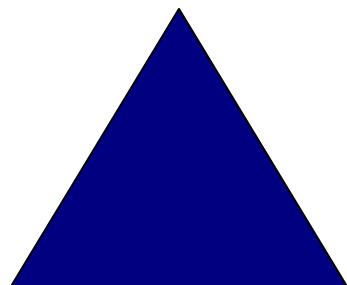


A guide to Web Application Security

Attenda



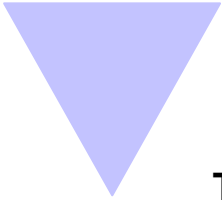
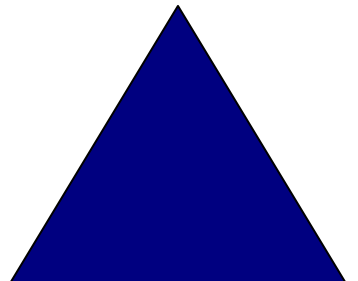
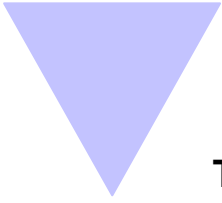


Table of Contents

The problem	3
The vulnerabilities	4
Application design / configuration.....	5
Validation and control of user input	6
Buffer overflows	6
Cross site scripting	7
SQL Injection	9
Access control.....	11
Forceful browsing	11
Path Manipulation.....	12
Conclusion – Fixing things.....	14
About Attenda Limited	15

© Attenda Limited 2006. All rights reserved.





The problem ...

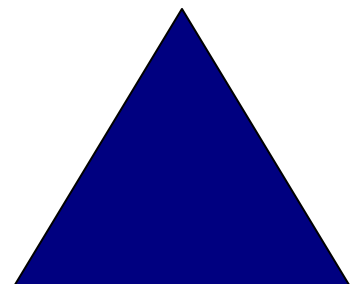
Is your Web Application vulnerable to attack? Is it being used as a platform for attacks against your own infrastructure, or other Web Applications? Are your users' cookies secure? Is your database behaving as your developers intended? Can the secured areas of your application be accessed without proper credentials?

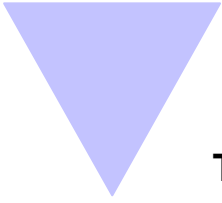
The risks and the jargon used to describe the current security phenomena of Web Application hacking are bewildering and yet often the root causes of security breaches are surprisingly mundane. Read any security web site or IT security trade magazine and you will be bound to see references to the threats, but what exactly are they and what are their impacts?

The reality of the situation is that you may have witnessed a number of security problems yourself, without realising that they are Web Application security problems. Examples include native error messages returned from the server, cookies that auto-logon users without requiring passwords, and pages that are meant to work only under SSL protection, but work in raw HTTP.

These are classic examples of obvious application security exposures that can occur within applications, but beneath the bonnet, there are hundreds of other less obvious problems. Developers often lack the knowledge of how applications are attacked, and consequently do not understand how to write code that resists attack. Most validation routines can be bypassed with character encoding methods, most login processes have flaws that can be exploited to gain access to the application or to lock out valid users. There are thousands of methods of parameter manipulation that can be targeted at your application to either create denial-of-service conditions (DoS) or unpredictable application behaviour.

This is why individuals who have responsibility for, or depend on application security are worried. In the last 12 months there has been a realisation that spending money on network security (firewalls, proxies and IDS/IPS etc), is wasted if the application is not properly secured. They are probably right to be worried; according to an analysis of our application test results, most sites are vulnerable to multiple types of abuse.





The vulnerabilities

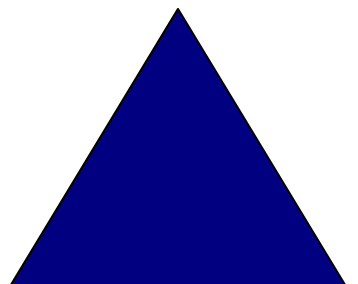
This briefing paper will outline the types of hacks that can occur and the risks they can expose you to.

