- R** *Repudiability*
- I** *Information disclosure (disclosure)*
- D** *Denial of Service*
- E** *Elevation of privilege*

There are three additional threats defined in the STRIDE model compared to the threats listed by Sommerville. These are Spoofing users' identity, repudiability and the elevation of privileges.

Spoofing another user's identity happens when an attacker uses another person's identity to gain access to a system. This can be done by guessing passwords or stealing the session identification and hijack the session.

Repudiability means that a user might perform illegal operations which get not traced because the system lacks this ability.

The elevation of privilege means that unprivileged users gain privileged access to the system. The attacker becomes member of a trusted system and can cause damage.⁴

⁴ compare with [Howard, Levy, Waymire 2000, p.19-20]

3 SECURITY VULNERABILITIES

This chapter discusses the main security vulnerabilities that exist in web applications. The vulnerabilities are explained in detail with code samples. There exist more vulnerabilities (e.g. different kind of code injections) but they rely on those researched in this chapter.

3.1 Vulnerability defined

“Vulnerability is a weakness in a computer-based system that may be exploited to cause loss or harm.”

[Sommerville 2000, p. 268]

“A vulnerability is a weakness in a system, such as a coding bug or a design flaw.”

[Howard, LeBlanc 2002, p.36]

The second statement clearly states that vulnerabilities can come from coding errors and design failures. The second statement implies that every bug is a vulnerability. In my opinion the statement from Sommerville is more appropriate because it distinguishes between bugs and bugs that might be exploited.

3.2 Buffer Overflow

According to the [CERT ADV 2002] buffer overflows are responsible for more than 50% of all software security leaks.

There are famous worms that used buffer overflows:

- Buffer overflow in fingerd in 1988 [Spafford 1991]
- Nimda Worm [SYMANTEC 2002]
- SQL.Slammer in January 2003 [CERT ADV 2003]

What is a buffer overflow?

A program needs to store information in memory. Therefore the application has to allocate space. This allocated space is the buffer. There are two storage places, the stack and the heap. This work just considers buffer overflows that can occur by allocating memory on the stack. According to [Viega, McGraw 2002, p. 155] heap overflows are more difficult to achieve, but it is also possible.

The buffer can be used to store information. The programming language C does not check if you insert too much data into the buffer. The memory space after the allocated buffer gets overridden when a data block is inserted that is over the size of the allocated memory block – this is called a buffer overflow or sometimes also referred to as buffer overrun.⁵

A sample – The stack overflow

```
#include <stdio.h>

void callCode(char* input)
{
    char buffer[4];          /* 4 character buffer */
    strcpy(buffer, input);  /* copy the input into 4 char buffer */
    printf("The buffer holds: %s\n", buffer);
}

int main(int argc, char* argv[])
{
    callCode(argv[1]);     /* call the function with the given input */
}
```

Figure 1 Buffer overflow example

⁵ compare with [Howard, LeBlanc, p. 63]

What happens when a 6 character string is copied into a 4 character buffer?
When a buffer overflow occurs the following can happen according to [Viega, McGraw 2002, p. 138]:

- Program acts in an unpredictable way
- Program could fail completely
- The execution goes on without any noticeable difference in execution

When a function gets executed the arguments are pushed onto the stack. Then the base pointer and the instruction pointer are pushed on the stack. Afterwards the execution jumps to the function and space for the buffer variable is allocated.

At this time the stack looks like:

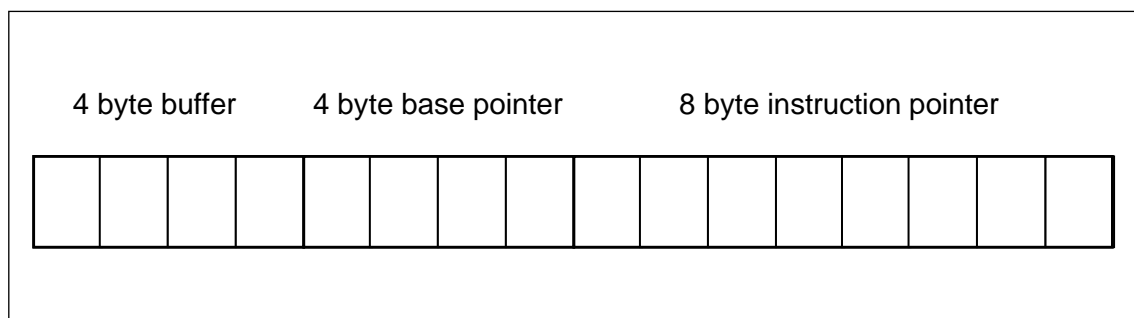


Figure 2 The stack

If the input has more than 4 bytes the base pointer gets overridden and if the input exceeds 8 bytes also the instruction pointer (which holds the return address to the calling function) is overridden.

For instance if 12 characters are copied into the buffer it would overrun and the stack would look like:

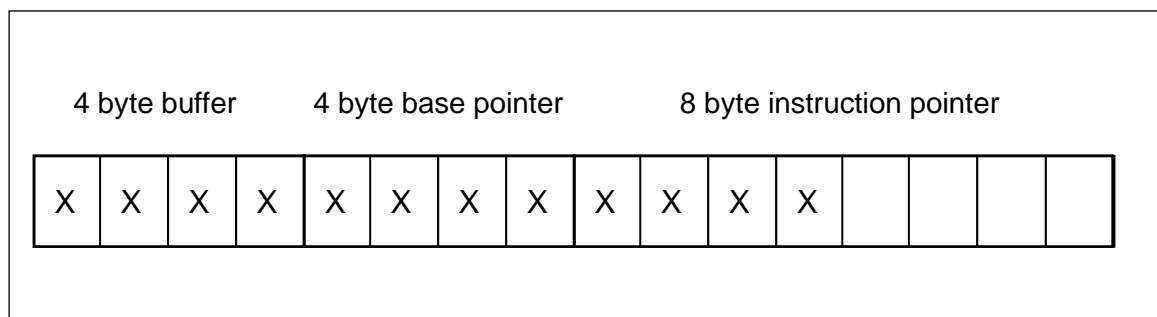


Figure 3 Overridden buffer

An application reacts on such input as shown below:

```
mano# gcc stackoverflow.c -o stackoverflow
mano# ./stackoverflow 123
the buffer holds: 123
mano# ./stackoverflow 123456789012
mano# Feb 14 20:55:44 mano /kernel: pid 248 (stackoverflow), uid 0:
exited on signal 11 (core dumped)
```

Figure 4 executing the code

This leads to a crash of the program. Taking a look into the registers using the debugger gdb⁶:

```
mano# gdb ./stackoverflow ./stackoverflow.core
(gdb)info registers
eax          0x19      25
ecx          0x280ea478    672048248
edx          0xbfbffb74    -1077937292
ebx          0x2        2
esp          0xbfbffb0     0xbfbffb0
ebp          0x38373635    0x38373635
esi          0xbfbffc5c    -1077937060
edi          0xbfbffc68    -1077937048
eip          0x32313039    0x32313039
eflags      0x10282    66178
```

Figure 5 registers during the crash

The registers show that the instruction pointer (eip) holding the value of 0x32313039 (in decimal this is 9012) – which are the last 4 bytes that we entered. The hex value has to be read from back to forth. This value is not a valid return address. This is the reason why the application crashed.

⁶ Copyright 1998 Free Software Foundation, Inc

The instruction pointer could be overridden with a value of a valid address and the program execution would jump to this address and code execution would continue.

Whereas buffer overflows might not directly subject web applications, it has to be considered that components (dynamic link libraries) which are used by web applications are written in a programming language like C. If the web application does not correctly handle user input, functions of the DLL might be called and a buffer overflow can be exploited.

3.3 Code Injections

Injection vulnerabilities happen when data entered by a user becomes executable. Examples are SQL, Server Side Includes or PERL. To show how this kind of attack works the SQL Injection has been researched in detail. For demonstration purposes a sample application and database have been implemented.

3.3.1 SQL Injection

SQL the Structured Query Language for database systems is a powerful way to retrieve and update data in a database system. Web applications that do not validate the input correctly could be injected with SQL code that allows the attacker to retrieve sensitive information and in the worst case to gain access to the whole system.

Assume the following web application (this application has been implemented and can be found on the compact disc included with this project):

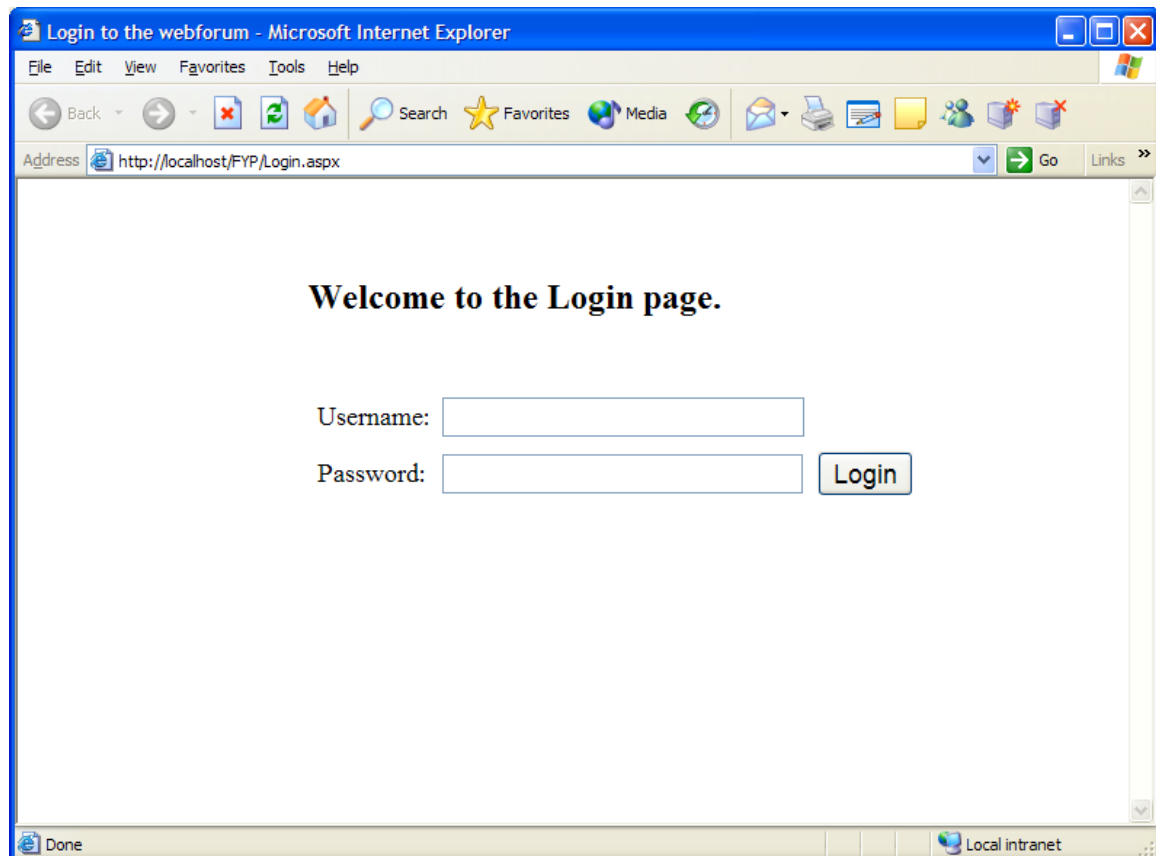


Figure 6 Login mask

[the application can be found on the CD]

This application has to validate the entered information (username and password) against a database to figure out if the entered user exists and the password is correct.

