



WHITE PAPER

# Common web application security problems and how to identify them

## Part 2 Broken access control

By

Lee J Lawson  
Lead Penetration Tester, **dns**.

# Introduction

This is part 2 of a series of papers designed to raise the level of security knowledge regarding common security flaws in web applications. All papers revolve around the top 10 list of web application security issues as defined by the Open Web Application Security Project (OWASP). Part 1 described the vulnerability known as **Unvalidated input**. This paper concentrates on **Broken access control** and the identification and resolution of this flaw.

Here is a list of the most common security flaws at the time of writing taken directly from the OWASP site.

1. **Unvalidated input**
2. **Broken access control**
3. **Broken authentication and session management**
4. **Cross site scripting**
5. **Buffer overflows**
6. **Injection flaws**
7. **Improper error handling**
8. **Insecure storage**
9. **Application denial of service**
10. **Insecure configuration management**

It should be noted that this paper is not intended to be used as a replacement to professional penetration testing. Rather it is aimed toward raising the level of security knowledge in the community. Professional penetration testing requires years of experience to be able to apply worthwhile interpretation to results from testing tools and gain the skills required to perform manual assurance testing.

## Broken access control

controlling access to areas of a web application, files or other accounts can sometimes be difficult

*Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorised functions.*

Controlling access to areas of a web application, files or other accounts can sometimes be difficult to achieve as it may require a blend of server and code changes. This may force a network administrator and developer to work together to achieve a single goal. Unfortunately, at best the relationship between programmers and 'techies' can sometimes be adversarial.

### The Problem

Most applications allow a user to authenticate against it and be given authorisation based on their account type. This authorisation is then used when they want access to a resource such as a document or specific web page. How an application controls the authentication mechanism and maintains knowledge of user's rights differs greatly. Probably the most common method is to use cookies to hold the authorisation details although many solutions are available. Whatever mechanism is in use, it is the checking of authorisation that is sometimes lacking.

If an application allows a user to log on and then downloads a session or permanent cookie so that session data can be maintained, but does not check the cookie to see if the user has permission to access a resource when they request it, then the access control has been broken.

Take an example of a web site that has two levels of accounts, user and administrator. The username and password entered is checked against a back end database to retrieve their authorisation details, this authorisation is then passed to the browser as a cookie. This cookie is hereafter used to check if the user/administrator has permission to access a certain resource. No problem so far.

As the user browses the site, they find a link to the 'Admin' section of the database for which access is determined by the cookie and therefore the account level. If the user attempts to access the 'protected' area of the website they will be denied and an appropriate error displayed. How the application decided on who has the required permissions and if they checked those permissions prior to accessing any area that is supposed to be protected depends greatly on the security of the mechanism in use.

Let us say that the web application has a hierarchy of folders as follows:

```
/site/admin/resources  
/site/user/resources  
/site/images
```

is there any code that checks for authorisation prior to retrieving those files? If not, can the user level account retrieve them?

The low level user account has access to **/site/users/resources** and **/site/images directories** but cannot access the **/site/admin** directory which would give them links to files in the **/site/admin/resources** directory. The **/site/admin** directory checks the cookie to see if that account holder has the correct authorisation and if they do not then denies them access. What would happen if the user attempted to bypass that piece of code by requesting files in the **/site/admin/resources** directly? Is there any code that checks for authorisation prior to retrieving those files? If not, can the user level account retrieve them?

This is just an example but it clearly demonstrates the problem that application developers face when dealing with multiple authorised accounts and their access to resources.

## Identification

Again, this is a long winded identification process as it tends to be a manual approach. You should attempt to browse your web application with different levels of accounts to see if you can enter areas or access resources that are supposed to be protected.

If you can retrieve pages that are account specific, such as account management pages, can you retrieve other user's pages? This may be easier than it seems as some applications use the URL to define the user:

<https://profile.site.com/accounts/managment.asp?act=lee>

Can you change the above example URL to read another user's details and retrieve information from their account?

<https://profile.site.com/accounts/managment.asp?act=admin>

instead of retro-fitting access controls into a current application, plan each and every action in the development stages and identify which actions require authorisation

## Countermeasures

Ensure that prior to any protected action being performed, that the authorisation of that user is checked to confirm that they have the required permissions. This should be global for the entire application.

The best approach for this is instead of retro-fitting access controls into a current application, plan each and every action in the development stages and identify which actions require authorisation. Then perform that authorisation accordingly.

The next paper in the series focuses on **Broken authentication and session management**.