Attenda has been involved in Web Application security testing for more that 2 years and when looking at the results, the risks and problems fall into the following categories:

1. Parameter Manipulation
 - a. Cross-site scripting
 - b. Parameter injection
 - c. Parameter overflow
 - d. SQL Injection
2. Path Manipulation
 - a. Path truncation
 - b. Character append/prepend
 - c. Character encoding
 - d. Canonicalisation attacks
3. Web / Application server
 - a. HTTP compliance / methods
 - b. Absolute path detection
 - c. Test and sample files
 - d. Program dumps
 - e. Stored application data
 - f. Native error messages
4. Authentication attacks
 - a. Denial-of-Service
 - b. Brute force
 - c. Logic defects and pseudo-random numbers
 - d. Browser vulnerabilities
 - e. User data security (including cookies)
 - f. Session attacks
5. Validation
 - a. Encoding weaknesses
 - b. Validation depth
 - c. Logic defects

For the purposes of this article, these could be summarised as:

- Application design / configuration
- Validation and control of user input
- Access controls





Application design / configuration

A typical application is architected to perform functions designed to deliver the end user requirements. The application design seldom considers an approach that incorporates how to resist attacks. Coupled with the fact that QA testing has an emphasis on functionality rather than security testing – evidenced by the lack of application security test tools in the QA testing portfolio, most applications go live without any real emphasis on security either at the design, development or QA phase.

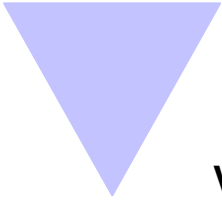
Where security defects are discovered, they are usually found late in the delivery-cycle, and fixed with quick patches that only address a very narrow portion of the total overall risk.

The configuration of the application is also very susceptible to introducing vulnerabilities – remember, an attack will make use of even the smallest defect to either gather more information about a server, or utilise the issue as a part of a more complex attack. A failure to properly handle all types of errors and catch unhandled exceptions, often gives rise to significant information leakage about the details of the application or what part of the system is vulnerable to other attacks. A reliance on simple server-based authentication methods, rather than using properly designed application authentication, means that attackers can quickly find flaws which can lead to stealing user credentials, reading unsecured traffic, or bypassing authentication completely.

To complicate matters, this problem frequently falls through an organisational responsibility gap. Most infrastructure departments will happily take responsibility for a web server, but if the vulnerability derives from content creation and delivery software, the situation may be very different. Development teams would rarely take responsibility for the security of software (other than the code they develop). The likely effect is that the software appears on a production server with standard defaults, which do not always adhere to high security levels.

Other common examples in this category include:

- Unpatched 3rd party application vulnerabilities (e.g. .Net, WebSphere, J2EE etc).
- Information disclosure – those standard error returns that can easily be used to probe the server.
- Canonicalisation attacks where directory structures become transparent to attackers
- The existence of test, sample or system dumps within the application.
- Logic defects, where attackers can find “shortcuts” to secured parts of the application, or find ways to reset passwords, or access administrative functions.

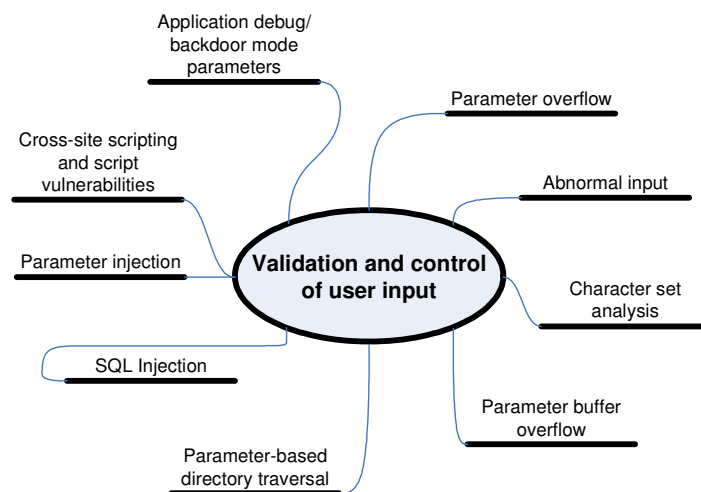


Validation and control of user input

Anyone that has spent any time programming understands the normal result of developing systems that do not validate user input correctly. Typically, the system stores or displays rubbish and then stops: GiGo “garbage in garbage out” - the old adage of system and program design is as true today as it ever was.