This is often done with a concatenated SQL query string which is generated like:

```
loginSql = "select userid from users where username='" +  
            TextBoxUsername.Text + "' AND password='" +  
            TextBoxPassword.Text + "'";
```

Analysing the SQL code which is executing against the database with a profiling tool⁷ shows that the following statement is executed against the database system:

```
exec sp_executesql N'select userid from users  
where username='mano' AND password='p@ssword'''
```

We can knock on the door of the application by entering:

```
` OR 1=1; --
```

The ` at the beginning correctly closes the first SQL statement. The *OR 1=1* statement makes a valid SQL query that is always true. By adding two dashes at the end the following code is considered as a comment by the database system.

⁷ In this case the "Profiler" which is part of Microsoft SQL Server 2000 is used.

For instance, this input creates a valid SQL query:

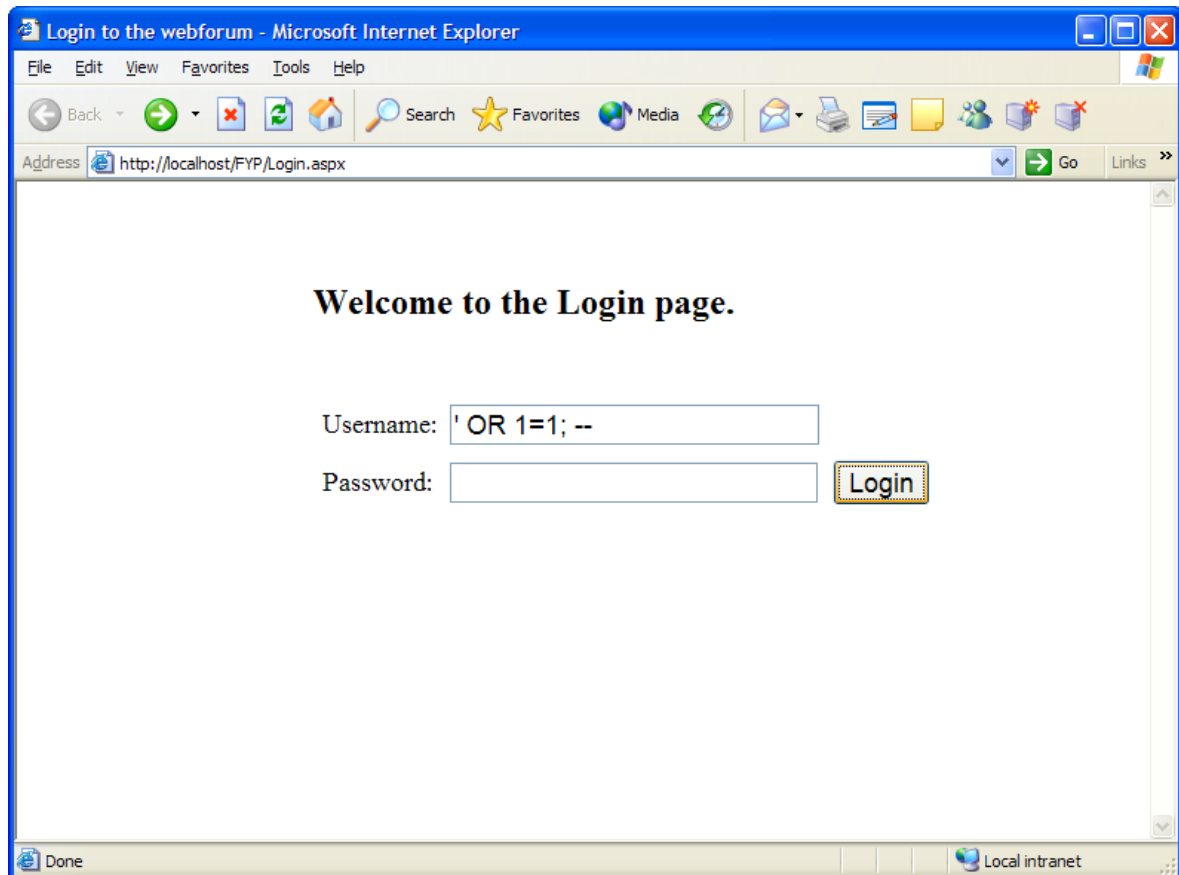


Figure 7 Input that evaluates always to true

The profiler shows the following statement which was executed:

```
exec sp_executesql N'select userid from users where username='' OR  
1=1; --'' AND password='''''
```

After this statement is executed we are logged into the system. If the database system runs under an over privileged account the outcomes might be fatal.

Let's assume the user that is used to connect to the database has database owner privileges in the database:

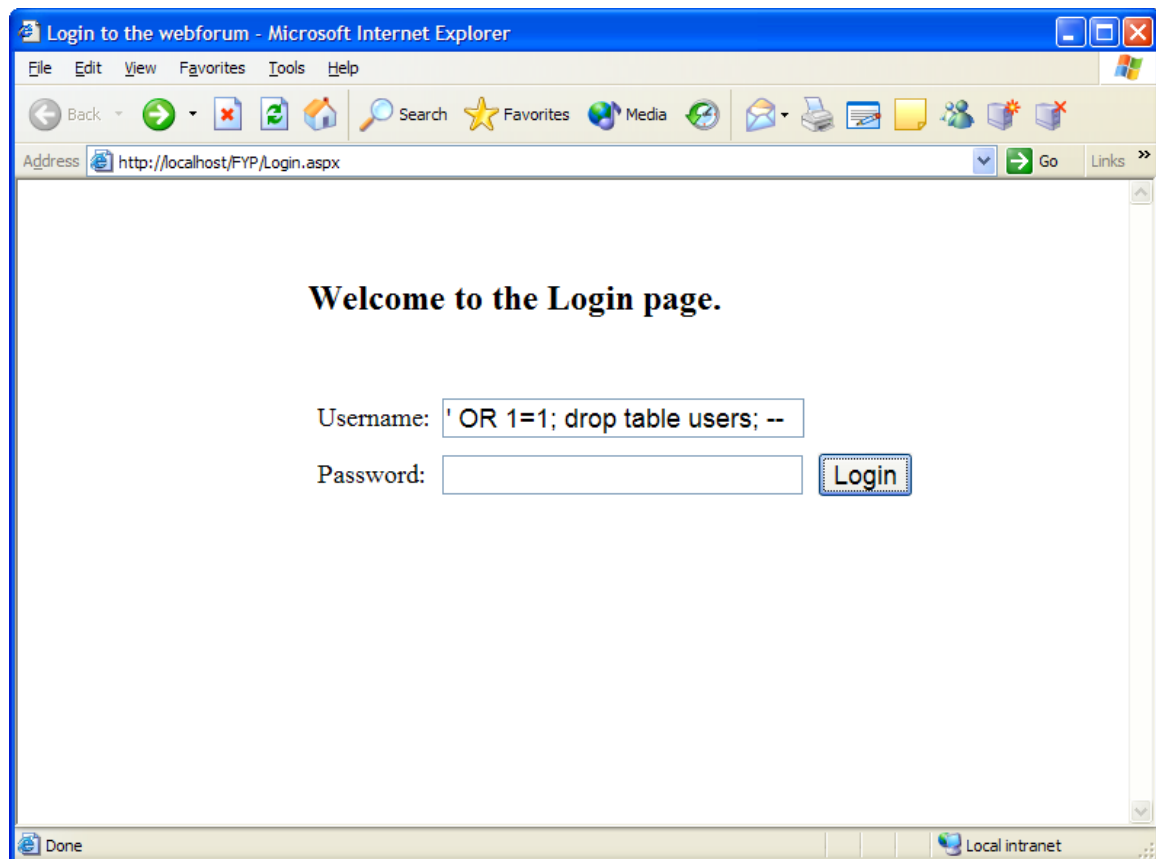


Figure 8 Dropping a table from the login mask

The SQL Server Profiler shows that this gets executed against the database system:

```
exec sp_executesql N'select userid from users where username='' OR  
1=1; drop table users; --'' AND password=''''
```

This drops the users table from the database.

3.4 Cross Site Scripting

Cross site scripting are a relatively new class of attacks in comparison to buffer overflows. The name comes from [CERT 2000] and was published first in February 2000. To show the aims of a Cross Site Scripting attack a sample message forum application has been implemented.⁸

Session Cookie handling

Web applications have to create some kind of session information because of the nature of the HTTP protocol. This session identifier is normally created upon the logon of the user to the application. Another person could sniff this information that is sent by the client with every request. For instance in ASP.NET this session identifier is a cookie which is sent by the client through an HTTP header or the URL. With Cross Site Scripting it is possible to gain this information. The session identifier can then be used to hijack the session.

⁸ This application can be found on the compact disc.

This is done by sending requests to the web application with the session identifier (cookie) from another valid session. I would prefer the use of session cookies via an HTTP header rather than sending the cookie in the URL because the information in the URL is stored in the browser history.

