Secure Programming questions

BV	:	ITsec Security Services bv
To	:	software engineering students HvA
From	:	Jannes Smitskamp (ITsec)
Project	:	SS-REL-HOVAM-1201
Subject	:	homework questions
Documentnr.	:	SS-REL-HOVAM-1201-11.301

Discussed Sins

Lesson	Sin	Sin
1	1	SQL Injection
1	24	Trusting Network Name Resolution
1	12	Information Leakage
1	5	Buffer Overruns

Introduction

This document contains a number of questions about the Sins discussed during the lectures. The purpose of those questions is to apply this knowledge on the basis of examples and code snippets. Answering the questions should lead to a better understanding of the learned material and apply this to real scenarios.

Code samples are included in a frame and can be written in one of the following languages:

- Java/JSP;
- .NET (C#, VB.NET, ASP.NET, etc) ;
- C/C++ ;
- PHP.

Intensive understanding of these programming languages is not necessary to answer the questions. In order to focus on a specific problem some parts of the code have been omitted and have been replaced by "[...]".

Each line in the source code examples has a line number. These line numbers are references only and are not present in the "real" code.

Question 1

a) An application may disclose information in several ways. In some cases this may affect the security of this application. One way in which an application might reveal information is via a so-called "side channel". Give an example of such an information leak..

When an attacker is able to perform SQL-injection on a website or application, he or she can do this unnoticed(depending on security measures) when using a Timing Channel. The idea is that the attacker uses a query that (for example) checks if a certain table exists. If the given table does exist, the remaining part of the query will have a delay in it. The attacker waits for a response from the (web)application and if it matches the given delay, the attacker then knows the given table exists.

b) Applications have to deal with errors. In some cases, an error will result in a situation where the application cannot continue. In this case, an error message has to be displayed.

Describe the save solution to inform a user that an error has occurred, without giving too much information and without losing any relevant information about the error. Note that this concerns a web application.

If you are not completely sure about who the end user is, then the most secure solution is to display a message to the usage which states that an error has occurred and he or she should contact the administrator of the (web)application. At the same time, in the back-end of the (web)application the error should be logged so that only the administrator can view what has happened and possibly resolve the problem.

Question 2

a) Briefly describe how a Man-in-the-Middle attack works.

During a MITM attack a malicious machine(proxy) intercepts the network traffic going from the client to the server and the other way around. In case the traffic is not encrypted or protected in an other way, then the malicious machine is also able to alter the contents of the network traffic, without anybody noticing it. This is, of course, a major security breach.

b) How does an end user know that his browser is really connected to the right website?

If an website is being connected to by using SSL, then the end user can take a look at the certificate of the website. The browser will show the user if the certificate is valid. These certificates are being given out by certain companies that are whitelisted and therefore(most of the time) trustworthy. As long as these certificates are valid, the user is connected to the right website.

c) Besides a man-in-the-middle attack between a browser and web server, a MitM attack can also take place between other entities. Describe another interesting scenario where a MitM attack can take place.

A MITM attack can also be in the form of a spoofed DNS server. If an attacker manages to retrieve the IP address of the DNS server the end user is using, the attacker might be able to send out IP addresses that do not correspond to the given domain name. This allows the hacker to forward the end user to a malicious website that for example resembles a login form that is familiar to the end user, but in reality is not the real deal.

Question 3

Using stored procedures is often recommended mitigate SQL injection attacks. However, this is not a valid solution, see the following example:

```
CREATE PROCEDURE dbo.doQuery(@id nchar(128))
AS
  DECLARE @query nchar(256)
  SELECT @query = 'SELECT ccnum FROM cust WHERE id =
''' + @id + ''''
  EXEC @query
RETURN
```

a) Explain why this example is vulnerable to SQL injection.

The attacker is in total control of the id variable. He or she can adjust the content of the variable to, in essence, get rid of the where clause. This can, for example, be done by setting the id to: "1 OR 2>1 --". This lets the where clause always return true and comments the remaining part of the query out. In this particular query, a list of all credit card numbers is returned.

b) Provide a clear example of how this vulnerability can be exploited by an attacker.