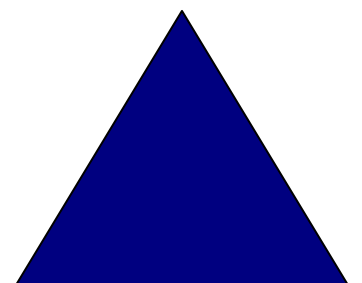
Don't be fooled into thinking that the error message on the machine is the only effect. Sometimes, these actions result in a corporate database containing invalid or past dates and other fields generally containing inconsistencies. Also, if on-line code has insufficient validation, it is more than likely your batch processing has none at all, in which case you may have created a denial of service attack, which will stop that night's batch run with arithmetic exceptions. Similarly, if you do validation in client side JavaScript, you have little reason to feel smug. The first thing any self-respecting hacker would do is bypass any client-side validation code using a local proxy and resubmit the form without any validation - then your site becomes as vulnerable as a site with no validation at all.

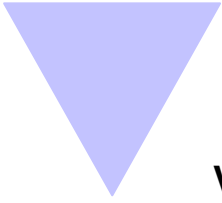
But this is all common sense, and much more about the quality of the code than “code security”. Certain omissions in validation can result in a number of much more damaging types of attacks:



Buffer overflows

Buffer overflows take many forms, but can broadly be broken down into two types; length attacks and metacode attacks. Regardless of the type of buffer attack, if there is incorrect validation applied to the parameter the attacker can either fill-up the storage reserved for the expected value, or cause user and program data spaces to try and use the same memory space. The most common result of this is that the program segment experiences a failed condition (usually an unhandled exception) and the web server issues an error message.





Validation and control of user input (*contd*)

However, sometimes with effort and persistence, security analysts discover situations where properly targeted value can overwrite special pieces of storage that control execution sequence. These stack-overflows allow attackers to manipulate various instruction pointers used by the program to remember what to do next. With clever manipulation an attacker can provide a value to a parameter that will cause execution to jump from the intended functionality of the normal program to a section of code that loads another program or command and then executes it – a dream for any hacker, being able to run a command of his choice.

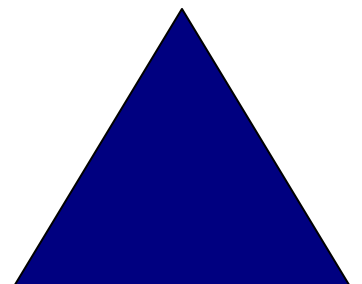
Buffer overflows used to be discovered by accident or restricted to expert assembler programmers with too much time on their hands. Today, comprehensive papers, methodologies and utilities have been published which make it well within the reach of an average script kiddie. In fact, approximately 20% of all security vulnerabilities reported to CERT are now buffer overflow based. To show how straightforward discovering these attacks has become, the basic procedure is outlined below:

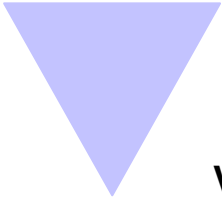
1. Select a potentially vulnerable target program. If it is a common program (either open-source or readily available), you should install it on a local system; this will make the whole process much easier by allowing a greater speed of work and allowing direct memory access.
2. Using an http request mutation engine (again freely available from your local hacker site) to repetitively pass the target program longer and longer parameters until the program suffers an abnormal termination (indication of a buffer-overflow). Use a debugging program to attach to the running program or analyse your dump (core) file to identify important areas of memory.
3. Using the addresses from the debugger and a simple bit of maths, you can craft a parameter of a particular length, containing a section of machine code. This code, known as ShellCode, can be downloaded readily from the Internet and when used in the correct position in the parameter will invoke a command shell (i.e. `/bin/sh` in Unix or `cmd.exe` in Windows).

Cross site scripting

Cross-site scripting has been known about for at least five years – and largely ignored because many security managers and developers do not understand the implications. A cross-site exploit occurs when a script command (typically JavaScript or HTML) is entered as a parameter value and submitted to the web server. If the web server replays the unaltered script to the browser, the script will be executed.