The implemented application looks like:

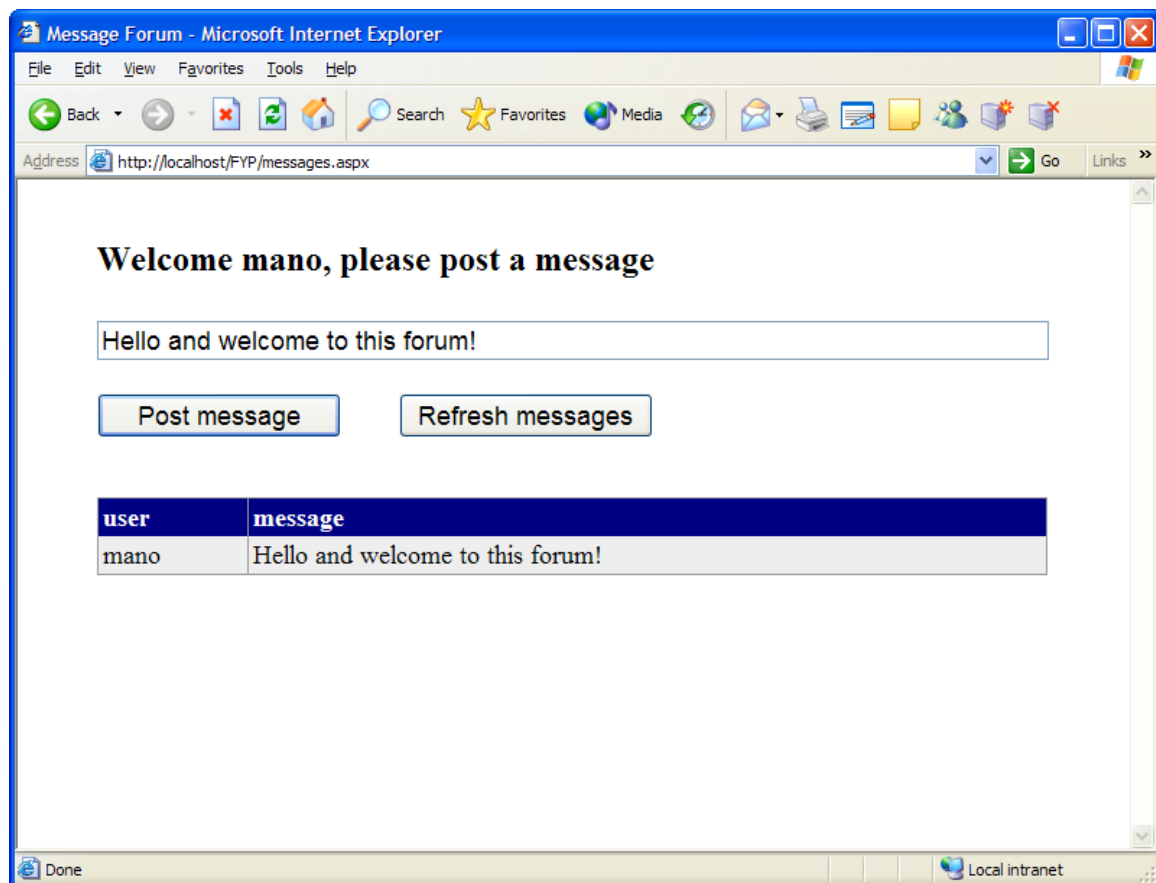


Figure 9 Message Forum

In the sample application above it is possible to post the following message to the forum:

```
<script language="javascript"> alert("hello"); </script>
```

The next time the messages are refreshed the message above is read and integrated into the HTML page which ends up with a message box that looks like:

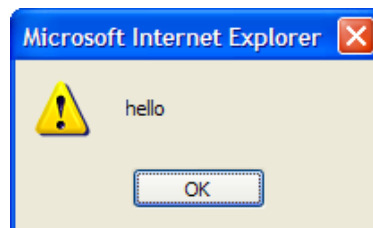


Figure 10 Output

Attackers with knowledge of JavaScript can for instance display the cookie information in the forum with:

```
<script language="javascript"> document.write(document.cookie+"\n");
```

This leads to the following response of the application:

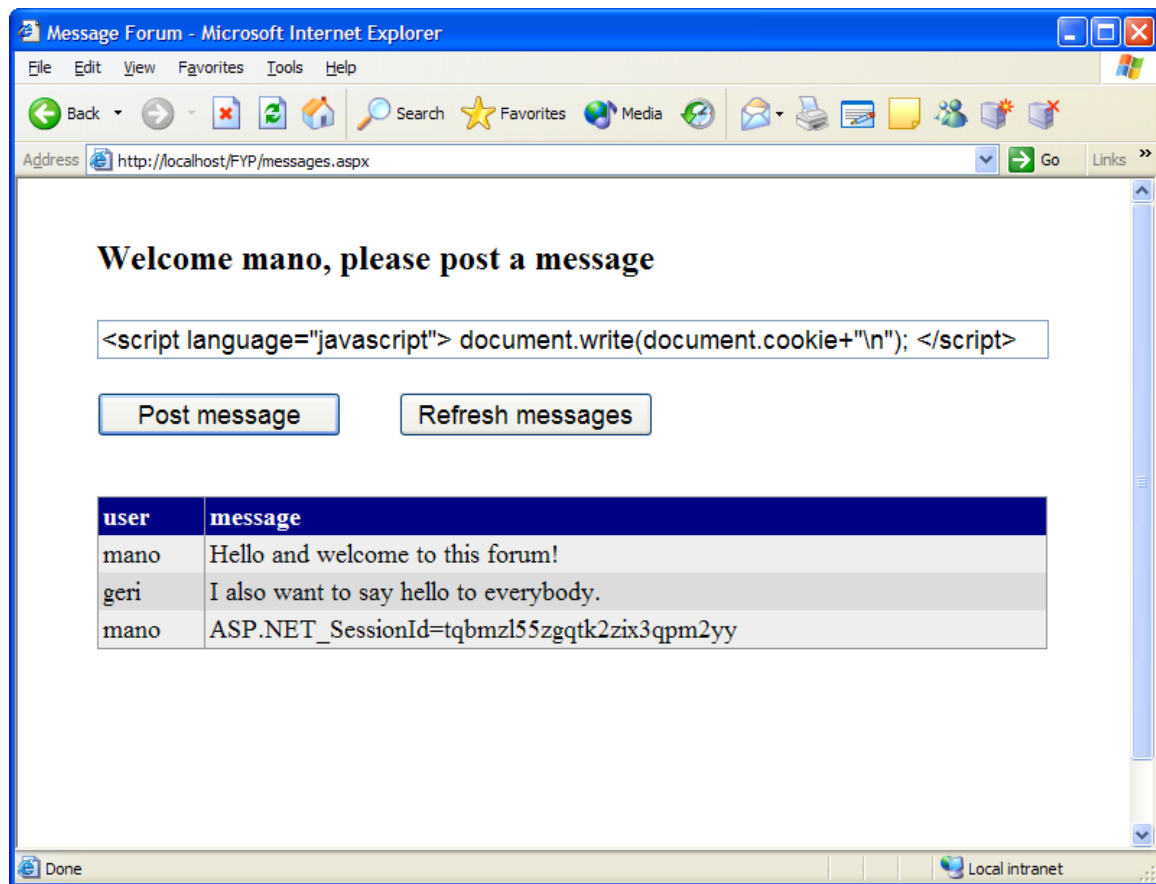


Figure 11 Displaying the cookies

We can use this information (the session cookie) and send it to a third server with:

```
<script language="javascript"> image = new Image();  
image.src = "http://ThirdServer/cookie.gif"+document.cookie;
```

With this script the client sends the cookie information to a third server. The information posted could be used to hijack the session of the user because the session cookie has been stolen. A web application is vulnerable to Cross Site Scripting if it returns data users entered without proper validation and encoding.

Client side validation of input information is not enough. There are tools available that allow bypassing this client side scripting validations.

3.5 URL Traversal

For researching purposes a Windows XP system running the web server IIS 5.1 was installed and connected to the Internet. After days of running the web server the log files were analysed. The following entries could be found⁹:

```
GET /scripts/../../../../winnt/system32/cmd.exe 404
GET /scripts/../../../../winnt/system32/cmd.exe 404
GET /scripts/../../../../winnt/system32/cmd.exe 404
```

These entries are chosen because they show a Unicode Web Traversal attacks. The attacker tries to gain access to directories outside of the virtual root. The request shown above is typical for the [W32.Nimda.A@mm](#) first discovered in September 2001 [SYMANTEC 2001].

The aim of a URL Traversal attack is to jump out of the virtual directory of the web application and execute commands. Because the system runs Windows XP and IIS 5.1 the attack was not successful.

⁹ On a standard installation of Windows XP the log files can be found under %systemroot%\system32\logfiles\W3SVC1\.

A web browser encodes certain characters before it sends the request to the web server. It encodes them to Unicode characters and replaces the 00 with a %.

For instance:

```
\ is Unicode encoded 005c the browser replaces the 00 with % which leads to %5c  
/ is Unicode encoded 0025 the browser replaces the 00 with % which leads to %25
```

The aim of such a traversal attack is to gain unauthorised access to system commands. The following sample shows that aim.

```
http://webserver/../../../../winnt/system32/cmd.exe?/c+ dir
```

This request tries to execute *cmd.exe*. Internet Information Services denies the *../* (dot dot slash). An issue is that an already encoded value can be encoded again – some web servers do have a problem with this. When the */* is encoded to its Unicode representation (*%5c*) and afterwards the *%* is again encoded the attack might work if we have execute permissions in the directory of the web server.

On a standard installation of Internet Information Services 4 and 5 the scripts directory under *c:\inetpub\scripts* has execute permissions. Calling the command from this directory with the following request returns a directory listing.

```
http://webserver/scripts/..%255c..%255c../winnt/system32/cmd.exe?/c+d
```

These URL Traversal attacks are well known and the vendor's statement is described in [MS00-078]. There is a patch available for this vulnerability but installing the virtual root onto a different partition than the system would not have given an attacker the possibility to access system commands at all.

Tools like the URLScan utility support the validation request URLs.¹⁰ Internet Information Services 6.0 will also use this utility. [Berry 2002]

¹⁰ for additional information see URL Scan Utility:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=12244f33-a5da-4203-a3a8-83f4388bb71f&DisplayLang=en>

4 SECURITY VERSUS USABILITY

In this chapter the impacts of security on the usability of web applications are discussed. This is done by examining the server side and the client side of web applications.

4.1 Generally

[Howard, Levy, Waymire 2000, p. 6] explain that there exists a general trade-off between security and usability. They say: “*Secure systems are usually less usable.*”

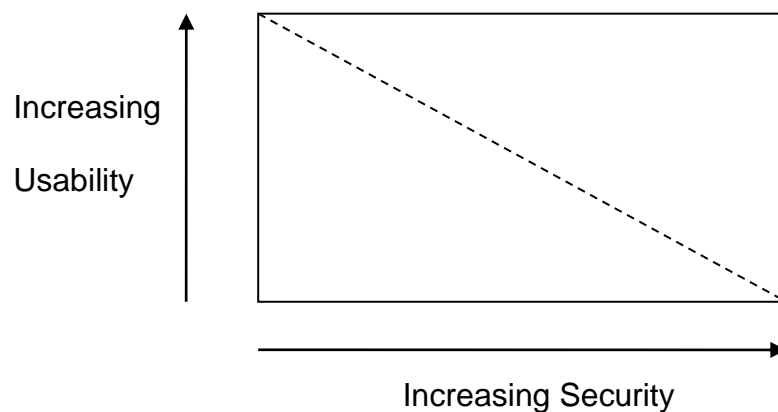


Figure 12 Usability vs. Security

[Source: Howard, Levy, Waymire 2000, p. 6]

4.2 Server

Most software products shipped can be used after the installation process has completed. Users do not have to customise services if they want to use them. Although this might be okay for development or testing purposes, running software in a production environment with the default settings is one of the main security risks.