See 3.a.

Question 4

In the following example a programmer tries to safeguard against SQL injection by using the `mysql_real_escape_string()` function.

```
<?php

 $db = mysql_connect("localhost", "root", "secret");
   mysql_select_db("CreditCards", $db);

   $query = "SELECT CCNumber FROM UserData WHERE id=";
   $query .= mysql_real_escape_string($_GET["id"]) ;

   $result = mysql_query($query, $db);

   if ($result)
   {
     // do stuff with result
   }
   else
   {
    echo "SQL error when using the following query : "
    echo $query ;
   }
?>
```

According to the PHP reference manual this function does the following:

mysql_real_escape_string() calls MySQL's library function mysql_real_escape_string, which prepends backslashes to the following characters: \x00, \n, \r, \, ', " and \x1a. This function must always be used to make data safe before sending a query to MySQL.

a) Does the escape function prevents SQL injection in this case? Why (or why not)?

The escape function does not prevent SQL injection in this case, because an attacker is still able to set the id variable to "1 OR 2>1 --", and in this case it will still return all credit card numbers.

b) What action(s), instead of escaping, could the programmer have taken to prevent SQL injection?

The programmer should have used prepared statements. In addition to that, in case the id is a number, the programmer should have checked if it is in fact a number. This should rule out SQL injection.

Copyright © 2012 ITsec Security Services bv

c) What other deadly sins, did the programmer commit?

The programmer prints out the error messages to the webpage. This is information leakage(sin 12) and in a very dangerous way. The attacker can immediately see what the query was like and change his strategy.

Question 5

The most classic form of buffer overrun is called a "stack-based buffer overrun" or "stack smashing". When an attacker attempts to exploit such a vulnerability, it will usually try to tamper with the "return address" on the stack.

a) What is a "return address" and where is it used for?

The return address is the 'address' in the memory where the application should continue after it had completed its current task.

b) Why would an attacker want to change this value?

An attacker would want to change this value if he or she wants execute his or her own code after the application has handled its current task.

To counter such attacks sometimes a so called "canary" (or "security cookie" [Microsoft]) is placed between the "return address" and the local function variables. A copy of this canary is also stored on a different place. When a function completes, the value on the stack is compared to the copy. If these values do not match then it is assumed that there a buffer overrun has occurred.

c) What action should be taken if the canary-values do not match?

The application should crash itself in order to prevent more damage being done.

Question 6

The following code is used to include linefeed characters in a given string to convert to a cariage-return + line feed (lines 19-21). The resulting string is stored in the *buf*-variable. This buffer is 256 characters wide (line 3). Initially the *ptr*-variable points to the beginning of *buf*. By increasing this variable *ptr* always points to the next free position in *buf* (lines 20, 21 & 24). In order to prevent writing beyond the buffer, it is always checked whether the value of *ptr* points at the end of *buf* (line 9). If this is the case the while loop will end.

```
01:  void bad_fn(char *input)
02:  {
03:     char buf[256], *ptr, *end, c;
04:     ptr = buf;
05:
06:     // point to last valid char in buf
07:     end = &buf[sizeof(buf)-1];
08:
09:     while(ptr != end)
10:     {
11:        c = *input++;
12:        if(!c)
13:        {
14:           return;
15:        }
16:
17:        switch(c)
18:        {
19:        case '\n':
20:           *ptr++ = '\r';
21:           *ptr++ = '\n';
22:           break;
23:        default:
24:           *ptr++ = c;
25:           break;
26:        }
27:     } // end while
28:    [...]
29:  }
```

a) despite of the above-described control, it is still possible to cause a buffer overrun. Specify the circumstances under which the overrun might occure.

In case the '\n' character is placed at index 254 of buf, then line 21 will increase the value of ptr to 256, which is larger than the value of end. This means the while loop will still continue, even though the memory address is located outside buf.

b) By making a slight change to the code, the problem can be solved. Provide a possible fix for the problem.

Change line 9 to: 'while(ptr < end)'