This means that an attacker can simply provide hyperlinks in email, websites, blogs etc for victims to click on, and YOUR website will return the hackers script to the innocent users. It is ignorance and a lack of appreciation of the threat that mean that security managers rationalise away any dangers of cross-site scripting as a minor inconvenience. Cross-site scripts are at the heart of all second-generation Phishing attacks. Before we inspect and shoot this particular school of thought right down in flames, let's look at an example.





Validation and control of user input (contd)

The most basic possible Cross-site scripting (known as XSS) is;

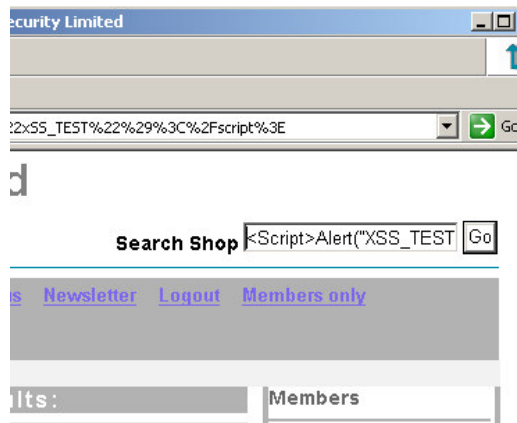
```
<SCRIPT>  
alert("xSS_TEST") </SCRIPT>
```

The <SCRIPT> and </SCRIPT> tags simple define the start and end of the script body. The real functionality is;

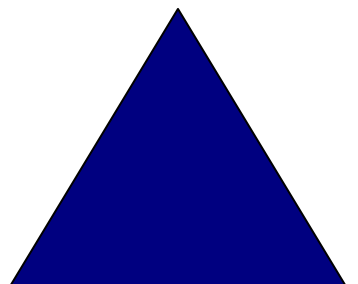
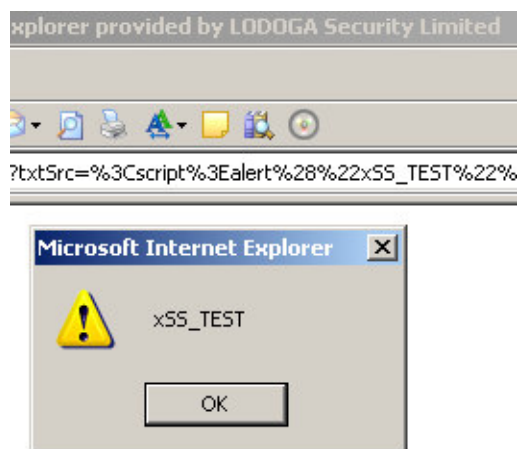
```
Alert("xSS_TEST")
```

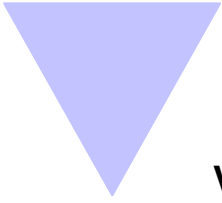
This displays a message box, "xSS_TEST" on the screen. The effect is clear and illustrated using the screen shots of a dummy Web Application, which were based on a real application.

This screen image shows the script command being entered:



This second screen shot shows the results of the script command, with the alert box requesting user action;





Validation and control of user input (contd)

Security managers that ignore the exposure run the risk of causing regulatory or even serious legal problems. With a small element of social engineering, the insecurity of the application will be exposed and damage the brand of any organisation whether or not they were aware of the exposure. Imagine if the attack was initiated from a link on a web site, search engine or an email, like the recent *Phishing* attacks on banks that conned the users into revealing their user id and passwords. If a login panel was displayed from an XSS that looked exactly like the real thing and came from the support desk of the organisation, who would question it? Especially if when the cautious user clicked the padlock at the bottom of the screen, the certificate information showed clearly that the correct organisation was being accessed. Even if you are a highly security conscious organisation and use client side certificate based mutual authentication (PKI), the XSS will still work.

If your application is highly configurable, and the attack can be planted in the cookie or database that holds the configuration options, the effects can also be devastating.

Up until now the full effect of XSS has not been felt. But it is really just around the corner.

SQL Injection

SQL injection is an attack that manipulates parameters that are used directly in SQL statements. These form fields are the normal input fields that you see on your screen when you enter your customer number, address or search phrase at your favourite web sites, so no special equipment or particular expertise is required beyond basic knowledge of the technique. Our experience leads us to believe that in the region of 60% of web sites that also use a database are susceptible in one form or other. At the very least, this allows you to augment conditions so you can read data that you would not normally be allowed to access. It is not difficult to see how this could lead to legal repercussions (i.e. breach of the Data Protection Act) or damaging publicity **simply by entering a series of normal speech marks and SQL commands at the browser.**

In more serious cases, it allows attackers to insert fraudulent records or delete any record and even run commands on the target server.