Microsoft is a company that is famous for ready to use software products – on the desktop and also on the server market. The ease of use by having a graphical user environment to fulfil administrative tasks on the server side is one point that splits system administrators. “Why does a server operating system need a graphical user environment with a 24 bit colour desktop mode, running screensavers that slow down the overall performance of the server?” On the other hand people say that this provides more comfort and the product is easier to use. Many operating systems install additional software components that might not be needed to fulfil the specific task the system is aimed for. Windows 2000 Server is installing Internet Information Services automatically, although you may not even host web sites on the machine. This means an extreme overhead that has to be accomplished by de-installing the services.

For this purpose unattended machine setups should be generated that install only the needed components. Companies should integrate this feature that for instance Windows products provide to allow quick installation of previously defined standard systems. In addition a “slip streamed“ setup¹¹ should be generated to accomplish the integration of the newest service packs and hot fixes at installation time.

According to [Gates, 2002] – the “*Trustworthy Computing Statement*” - future versions of Microsoft products will not be enabled with all features by default. The aim is to provide a system which is “Secure by Default”. The new Windows Server 2003 will have no web server installed by default and also after installing the new IIS 6, it will only allow to run static web pages by default – so system administrators have to turn on those components they need. It will be very interesting to see how the community will react to this new situation because it becomes more difficult to run and administer systems. But this will definitely lead to more secure systems.

¹¹ This means that the operating system has automatically installed the new service pack. Usually a service pack can be “slip streamed” to the base image with the *update* command.

Oracle ships its database system 9i with an Apache server that is started automatically. Or diverse Linux distributions install components by default, which are not needed. It will be interesting to see if the strategy of those companies and distribution providers will also change.

Most recent operating systems have an automatic update service or you can add software to the system with specific tools that ship with the operating system. For instance under FreeBSD 4.7 you can use the *pkg_add* command to install additional components (so called ports).¹² You need to trust software that gets installed in such a way because you may not be able to verify the origin. This can be done with the use of signatures.

For Intranet scenarios I would recommend to generate a dedicated server that is connected to the Internet and retrieves the most recent updates. Clients should connect to this server to download the updates. This design allows better use of the Internet bandwidth and decreases the duration of updates. There is a server product called the Microsoft Software Update Service that provides this functionality.

¹² The ports can be found under <http://www.freebsd.org/ports>

Software designers have to consider that if the software is not usable it is predicted to die. Think about the highly flexible software solutions that have to be configured by dozens of configuration files – system administrators which have grown up with graphical user interfaces do not like those systems and may abandon them. System administrators generally should use scripts to configure the system in a consistent and documented way.

4.3 Client

On the client side the usability is more important than on the server side. For example according to the HTTP protocol after a user requests a web page secured by “Basic Authentication” the browser has to ask the user for a username and password.¹³ This check has to be done by every resource which the client is accessing (e.g. html files, images,...). Browsers simplify this task by asking the user once for this information and cache it for future use.

¹³ compare with [RFC 1945 HTTP1.0] and [RFC 2068 HTTP1.1]

Browsers like Internet Explorer or Mozilla also provide a methodology to store username/password combinations for future uses.¹⁴ This seems to be an useful feature because users often use different credentials when accessing web sites. It is hard to remember those username/password combinations. The usability is increased but what is with the security side? Having stored this information locally on the PC means that the possibility exists that unauthorised people gain access to this information.

¹⁴ Mozilla has the possibility to store the entered username with the according password and the next time you visit the login page of the application the username and password input fields are filled out automatically. Internet Explorer acts in a similar way and presents the stored information after the user starts entering data into the input field, this feature is called *Auto Complete*.

5 PRINCIPLES OF SECURE DEVELOPMENT

The aim of this chapter is to figure out software security principles. This is done by researching existing principles. It is shown how these principles can be applied to web applications and how to prevent the vulnerabilities explained in the previous chapter.

5.1 Generally

“The goal of these principles is to identify and to highlight the most important objectives you should keep in mind when designing and building a secure system. Following these principles should help you avoid lots of common security problems.”

[Viega, McGraw 2002, p. 92]

“A comprehensive security strategy first requires a high level recognition of overall Security Principles.”

[Romanosky 2002]

5.2 Existing Principles

[OWASP GUIDE 2002] define the following security guidelines:

- Validate Input and Output
- Fail Securely (Closed)
- Keep it Simple
- Use and Reuse Trusted Components
- Defence in Depth
- Only as Secure as the Weakest Link
- Security by Obscurity Won't Work
- Least Privilege
- Compartmentalization (Separation of Privileges)

[Viega, McGraw 2002, Chapter 5] define 10 Guiding Principles:

- Secure the weakest link
- Practice defence in depth
- Fail securely
- Follow the principle of least privilege
- Compartmentalize

- Keep it simple
- Promote privacy
- Remember that hiding secrets is hard
- Be reluctant to trust
- Use your community resources

[Meier, Mackman, Vasireddy, Dunner 2002] identify the following principles:

- Adopt the principle of least privilege
- Use defence in depth
- Don't trust user input
- Use secure defaults
- Don't rely on security by obscurity
- Check at the gate
- Assume external systems are insecure
- Reduce surface area
- Fail to a secure mode
- Remember you are only as secure as your weakest link
- If you don't use it, disable it

The following table shows the principles and the resource that defines it.

| Principle | OWASP (2002) | Viega, McGraw (2002) | Meier, Mackman, Vasireddy, Dunner (2002) |
|---|-----------------|-------------------------|--|
| Validate Input, Don't trust user input | ✓ | | ✓ |
| Validate Output | ✓ | | |
| Fail Securely (Closed), Fail to secure mode | ✓ | ✓ | ✓ |
| Keep it Simple | ✓ | ✓ | |
| Use and Reuse Trusted Components / Use your community resources | ✓ | ✓ | |
| Defence in Depth | ✓ | ✓ | ✓ |
| Only as Secure as the Weakest Link | ✓ | ✓ | ✓ |
| Security by Obscurity Won't Work | ✓ | | ✓ |
| Least Privilege | ✓ | ✓ | ✓ |
| Compartmentalization (Separation of Privileges) | ✓ | ✓ | |
| Promote privacy, Reduce surface area | | ✓ | ✓ |
| Remember that hiding secrets is hard | | ✓ | |
| Be reluctant to trust | | ✓ | |
| Use secure defaults | | | ✓ |
| Check at the gate | | | ✓ |
| Assume external systems are insecure | | | ✓ |
| If you don't use it, disable it | | | ✓ |

Figure 13 Existing security principles for software development

The listed principles are not only for web applications but for software applications in general. The given principles are discussed in detail below. Some of those principles go hand in hand but others are in conflict to each other.

5.2.1 Validate Input

Many web applications allow users to input data. To react on the entered information the input has to be processed. Often the user input is forwarded directly to business or even data access components without proper validation. This can lead to unwanted behaviour as described in the previous chapter (Buffer Overflows, SQL/Script Injections, Cross Site Scripting).

The input validation principle is important and vulnerabilities can be avoided by applying an enterprise wide policy. The policy should state that every input is denied by default and that it clearly has to be defined which characters are allowed for specific purposes. Developers have to be aware of this crucial part of their work. Every function every method should be secure on its own by validating input and output. Normally characters like < > % .. are not necessary for users to enter and can lead to vulnerabilities as we have seen in the previous chapters. The input has to be validated.

It is important that this input validation checks are not only done at the client side. Implementing input validation with Java Script on the client side gives the user a better usability but it does not provide security. The information sent by a client can be manipulated, leading to a client sending the characters which are explicitly forbidden. Or the client side scripting functionality can be turned off.

The validation checks can be implemented with regular expressions. In ASP.NET the *RegularExpressionValidator* control can be used to implement the validation check.

First the *ValidationExpression* property has to be defined. For instance for an Austrian postal code it is: `^(A-)?\d{4}`. This is the valid input pattern. Then the *ControlToValidate* property has to be set to the textbox that should be validated against the defined pattern. When the page is posted to the server then the validation is done and if it matches the defined pattern everything is all right. The *RegularExpressionValidator* does the validation check always on the server but in addition it is possible to provide client side checks which will improve usability.¹⁵ Implementing this kind of input validation for every textbox in a web application might be much overhead.

¹⁵ The *EnableClientScript* property is enabled by default.

Therefore a new class should be created which derives from the standard textbox. In addition this textbox implements regular expressions to validate the input. Regular Expression can become very complex.¹⁶ Therefore I suggest using existing resources on the Internet that have templates available for common used Regular Expressions.¹⁷

5.2.2 Validate Output

¹⁶ Regular Expression for an valid IP-Address [taken from <http://regexlib.com>]:

```
^(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])\.(25[0-5]|2[0-4][0-9]|[0-1]{1}[0-9]{2}|[1-9]{1}[0-9]{1}|[1-9])$
```

¹⁷ See <http://regexlib.com> for templates and help about Regular Expressions.

Many applications just return the output that is coming directly from the system. The information provided in those standard output messages is very useful for attackers. An attacker may find out which systems are running behind the web application. This gives an attacker the chance to test for specific vulnerabilities which are known for those systems.

Providing uncontrolled output messages give users information that they should not get. There are two kinds of users. Those who do not understand the output that is generated by the application – given a user friendly output message to them might help them to fix the problem themselves. Others might be those who understand what is going on under the hood and they learn to get familiar with internals of the application.

For instance if a user provides wrong user credentials at the login page the output should be like: "Unknown user or wrong password." The output should never include which of the two values (username or password) are incorrect. By providing information like: "The user does not exist!" an attacker gains information about the active user accounts of the system.

Usually the standard error messages of web applications display the file name in which the error occurred. This information should be avoided. Under

ASP.NET this can be configured in the *web.config* file of the application. With the use of the *customErrors* mode attribute, it is possible to customise error messages generated by the system. I would recommend to set the *customErrors* mode *On* in production environments. This will not display detailed error information (filename, stack trace) to the user.

```
<customErrors mode="On" defaultRedirect="ErrorPage.aspx"/>
```

Figure 14 Do not display detailed error messages

To properly handle exceptions in a secure and consistent way the Exception Management Application Block for ASP.NET should be used.

“The Exception Management Application Block can easily be used as a building block in your own .NET application. If you use it, you will reduce the amount of custom error handling code you need to create, test, and maintain. You will also make your application more robust and easier to debug.”

[Jones, Malcolm, Mackman, Jezierski 2002]

5.2.3 Fail Securely

“Failure is unavoidable and should be planned for.”

[Viega, McGraw 2002, p.97]

Because failures are unavoidable it is important that the system falls in a well defined (secure) mode when a failure occurs. Think of doors during a fire-alarm they normally automatically close to prevent the fire from growing. In an application this might mean that sensitive data becomes unprotected. For instance exceptions are thrown and detailed error messages are displayed (see the previous principle for output validation)

5.2.4 Keep it simple

“Complex design is never easy to understand, and is therefore more likely to include subtle problems that will be missed during analysis. Complex code tends to be harder to maintain as well. And most important, complex software tends to be far more buggy.”

[Viega, McGraw 2002, p. 104]

5.2.5 Use and Reuse Trusted Components / Use your community resources

“Using and reusing trusted components makes sense both from a resource stance and from a security stance. When someone else has proven they got it right, take advantage of it.”

[OWASP GUIDE 2002, p. 10]

Using well-known algorithms that are already tested and running for a long time are a better solution than an own implementation. For instance implementing an own cryptographic algorithm might be a very interesting task for many developers, but using an existing algorithm, which is known to be working is much better. Good cryptographic algorithms are designed in a way that detailed knowledge of the algorithm does not influence its security. This means that a good cryptographic algorithm works because of its design and not because the implementation is hidden.

“Repeated use without failure promotes trust.”

[Viega, McGraw 2002, p. 93]

5.2.6 Practice Defence in Depth

According to [Viega, McGraw 2002, p. 96] the aim of this principles is to have different layers of security. If one layer is inadequate for catching a failure the next layer should catch it. This means that redundancy should be included in the system.

For instance to prevent SQL Injections, input validation should be done at the business layer. With the correct use of stored procedures at the data access layer SQL Injections can also be prevented. This shows two layers where security can be applied to avoid SQL Injection vulnerabilities. In addition stored procedures allow better performance and easier administration. For instance to correctly call a stored procedure in C# the ADO.NET command object with the use of the “*Parameters*” collection have to be used.

```
SqlCommand com = new SqlCommand("usp_GetStockPrice", con);  
  
com.CommandType = CommandType.StoredProcedure;  
  
SqlParameter paramStock =  
com.Parameters.Add("@Stock", SqlDbType.NVarChar, 30);  
  
paramUsername.Value = txtStock.Text  
  
com.ExecuteNonQuery();
```

5.2.7 Secure the weakest link

[Viega, McGraw 2002, p. 93] state that *security is a chain and therefore just as a chain is only as strong as the weakest link, a software security system is only as secure as its weakest component.*

In my opinion social issues are often the weakest link in the security chain. People often choose passwords which reflect their interests or hobbies. By researching these interests, passwords might be guessed. Another sample of social engineering is that for instance helpdesk employees give password information to people who call them. Typically in this scenario there is no identity verification of the caller.

5.2.8 Security by Obscurity, Transparency, Ease of Use

“[...] the main problem stems from a false belief that code compiled into binary remains secret just because the source is not available. This is wrong.”

[Viega, McGraw 2002, p. 69]

People who are able to read machine code can find out what a program does. There are also tools available that disassemble binary code. When programming languages like Java and C# are used this task gets even simpler because the generated code is available in an intermediate form (Byte Code or Intermediate Language). This intermediary format is much easier to read than machine code. With tools like Anakrino¹⁸ it is possible to reverse engineer the source code.

5.2.9 Principle of Least Privilege

“The principle of least privilege states that only the minimum access necessary to perform an operation should be granted, and that access should be granted only for the minimum amount of time necessary.”

[Saltzer 1975]

Services and applications running under accounts having too high privileges (e. g. Administrator or root accounts) have the potential to cause harm. The password for the administrative accounts of Oracle and SQL Server database

¹⁸ <http://test.saurik.net/anakrino>

systems are well known. Systems having enabled these accounts with the standard passwords give attackers the possibility to gain access to the whole system. For instance attackers exploiting SQL Injection vulnerabilities are able to submit commands that harm the database system and even the operating system. The reason for this is that often developers choose a powerful user account to simplify the development and debugging process.

5.2.10 Compartmentalization, Segmentation

“The basic idea behind compartmentalization is to minimize the amount of damage that can be done to a system by breaking up the system into as few units as possible while still isolating code that has security privileges.”

[Viega, McGraw 2002, p. 102]

Every bank is working with this principle. They have many different security and auditing systems installed. You are tracked while entering the bank. People behind the shelf do not have access to a big amount of money – they are not able to hand out a big amount of money to a robber. Adding this principle leads to a more complex design of the overall web application.

5.2.11 Promote privacy / Reduce surface area

“Promote privacy for your users, for your systems, and for your code.”

[Viega, McGraw 2002, p. 109]

Attackers might try to find out which system is running. With this information an attacker can search for a system which is vulnerable to a specific issue that can be exploited. There are tools available like the *whois* and *host* command which can be used to figure out more about a specific Internet domain. Or on the Internet there are web sites which monitor web servers and domains.¹⁹

5.2.12 Remember that hiding secrets is hard

“Security is often about keeping secrets. Users don’t want their personal data leaked. Keys must be kept secret to avoid eavesdropping and tampering.”

[Viega, McGraw 2002, p. 109]

¹⁹ for instance <http://www.netcraft.com>

There are many possibilities to store information like connection strings or passwords. Deciding where and how to store this data is important to the security of a web application. In order to securely store a password in a database, only a hash of the password should be stored and not the password itself in clear text.

Using .NET this can be achieved with the use of the *HashPasswordForStoringInConfigFile* method and the Cryptographic Service Provider. This method takes two arguments the password to be hashed and the algorithm that should be used:

```
string hashPwd =  
FormsAuthentication.HashPasswordForStoringInConfigFile(myPassword, "md5");
```

5.2.13 Be reluctant to trust

According to [Viega, McGraw 2002, p. 112] one thing that has to be noticed is that often by trusting one specific entity you implicitly trust every sub entity that the entity trusts. This can lead to unwanted trust chains.

5.2.14 Use secure defaults

This principle is wrong. It implies that software is “Secure by Default” which is not the case. I do not agree to this principle because software is not secure by default as vulnerabilities have shown in the past. Also the “Trustworthy Computing” statement from Bill Gates clearly states that “Secure by Default” should be standard for software products [Gates 2002].

5.2.15 Check at the gate

“If you design solid authentication and authorization strategies at the gate, you can circumvent the need to delegate the original caller’s security context all the way through to your application’s data tier.”

[Meier, Mackman, Vasireddy, Dunner 2002, p. 6]

The main reason for this principle is a performance issue. This principle is clearly against the principle of “Segmentation”, because components or tiers are not secure on its own, they have to rely on a security check done earlier.

For instance a network that only provides security checks at the firewall is insecure because a single client could use a dial up connection to connect to the Internet and so the network is open to attacks.

5.2.16 Assume external systems are insecure

“If you don’t own it, don’t assume security is taken care of for you.”

[Meier, Mackman, Vasireddy, Dunner 2002, p. 6]

If third party components are used in web applications developers often have to rely on the security of that component – therefore only trusted components from trusted vendors should be used. This principle correlates with the principles of input and output validation because for instance with the use of Web Services a system might retrieve data from a business partner. In this case the caller also has to check if the information sent by the Web Service is valid and vice versa.

5.2.17 If you don't use it, disable it

“You can remove potential points of attack by disabling modules and components that your application does not require.”

[Meier, Mackman, Vasireddy, Dunner 2002, p. 7]

Many products often have services installed which provide surface for an attack but are not used. It is a good advice to disable all services which are not needed.

6 ELEMENTS OF A SECURE DESIGN

In this chapter the elements of a secure design [as given by Howard, LeBlanc and Waymire 2000] are researched and discussed.

6.1 Web Based Enterprise Solutions

Enterprise Solutions are applications that are used by companies to run their businesses. Those applications include and integrate different components and systems. Web based applications are made of different components. [Howard, LeBlanc, Waymire 2000, p. 7] identify the following elements:

- Authentication
- Authorisation
- Auditing
- Privacy
- Integrity
- Availability
- Nonrepudiation

6.2 Authentication

“Authentication is the process of determining if a user or entity is who he/she claims to be.”

[OWASP GUIDE 2002, p.16]

6.2.1 Anonymous Access

This is the authentication where everyone accessing the web site is allowed access and no authentication happens.

This type of authentication (which in fact is no authentication) should only be used when the information on the site holds public content or if authentication check is done at the application layer (see *Forms Authentication*).

6.2.2 HTTP Basic Authentication [RFC 2617]

When a client requests a resource that forces basic authentication the server returns the HTTP code *401 - unauthorized access*. Typically the web browser asks the user for his credentials (username and password) to gain access to the resource. Username and password are transmitted using base 64 encoding. [Wong 2000] This means that the credentials are not encrypted and can be monitored by other users. Secure Socket Layer should be used to guarantee information disclosure.

6.2.3 HTTP Digest Authentication [RFC 2617]

The big advantage over “Basic Authentication” is that “Digest Authentication” does not send the password in clear text over the wire. The transmitted information is hashed using the MD5 algorithm developed by RSA Data Security. For additional information see [RFC 1321].

According to [OWASP PLAN 2003, p. 17] Digest Authentication is part of the HTTP 1.1 specification but it has been introduced earlier. When using the original digest scheme it also works with HTTP 1.0.

6.2.4 Forms Authentication

Web applications can use application layer authentication to validate user credentials. This is most often implemented using an HTML page to ask the user for username and password. Afterwards the information is posted to the web server. Application designers have to be aware that this data is posted in clear text. Therefore SSL should be used, at least for the login process.

This type of authentication is integrated into ASP.NET with the *FormsAuthenticationModule* class. ASP.NET uses a cookie to send the session identifier, either as an HTTP header or as part of the URL.

6.2.5 Integrated Windows authentication

This authentication types are proprietary and can only be used with the combination of Internet Information Services and Internet Explorer.

NTLM and Kerberos

NTLM was the method of authentication in Internet Information Server before Windows 2000. With the approach of Windows 2000 Kerberos became the protocol for Integrated Security. Clients that are not able to communicate with the web server via the Kerberos protocol will use NTLM.

The main advantage of Kerberos is that both the server and the client are checked. According to [Howard, LeBlanc 2000, p. 120] with NTLM the client might talk to a server that is not valid because the server is not authenticated.


Microsoft Passport

In the next version of Windows Server (Windows Server 2003) the web server from Microsoft has the possibility to use Microsoft Passport for the authentication of users.

6.2.6 Digital Certificates

“Both SSL and TLS can provide client, server and mutual entity authentication.”

[OWASP GUIDE 2002, p. 18]

Web servers can communicate encrypted with clients. This means that certificates are used to identify the entities. In most cases only the identity of the server is guaranteed but mutual entity authentication is also possible. If a secure connection is established a closed key symbol () appears in the browser.

Digital Certificates should be used together with Basic Authentication to provide secure transmission of user credentials when Integrated Windows Authentication is not possible. This is typically the case with non Intranet solutions. How Digital Certificates work in detail is not part of this work.

The trade-off of Digital Certificates is that they slow down the performance of the web servers because the server has to validate the certificates and encrypt/decrypt the data stream.

6.3 Authorization

“Authorization is the act of checking to see if a user has the proper permission to access a particular file or perform a particular action, assuming that user has successfully authenticated himself.”

[OWASP GUIDE 2002, p. 27]

The OWASP identifies the following access control mechanisms:

- Discretionary Access Control
- Mandatory Access Control
- Role Based Access Control

“Authorization is determined by performing an access check to see whether the authenticated principal has access to the resource being requested.”