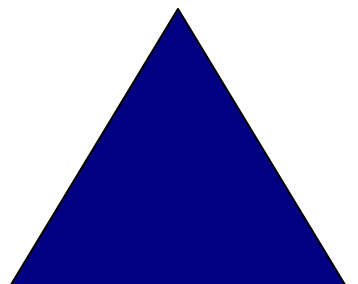
For most managers that are unaware of this vulnerability, this information is TOO much information – they simply want to check if they are vulnerable, immediately.

Imagine a code segment:

```
SelectMembers = " SELECT *  
" FROM tbl_ShopMembers " &  
" Where UserName = "" & Uname & "" and Pass = "" & Pass & ""
```

When a user Fred with a password "Newcastle" is entered, the query evaluates to

```
SelectMembers = " SELECT *  
" FROM tbl_ShopMembers " &  
" Where UserName = 'Fred' and Pass = 'Newcastle'
```





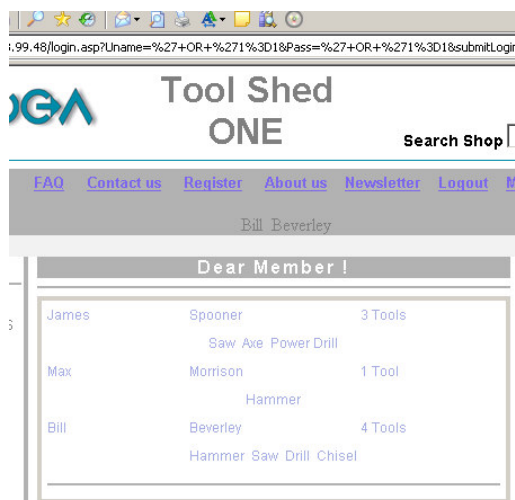
Validation and control of user input (contd)

As we do not have a Fred record, we receive no output.

But if we enter

' OR '1=1

for the userid and password we suddenly receive a listing of the whole database.



This is because we have modified the query by carefully positioning the parameters to

```
SelectMembers = " SELECT *  
" FROM tbl_ShopMembers " &  
" Where UserName = " Or '1=1' and Pass = " Or '1=1"
```

The SQL query works because the single quote before the OR closes the existing SQL query to make a null (two single quotes together) and the portion 1=1 is just a condition that is always true (because 1 always equals 1!) which causes the whole database to be listed. We have injected a new clause of SQL (i.e. or 1=1) into an existing SQL statement to change its behaviour. In this case we changed conditions so we were able to inspect extra details which in a less benign example could allow you to look at other users' account details or even passwords.

However, under the right conditions things can get a whole lot worse. You could run the;

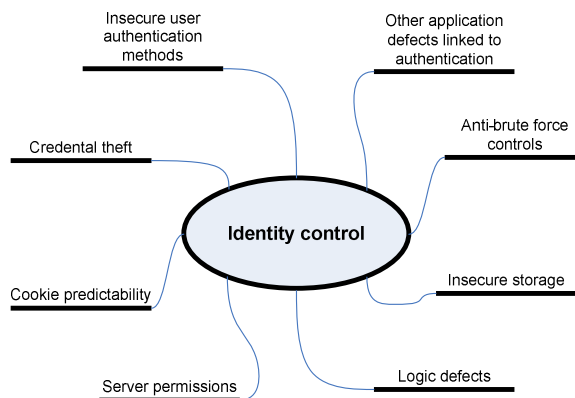
- Delete command
- Drop
- Insert command
- SQL exec commands which will allow you to run the command of your choice on the database server

It is easy to see how you could cause huge amounts of damage with this vulnerability.



Access control

Under the category of bad identity control, eight classes of application security threat can be grouped together.



The collection has been chosen because often when dealing with Web Applications and security breaches, it is not clear whether the flaw is in the authentication system or the authorisation mechanism. Consequently, a number of common situations in this area are described below.