[Howard, LeBlanc 2002, p. 15]

In a web application each web page should check if the user accessing the web page has the appropriate permissions. It is not a good to have a secret administration page where only the administrator knows the URL. People can guess these URLs. Or for instance a data file that is used by the application that contains additional sensitive information that is not provided by the web application could be downloaded by just accessing the URL of the database file. This would be “Security by Obscurity” as explained in the previous chapter.

In ASP.NET authorisation can be done with the use of the *URLAuthorizationModule*, the *FileAuthorizationModule* and *Role checks* can be implemented. The URL Authorization can be configured in the *web.config* file.

For instance the following example will allow everyone from the Managers Group of MyDomain access to the application.

```
<authorization>
  <allow roles="MyDomain\Managers">
</authorization>
```

To deny access to specific files in the virtual root appropriate ACL (Access Control List) have to be defined. Explicit role checks are implemented with the use of the *Principal* objects. These objects (Generic and Windows principals) allow checking if the calling user has the appropriate permissions (role membership) to access the web page. This is done with the *IsInRole* method. For additional information about how this can be implemented see [Meier, Mackman, Vasireddy, Dunner 2002]

6.4 Auditing

According to [Howard, Levy, Waymire 2002, p. 276] the process of auditing has two main purposes:

- Provides the ability to determine whether, how, and when you were attacked and what was attacked
- Provides the ability to troubleshoot security issues

The two points show auditing from a security perspective, auditing also has other purposes, for instance traffic analysis and statistics.

Generally logging should include information such as time of event, initiating process or owner of process and a detailed description of the event.²⁰ In my opinion sensitive data should not be logged (for instance credit card number or user information like passwords). The log files should only be accessible by administrators. Auditing to many events is contra productive. Analyzing log files and event logs is a very resource intensive task.

The process of defining what to audit might be very crucial to the overall security process – by auditing the correct events malfunctioning code, which accesses resources that it should not be allowed can be determined. Auditing everything might lead to an unmanageable system.

Web server log files should be analysed with tools to get a better view on system usage. There is a tool available called Advanced Web Statistics (awstats) that allows analysing log files of IIS and Apache.²¹

²⁰ see [OWASP PLAN 2003, Chapter 9 Event Logging]

²¹ <http://awstats.sourceforge.net>

6.5 Privacy

There are technologies that support privacy:

- Secure Socket Layer / Transport Layer Security
- Internet Protocol Security (IPSec)

Microsoft announced a new system that will allow software vendors and end users to enhance privacy, integrity and data security.

“We are working on a new hardware/software architecture for the Windows PC platform, code-named "Palladium," which will significantly enhance users' system integrity, privacy and data security.”

[Gates, 2002]

Recently this system has been renamed to “Next-Generation Secure Computing Base”. The system will provide applications an isolated part in the memory of the PC. This is guaranteed through hardware components which are currently not part of computer systems. This system is developed by the Trusted Computing Platform Alliance.²²

²² <http://www.trustedcomputing.org>

Communal Web Browsers

It is very common that web applications are browsed from different systems and that users might share computers, for instance in an Internet Café.

[OWASP GUIDE 2002, Chapter 12] suggests to provide a warning in the applications which includes:

- *The possibility of pages being retained in the browser cache*
- *A recommendation to log out and close the browser to kill session cookies*
- *The fact that temp files may still remain*
- *The fact that proxy servers and other LAN users may be able to intercept traffic*

Another aspect of the communal web browser is that the user of such a system can stay anonymous. This fact makes these systems an ideal place for starting attacks.

It is possible to set up these communal systems in a way that there are no passwords stored and the browser history is disabled. This can be accomplished with the use of Internet Explorer Administration Kit, Active Directory and Group Policies.

6.6 Integrity

“When used in a security context, integrity refers to staying the same”.

[Viega, McGraw 2002, p. 23]

This means that data needs to be protected from being altered. The data could be maliciously or accidental altered. Typical integrity technology includes Secure Socket Layer, Transport Layer Security or the IPSec protocol.

6.7 Availability

This means that users who are legitimated to a system can access it, whenever they need it. There are hard- and software technologies that support this design goal. For instance this includes hard- and software load balancing for distributing client requests upon a web farm.

6.8 Nonrepudiation

X.813 Information Technology – Open Systems Interconnection – Security frameworks in open systems defines Nonrepudiation²³ as:

- Nonrepudiation with proof of origin, which is used to counter false denial by a sender that the data or its contents has been sent.
- Nonrepudiation with proof of delivery, which is used to counter false denial by a recipient that the data or its context has been received.

“Nonrepudiation guarantees that the message sender is the same as the creator of the message.”

[Samtani 2002]

To guarantee that a message has been sent by one specific entity Digital Certificates should be used to digitally sign the message.

²³ see <http://www.itu.int/rec/recommendation.asp?type=items&lang=e&parent=T-REC-X.813-199610-I>

7 DIFFERENT DESIGN APPROACHES

7.1 About the programming language

As shown previously in this work, most security vulnerabilities are buffer overflows - more than 50%. The reason for this is because most applications are written in C and C++. These languages might not always be the best choice because they do not provide proper bounds checking (neither on the stack nor on the heap). This is the reason why I do not recommend to use these languages in the first place.

The reason why C is not the ideal programming language is clear. C was created to simplify operating system development. Before the advent of C, programmers had to use Assembler to accomplish this task. In the early 90s C and also C++ were rapidly adopted by students as their primary language. There are libraries in C (for instance the `string.h`) that provide a lot of non-secure functions that when used can lead to buffer overflows.

Examples for such functions are [Viega, McGraw 2002, p. 142]:

- strcpy
- strcat
- sprintf
- scanf
- sscanf
- fscanf
- vfscanf
- vsprintf
- vscanf
- vsscanf
- streadd
- strecpy
- strtrns

All these functions are not aware of buffer overflows. When arbitrary input is passed to these functions a buffer overflow can be exploited.

There are situations where these languages have to be used because of performance issues or simply because it is not possible to solve a problem in another programming language. However if these languages are used tools like the StackGuard²⁴ or AppVerifier²⁵ have to be used to check for buffer overflows.

For applications that implement business processes, languages like C# or Java are the better choice. Those applications are running in a managed execution environment that provides type safety and a garbage collection. The developer can focus on the business issue that has to be solved rather than on technical issues like freeing allocated memory.

7.2 Intranet

Many attacks are coming from within and organisation. It seems that most intranet systems are not good prepared for these kind of attacks and/or that management is not aware of that. In Intranet scenarios it is possible to create much richer applications because the infrastructure is better known.

²⁴ <http://www.immunix.org/stackguard.html>

²⁵ <http://www.microsoft.com/windowsxp/appexperience/appverifier.asp>

In an Intranet scenario it is often possible to use Integrated Windows authentication at the web server which provides better security than the other authentication types [Howard, Levy, Waymire 2000, p. 116]. The downside of the Integrated Windows authentication of Internet Information Services is that all clients who want to access the system need Internet Explorer. To provide privacy and integrity those systems should use SSL/TSL and IPsec where appropriate.

7.3 Internet

The design of an Internet application is often more rudimentary. Because services have to be provided to a broad community, it is necessary to use the standard protocols defined for Internet communication and rendering. HTML 3.2 and Java Script are supported by most browsers (Opera, Mozilla, Internet Explorer). Active content like Java applets or ActiveX controls should be avoided because of security and usability reasons.

To provide privacy and integrity SSL/TSL should be used where appropriate (for instance during logon). In addition digital certificates to sign messages should also be implemented where appropriate, for instance for e-Government solutions.

7.4 Web Services

Companies provide and consume services to other business partners. To do business with partners a contract has to be defined between the two or possible more business partners on how to communicate and interchange information. Web Services bring the solution to these problems. They allow communication between different platforms using standardised protocols like HTTP, SOAP and XML. Different applications from different vendors can “talk” to each other [W3C WG 2002].

Integrating businesses of different companies is one domain of Web Services. Because Internet communication is normally not encrypted most companies use web services together with “Basic Authentication” and wire encrypted protocols like SSL (Secure Socket Layer) to interchange information. Recently OASIS has proposed draft specifications for WS-Security, WS-Routing and others to enable secure end-point to end-point communication for web services. This is achieved with the use of digital certificates. [OASIS 2003]

IBM has created a SDK for their Web Sphere server and Microsoft is shipping the Web Service Enhancements that provides those security functionalities.

New security concerns arise with the use of Web Services. Therefore a lot of new standards are proposed to the OASIS that allow secure end to endpoint communication of web services. [OASIS 2002]

The security principles discussed earlier in this document (specially input and output validation) have to be applied to web services.

8 CONCLUSION

Security in software applications has become a very important topic over the last years. This is because today computer systems are highly interconnected and the Internet has gone main stream.

Over the last years the number of vulnerabilities in software applications has grown tremendously. The main vulnerability in software applications is the “buffer overflow”. “Buffer overflows” are very important to take into consideration in web applications. Many web applications call library functions which are written in C/C++ and those library functions could be vulnerable to a “buffer overflow”. If the entered information of the web application is not properly handled and the data is directly given to the library function (without validation) the applications might be vulnerable.

This leads us directly to another class of vulnerabilities the “Code Injections”. “Code Injections” happen when an attacker enters code into the web application that gets directly executed by the system. For instance SQL code is entered at the logon screen of a web application that executes malicious code against the database system.

Another vulnerability that web applications have to be aware of is “Cross Site Scripting”. “Cross Site Scripting” can be used to hijack a user session in a web application. This is done by stealing the session identifier (typically a cookie) that web applications have to use because of the stateless nature of the HTTP protocol.

There are different vulnerabilities that exist in software applications. Therefore a detailed literature research has been made on software security principles. This work shows how these principles can be applied to web applications.

The literature research has finally come up with 17 security principles for software development. Applying some principles might not be possible without violating other principles. This clearly shows that the task of secure development is more than just the use of those principles. For instance the principle of “Keep it simple” clearly states that software components should be easy to understand and that unnecessary code should be prevented. But the principle of “Defence in Depth” states that the application should have different layers and redundancy built in because if the one layer fails another catches the failure. This principle is clearly against the principle of “Keep it Simple”. This makes it hard for developers to decide which is more important.

The principle “Check at the gate” states to do authorisation and authentication check at one point – this is suggested because of performance issues that might occur when handling the user credential on to other components. The principle of “Compartmentalization” and others state that every component should be secure on its own and that different layers of security should be implemented.

The principle “Use secure defaults” is wrong because software products are often not secure by default. Generally it is always better to de-install components and services which are not used to provide fewer surfaces for attack.

There will never be a perfect list of software security principles. Every security principles list that was researched has leaks and some principles conflict with each other. Following the given principles alone will not lead to a 100% secure web application but many troubles can be avoided. The principles point out common pitfalls that are made by developers. Software developers should be aware of those principles to avoid the main issues.

The most important principle is input validation. Not properly validating the user input can lead to “Buffer Overflows”, “Cross Site Scripting” vulnerabilities and “Code Injections”. No other principle has as much momentum to prevent so many issues at once.

The work shows how to apply regular expressions to prevent invalid information from being processed by the web application. It is important to validate the input on the server side. Validating the input only on the client side is not appropriate.

In addition to the research on vulnerabilities and security principles, the design considerations that have to be made when building enterprise web applications are figured out. The main elements of a secure design are shown. It is important to use the correct features of those elements to provide security.

For Intranet web applications that run on a Windows environment, “Integrated Windows Authentication” should be used at the web server. The benefits of this are that the password is not transferred over the wire and the Intranet users do not need to enter their credentials again for different web applications.

People do not need to remember different username/password combinations. This leads to more secure Intranet solutions.

The main drawback of “Integrated Windows Authentication” is that it can only be used with Internet Information Services and Internet Explorer. But in the Intranet the enterprise can roll out these systems if not already present.

To provide integrity and privacy, digital certificates should be used in web applications where appropriate.

Finally the work shows that the used programming language has impact to the security of the product. The work shows that C/C++ have library functions which are insecure and should not be used at all. Applications written in C/C++, that use these functions (e.g. strcpy or sprintf) are in most cases not aware of buffer overflows. Therefore C/C++ should not be used in the first place.

For applications that implement business processes, languages like C# or Java are a better choice than C/C++ because they provide a runtime environment that handles memory management.

9 CRITICAL EVALUATION

This chapter analyses the work in a critical way and reflects strengths and weaknesses of it.

The work is based on an extensive literature research and software security principles are defined. In addition sample applications which are easy to understand are implemented to demonstrate the main security vulnerabilities that exist in web applications.

The project focuses on the vulnerabilities and how they can be prevented by applying the researched security principles. The design elements of a web based enterprise solution are defined. The work does not concentrate on this subject as extensive as on the security principles.

The project concentrates on Microsoft technologies but the researched issues and principles can be adapted to other systems easily - for instance using Regular Expressions to validate input. The buffer overflow example is written in C under FreeBSD which gives the work also a UNIX perspective.

The distinction between software security and security in general is hard to find. In addition the distinction between software security in general and software security in web applications is difficult to find.

Discussing this document with different people showed that the security principle “Use secure defaults” can be misunderstood and should have been worked out more clearly.

10 BIBLIOGRAPHY

[Boehm 1988]

Boehm, B. (1988). *A spiral model for software development and enhancement*. IEEE Computer.

<http://computer.org/computer/co1988/r5061abs.htm>

[Berry 2002]

Berry Wayne (2003). *Innovations in Internet Information Services Let You Tightly Guard Secure Data and Server Processes*.

<http://msdn.microsoft.com/msdnmag/issues/02/09/SecurityinIS60/default.aspx>

[Brown 2000]

Brown K. (2000). *Programming Windows Security*. Addison-Wesley.

[BS7799]

The BS7799 is not for free. It can be purchased and downloaded at:

<https://www.bspsl.com/secure/17799/cvm.cfm>

[CERT 2000]

Carnegie Mellon University. *CERT Advisory Malicious HTML Tags Embedded in Client Web Requests*. <http://www.cert.org/advisories/CA-2000-02.html>

[CERT 2003]

CERT® and CERT Coordination Center®. Carnegie Mellon University. *CERT/CC Statistics 1988-2002*. http://www.cert.org/stats/cert_stats.html

[CERT ADV 2002]

CERT® Advisory CA-2002-03 Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP). Carnegie Mellon University. <http://www.cert.org/advisories/CA-2002-03.html>

[CERT ADV 2003]

CERT® Advisory CA-2003-04 MS-SQL Server Worm. Carnegie Mellon University. <http://www.cert.org/advisories/CA-2003-04.html>

[Gates 2002]

Gates B. (2002), *Trustworthy Computing*

<http://www.microsoft.com/mscorp/execmail/2002/07-18twc-print.asp>

[Hanson, van Velzen, Hittel, Roculan 2002]

Security Focus ARIS Top Ten 2001

Threats – Patches and Recommendations to Protect Your Enterprise

http://www.securityfocus.com/corporate/research/top10attacks_2001.pdf

[Howard, LeBlanc 2001]

M. Howard, D. LeBlanc (2001). *Writing Secure Code, Practical strategies and proven techniques for building secure applications in a networked world*. Microsoft Press.

[Howard, Levy, Waymire 2000]

M. Howard, M. Levy, R. Waymire (2000). *Designing Secure Web-Based Applications for Microsoft Windows 2000. Discover how to build secure Web-based solutions with Microsoft Windows 2000, Internet Explorer, Internet Information Services, SQL Server, and COM+.* Microsoft Press.

[Jones, Malcolm, Mackman, Jezierski 2002]

K. Jones, G. Malcom, A. Mackman, E. Jezierski (2002). *Exception Management Application Block Overview*. Microsoft Corporation.

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/emab-rm.asp>

[Meier, Mackman, Vasireddy, Dunner 2002]

J.D. Meier, A. Mackman, M. Dunner, S. Vasireddy (2002). Microsoft Corporation. *Building Secure ASP.NET Applications*. SecNet.pdf. Internet Download.

<http://www.microsoft.com/downloads/release.asp?ReleaseID=44047>

[MS00-078]

Security Bulletin MS00-078 (2000).

Patch Available for 'Web Server Folder Traversal' Vulnerability.

<http://www.microsoft.com/technet/security/bulletin/ms00-057.asp>.

[MSF 2002]

Microsoft Solution Framework

<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/tandp/innsol/msfrl/default.asp>

[OASIS 2002]

OASIS NEWS Article (2002)

<http://www.oasis-open.org/committees/wss/documents/WSS-Core-09-0126.pdf>

[OASIS 2003]

P: Hallam-Baker (VeriSign), C. Kaler (Microsoft), R. Monzillo (Sun), A.

Nadalin (IBM) (2003)

The most recent draft during working on this document was:

http://www.oasis-open.org/news/oasis_news_07_23_02.php

The complete index of drafts can be found:

<http://www.oasis-open.org/committees/download.php/1204/doc-index.html>

[OWASP GUIDE 2002]

Curphey M., Endler D., Hau W., Taylor S., Smith T., Russel L., McKenna G., Parke R., McLaughlin K., Tranter N., Klien A., Groves D., By-Gad I., Huseby S., Eizner M., McNamara R. (2002). *A Guide to Building Secure Web Applications*.

<http://unc.dl.sourceforge.net/sourceforge/owasp/OWASPGuideV1.1.1.pdf>

[OWASP PLAN 2003]

The Open Web Application Security Project Plan for 2003. (2003).

<http://www.owasp.org>

[Pfleeger 2001]

Pfleeger; S. L. (2001). *Software Engineering – Theory and Practice, 2nd Edition*. Prentice-Hall, Inc.

[RFC 1321]

Rivest R., MIT Laboratory for Computer Science and RSA Data Security, Inc. (1992). *The MD5 Message-Digest Algorithm*. Network Working Group.

<http://www.ietf.org/rfc/rfc1321.txt>

[RFC 1945]

T. Berners-Lee, R. Fielding, H. Frystyk (1996). *Hypertext Transfer Protocol - HTTP/1.0*. Network Working Group.

<http://www.ietf.org/rfc/rfc1945.txt>

[RFC 2068]

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee (1997). *Hypertext Transfer Protocol -- HTTP/1.1*. Network Working Group.

<http://www.ietf.org/rfc/rfc2068.txt>

[RFC 2518]

Y. Goland, E. Whitehead, A. Faizi, S. Carter, D. Jensen (1999). *HTTP Extensions for Distributed Authoring – WEBDAV*. Network Working Group.

<http://www.ietf.org/rfc/rfc2518.txt>

[Romanosky 2002]

Romanosky, S (4th, June 2002). *Enterprise Security Patterns*

<http://www.romanosky.net/papers/securitypatterns/EnterpriseSecurityPatterns.pdf>

[Saltzer 1975]

J. H. Saltzer, M. D. Schroeder *The protection of information in computer systems*. IEEE.

<http://denali.cs.washington.edu/relwork/papers/saltzer.html>

[Samtani 2002]

Samtani, G. (2002). *Top 10 Web service security requirements*

<http://builder.com.com/article.jhtml?id=u00320020610GXS01.htm>

[SECURITYFOCUS 2002]

Top Ten Vulnerabilities of 1st Quarter 2002

http://www.securityfocus.com/corporate/research/top10vulns_q1_2002.shtml

[Sommerville 2001]

Sommerville, I. (2001). *Software Engineering, 6th Edition*. Addison-Wesley

[Spafford 1991]

Spafford E. H. (1991). *The Internet Worm Incident. Technical Report CSD-TR-933*. Department of Computer Sciences, Purdue University.

<http://www.cerias.purdue.edu/homes/spaf/tech-reps/933.pdf>

[Spitzner 2002]

Spitzner, L. (2002). *Know your enemy*. Revealing the security tools, tactics and motives of the blackhat community. Addison Wesley

[SYMANTEC 2000]

<http://securityresponse1.symantec.com/sarc/sarc.nsf/html/w32.nimda.a@mm.html>

[Tzu 500 BC]

Tzu S., J. Clavell. *The Art of War*. 500 BC (1983). Delacorte Press.

[Viega, McGraw 2002]

J. Viega, G. McGraw (2002). *Building Secure Software – How to avoid security problems the right way*. Addison-Wesley.

[W3CWG 2002]

<http://www.w3.org/2002/ws/arch/2/06/wd-wsa-arch-20020605.html#IDATEYE0B>

[Wong 2000]

Wong, C. (2000), *HTTP Pocket Reference*. O'Reilly & Associates, Inc.

11 APPENDIX

This chapter includes Final Year Project Proposal, Meeting Protocols, Project Plan and the Interim Report.