Forceful browsing

In many e Commerce applications, authentication and authorisation functions tend to be provided by the application and therefore, not actually linked into the web server that serves the pages. This often results in a system design that is vulnerable to forceful browsing – a situation where a single start page provides all authentication and authorisation processing is bypassed and users may access pages that they are not authorised to view. By using forceful browsing and attack methods such as “Canonicalisation” and going directly to pages deep within the application the attacker has bypassed the entry page so he has not identified and authenticated himself, and hence the functions or pages in the application are available to everybody.

This may seem far-fetched but in recent tests vulnerabilities of this nature were found on three separate commercial applications;

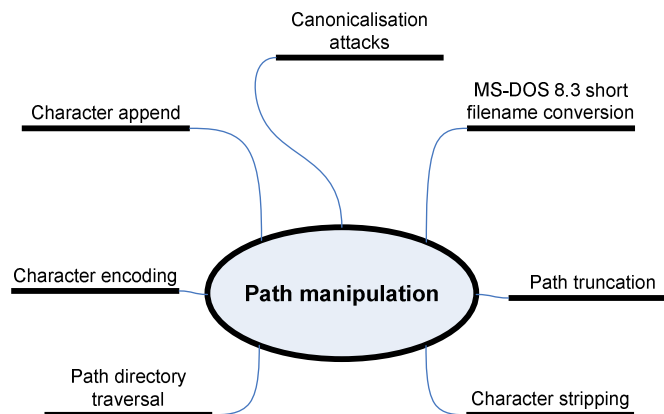
- An on-line mortgage application site
- A government information site
- A travel booking application

Some sites even used complicated token-based authentication, but with a design like this it provided less than satisfactory protection. This is often most deadly when the application authorisation system supports the concept of “power users” (privileged users that can perform administration). In all cases forceful browsing and other associated attack techniques allowed us to access either the admin screens or user account data, having not previously logged in.

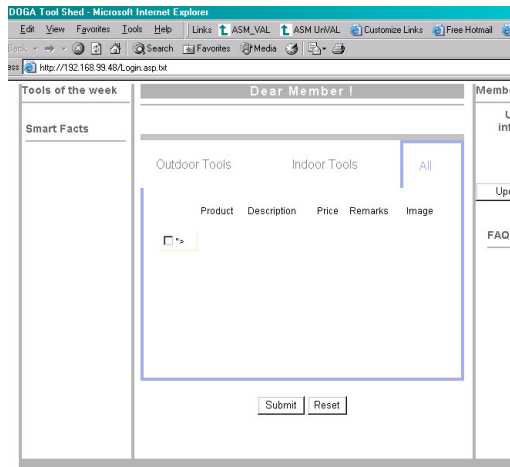
Access Control (contd)

Path Manipulation

The most well known of application hacks is called "Path manipulation" and surprisingly enough they are still fairly common. There are many variants of URL attacks and within each variant there are hundreds of mutations.



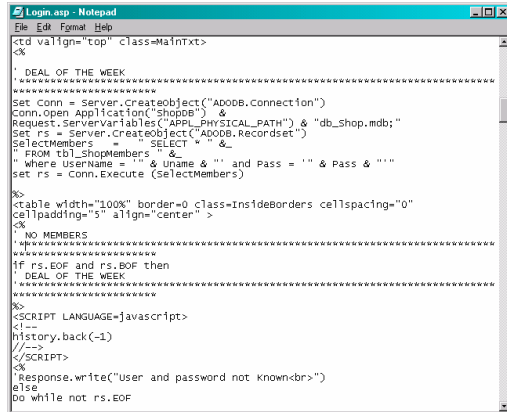
The example below shows how a backup file in the root of the web server is found by appending the extension "TXT" to a known file name.



On viewing the source of this web page one can see all the original ASP code that is usually hidden from the browser.