11.1 Proposal

| | |
|---|--------------|
| SCHOOL OF MATHS & COMPUTING FINAL YEAR PROJECT PROPOSAL | |
| NAME: Johann REHBERGER | |
| PROJECT TITLE: Web Application Security Principles – Designing secure web based enterprise solutions | |
| MAIN SUPERVISOR: Johann PREISSEL | |
| SECOND SUPERVISOR: | |
| TYPE OF PROJECT: Research, Development | |
| AIMS & OBJECTIVES OF PROJECT: <ul style="list-style-type: none">• to learn about security vulnerabilities concerning web applications• to show the principles of secure development• to show the elements of a secure design• find out how secure web applications should be designed to meet end users needs (security vs. usability)• develop a web application that allows to demonstrate researched issues | |
| EXPECTED OUTCOMES OR DELIVERABLES: <p>A report containing:</p> <ul style="list-style-type: none">• a list of security vulnerabilities concerning web applications• principles of secure web development• a description of the elements of a secure design• a comparison about different design approaches• an evaluation about security vs. usability <p>Sample Applications</p> | |
| METHODOLOGY: Research, Development | |
| HARDWARE & SOFTWARE REQUIREMENTS: All hardware and software requirements are provided by the author. | |
| LITERATURE & RESEARCH MATERIALS REQUIRED: <p>Writing Secure Code 2nd Edition, Michael Howard, David LeBlanc, MS Press HTTP Pocket Reference, Clinton Wong, 2000, O'Reilly & Associates, Inc. IETF, RFC's Journals, Magazines and Online Platforms like slashdot.org, securityfocus.com</p> | |
| SIGNATURE: | DATE: |
| SUPERVISOR SIGNATURE: | DATE: |

11.2 Meeting Protocols

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 03.10.2002

Number: 1

| |
|--|
| <p>Work Done Since Last Meeting</p> <p><i>Proposal handed in and approved by Mr. Lorenz. Mr Preissl agreed to supervise it.</i></p> |
| <p>Problems / Suggested Solutions</p> <p><i>Discussion of the content/proposal. Explained my intention for this FYP to Mr. Preissl. Mr Preissl suggested that also Web Services should be a part of the report.</i></p> |
| <p>People Met / Contacts</p> <p>-</p> |
| <p>Work During Next Period</p> <p><i>Work out the report structure in more detail. Research on vulnerabilities and security principles. Find literature that concentrates on these topics. Also a project plan will be worked out during the next weeks.</i></p> |
| <p>Tutor's Comments</p> <p>-</p> |

Signed: _____
Johann Preissl

Next Meeting

Date 10.10.2002

Time 18:00

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 10.10.2002

Number: 2

| |
|--|
| <p>Work Done Since Last Meeting</p> <p><i>I have created a Microsoft SharePoint team site at rehberger.sharepoint.bcentral.com/fypjr. There I will place all the meeting protocols and documents for discussion. This helps me to manage the documents because I am often at different locations using different PCs. Report structure has been defined in more detail.</i></p> |
| <p>Problems / Suggested Solutions</p> <p><i>Discussion of the content. Showed and explained Mr. Preissl my project plan draft - it will be worked out in detail during the next weeks.</i></p> |
| <p>People Met / Contacts</p> <p>-</p> |
| <p>Work During Next Period</p> <p><i>Work out the report structure in more detail. Research on security principles and design issues. Find literature that concentrates on these topics.</i></p> |
| <p>Tutor's Comments</p> <p>-</p> |

Signed: _____
Johann Preissl

Next Meeting tbd.

Date

Time

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 09.12.2002

Number: 3

| |
|--|
| Work Done Since Last Meeting <i>Finished Interim Report.</i> |
| Problems / Suggested Solutions <i>Presentation of the Interim Report. Discussion. Layout of the FYP. 2 lines spaced report.?</i> |
| People Met / Contacts - |
| Work During Next Period <i>Hand in the report until 20.12.2002 to Mr. Preissl. In addition also Mr. Lorenz wants a copy of the Interim Report, Define a meeting in 2003. A first draft of the report will be shown to Mr. Preissl. Read literature.</i> |
| Tutor's Comments - |

Signed: _____
Johann Preissl

Next Meeting tbd.

Date

Time

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 20.01.2003

Number: 4

| |
|---|
| Work Done Since Last Meeting <i>Interim Report was handed-in. Created a first version of the FYP.</i> |
| Problems / Suggested Solutions <i>Discuss the content of the FYP and the literature review.</i> |
| People Met / Contacts - |
| Work During Next Period <i>Continue writing Report. Generate sample applications to demonstrate the researched knowledge..</i> |
| Tutor's Comments - |

Signed: _____
Johann Preissl

Next Meeting

Date 27.02.2003

Time 19:00

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 27.02.2003

Number: 5

| |
|---|
| <p>Work Done Since Last Meeting</p> <p><i>Report writing and implementation of sample application for buffer overflow.</i></p> |
| <p>Problems / Suggested Solutions</p> <p><i>Referencing. Discussed the different methods for referencing and agreed to use the Harvard style and I will put the page number in the brackets. e.g. [AUTHOR 2003, p. 10] Discuss the content and the literature review.</i></p> |
| <p>People Met / Contacts</p> <p>-</p> |
| <p>Work During Next Period</p> <p><i>Continue writing Report and sample applications. Will deliver the next sample application until next meeting.</i></p> |
| <p>Tutor's Comments</p> <p>-</p> |

Signed: _____
Johann Preissl

Next Meeting

Date 13.02.2003

Time 19:00

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 13.02.2003

Number: 6

| |
|--|
| <p>Work Done Since Last Meeting</p> <p><i>Report writing and implementation of sample application to demonstrate SQL Injection.</i></p> |
| <p>Problems / Suggested Solutions</p> <p><i>Should the sample code be part of the appendix? I suggest to put the applications just on the attended compact disc. The important code snippets are part of the report content and the complete solution can then be found on the CD.</i></p> |
| <p>People Met / Contacts</p> <p>-</p> |
| <p>Work During Next Period</p> <p><i>Continue writing Report. and sample applications.</i></p> |
| <p>Tutor's Comments</p> <p>-</p> |

Signed: _____
Johann Preissl

Next Meeting

Date 27.02.2003

Time 19:00

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 10.03.2003

Number: 7

| |
|---|
| Work Done Since Last Meeting <i>Report writing and implementation of sample application.</i> |
| Problems / Suggested Solutions <i>When will the VIVA be? What is the exact date of the hand-in? Have problems with writing the Critical Evaluation. Discuss the content and the literature review.</i> |
| People Met / Contacts - |
| Work During Next Period <i>Continue writing Report and sample applications. Send a version of the nearly finished report to Mr. Preissl. Mr Preissl will read it completely and provide feedback.</i> |
| Tutor's Comments - |

Signed: _____
Johann Preissl

Next Meeting

Date 02.04.2003

Time 19:00

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 02.04.2003

Number: 8

| |
|---|
| Work Done Since Last Meeting <i>Report writing. Send report to Mr. Preissl.</i> |
| Problems / Suggested Solutions <i>Should the protocols be signed? Discuss the content and the literature review.</i> |
| People Met / Contacts - |
| Work During Next Period <i>Continue implementing discussed changes. Will bring a hard copy of the meeting protocols so that Mr. Preissl can sign them.</i> |
| Tutor's Comments - |

Signed: _____
Johann Preissl

Next Meeting

Date 02.04.2003

Time 19:00

School of Computing and Technology

University of Derby

FINAL YEAR PROJECT PROGRESS REPORT

Student Name: **REHBERGER Johann**

Date 07.04.2003

Number: 9

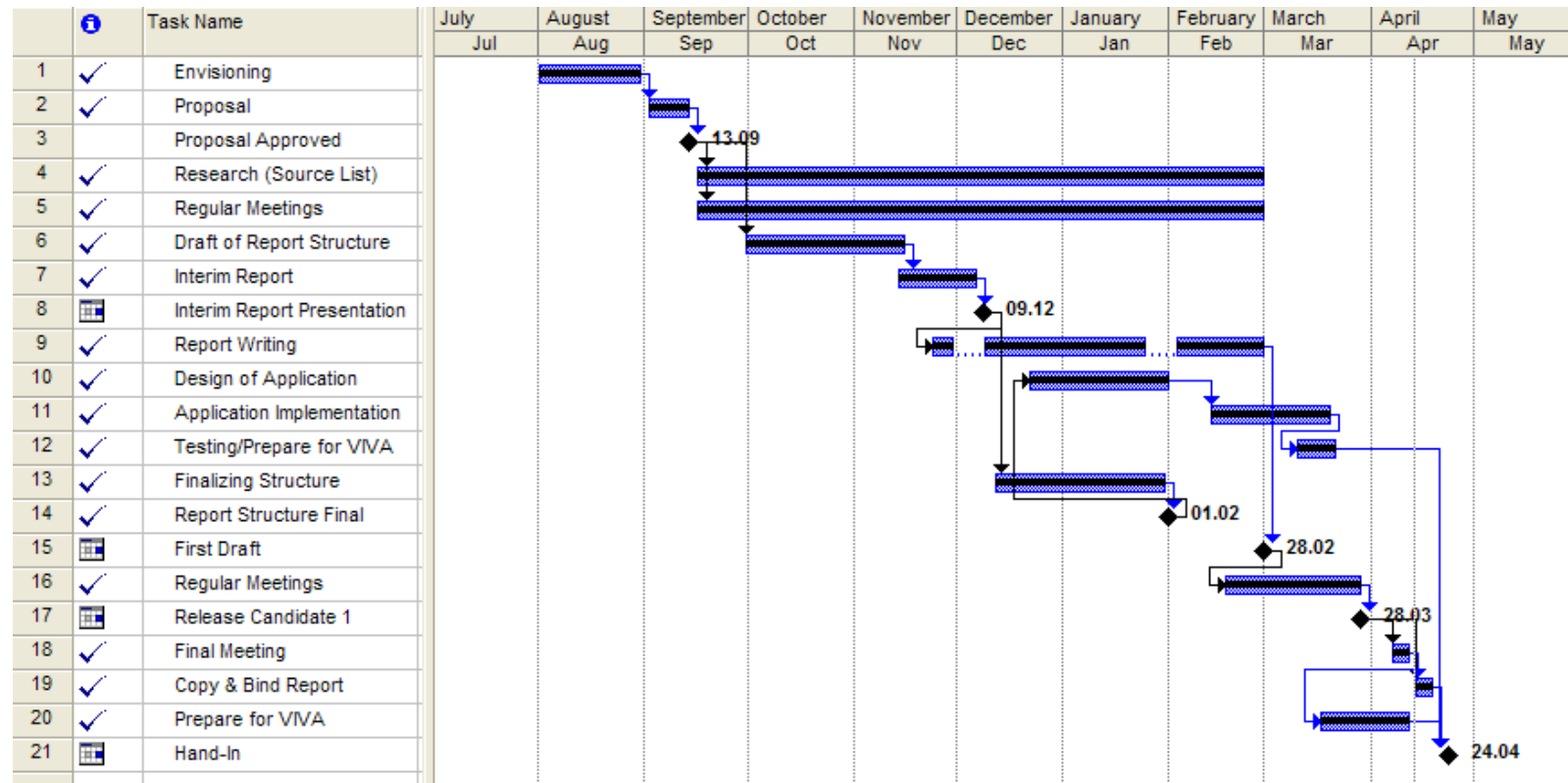
| |
|---|
| Work Done Since Last Meeting <i>Printed out the meeting protocol for signing.</i> |
| Problems / Suggested Solutions <i>Hand-in questions? When, how?</i> |
| People Met / Contacts - |
| Work During Next Period <i>Implement discussed changes. Finish Appendix documents (meeting protocols, project plan, proposal will be included)</i> |
| Tutor's Comments - |

Signed: _____
Johann Preissl

Next Meeting hand-in
Date

Time

11.3 Project Plan



11.4 Interim Report

On the next pages the Interim Report is attached. It is not included into this document electronically because it has been handed in as an own document in December 2002.