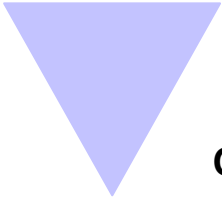
Access Control (contd)



```
login.asp - Notepad
File Edit Format Help
<td valign="top" class=MainTxt>
<%
' DEAL OF THE WEEK
*****
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open Application("shopdb") &
Request.ServerVariables("APPL_PHYSICAL_PATH") & "db_shop.mdb;"
Set rs = Server.CreateObject("ADODB.Recordset")
SelectMembers = " SELECT * &_
" FROM tbl_shopMembers " &_
" where UserName = " & uname & "' and Pass = '" & Pass & "'"
set rs = Conn.Execute (SelectMembers)
%
<table width="100%" border=0 class=InsideBorders cellspacing="0"
cellpadding="5" align="center" >
<%
' NO MEMBERS
*****
if rs.EOF and rs.BOF then
' DEAL OF THE WEEK
*****
%
<SCRIPT LANGUAGE=javascript>
<!--
history.back(-1)
//-->
</SCRIPT>
<%
' response.write("User and password not known<br>")
else
do while not rs.EOF
```

The reason for this vulnerability being exposed is because a developer has renamed a file on the server to work on the original and forgotten to delete the .txt version.

If you doubt how widespread the problem is try Googling the term “global.asa.bak” and have a look at the number of sites who expose not only their .asa file, but also their database DSN and passwords!



Conclusion – Fixing things

Web Applications have become so complex that a multi-tiered approach is required to provide acceptable application security.

Obviously, everybody wants good, resilient code. But if the code is live and being freely accessed via the Internet, every minute brings increased risk of the exposures being exploited.

After tests on moderate to large applications, we often find clients are left with 6 or 10 weeks programming effort to repair the code. That is a sizeable cost and a lengthy period to wait.

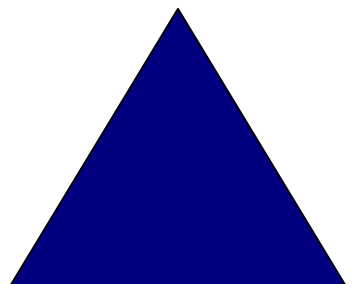
Currently, we strongly advise clients in this situation to install an application firewall. These are relatively expensive compared to, say, a Cisco PIX but do effectively cover all of the common exposures. Certainly compared to the immediate mobilisation of ten weeks programmer time, the cost and immediate risk reduction are considered by most to be reasonable.

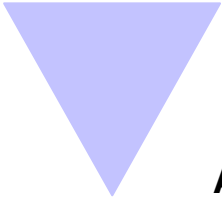
However, to ensure the security of both the code and the system design, programmers and designers need education. Workshops provide the ideal solution where teams can learn using their own application examples.

Code security also needs to be made measurable. To do this, a two-pronged approach is needed:

- The organisation needs to have security standards, covering design and coding.
- The organisation should undertake regular baseline scans. The traditional once a year penetration test is inappropriate these days. The application scans should be incremental and regular (either monthly or quarterly) and the results should be related to the organisation's application security standards.

This approach provides an effective and tested security framework.





About Attenda Limited

Every organisation is underpinned by a handful of systems which it relies upon to function efficiently. Those systems will kill the business when they are not working.

Whether they be ERP systems such as SAP, Messaging & Collaboration systems such as Microsoft Exchange, applications delivered by Citrix or those vital Web Applications that most businesses rely upon today, Attenda operates and manages those systems to ensure that they are 'Always On'. This allows our clients to selectively outsource their IT operations and re-focus on using IT to add strategic value to their business.

Through a commitment to operational excellence, we manage, secure and optimise the performance of applications, irrespective of the physical location of the infrastructure.

With over 5 years' investment into Attenda M.O., Attenda's operations platform, we provide the people, process and technology to deliver exceptionally high service levels, but at a cost that is amortised across Attenda's entire client base - currently, 95 of the UK's leading companies.

The company enjoys substantial financial backing, the industry's leading accreditations and an unrivalled portfolio of clients including Avis Europe, bmi, easyCar, Compass Group, Microsoft, NHS and Sun Microsystems.

Attenda is ISO27001 accredited, an HP SP Signature Partner, a Microsoft Gold Certified Partner (Advanced Infrastructure competency), an SAP Hosting Partner and a SunTone accredited managed service provider. Attenda is one of only seven companies to have been ranked in the UK's Sunday Times ARM Tech Track 100 for three consecutive years.

Attenda Limited
One London Road
Staines
Middlesex
TW18 4EX

www.attenda.net
info@attenda.net
T: +44 (0)1784 211 100

© Attenda Limited 2006. All rights reserved.

