



WHITE PAPER

Common web application security problems and how to identify them

Part 3

Broken authentication and session management

By

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

Introduction

This is **part 3** 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). The last issue described the vulnerability known as **Broken access control**. This paper concentrates on **Broken authentication and session management**, 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 authentication and session management

Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.

this problem really revolves around the mechanism used to identify one user and their session from another.

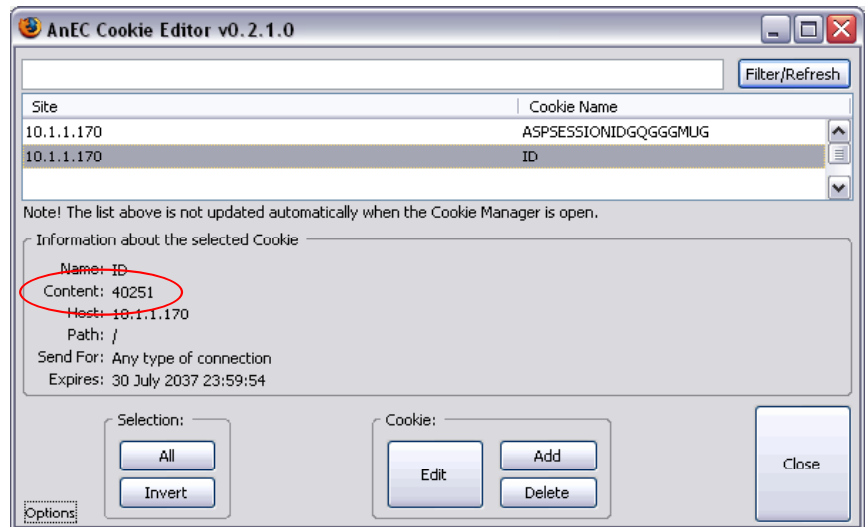
This problem really revolves around the mechanism used to identify one user and their session from another. Systems such as ASP SESSION ID's which apply a unique string of characters to identify one user from another are reasonably secure. The major problem is that some of these systems do not create a random enough string of characters, thereby making it easy for an attacker to brute force another user's ID and access their session.

The Problem

Again, an example lends itself to explaining this problem. We have a web application that uses a developer created session management system using cookies. The cookie is populated with a 5 digit string of numbers which is used to identify that user and that session. This 5 digit string of numbers is pseudo-randomly created by the system. It matters not how good the pseudo-random number generator is, the session ID is only 5 digits long. Here is some asp code that would create this cookie on the browser:

```
<%  
    'Creates a random number  
    Function RandomNumber(intHighestNumber)  
        Randomize  
        RandomNumber = Int(Rnd * intHighestNumber) + 1  
    End Function  
  
    'Assign random function a highest number of 99999  
    RndNr = (RandomNumber(99999))  
  
    'Create cookie called ID with random number  
    Response.Cookies("ID")=RndNr  
    Response.Cookies("ID").Expires = "July 31, 2037"  
    Response.Cookies("ID").Path = "/"  
%>
```

If the user were to analyse their cookies, they would find one called ID for that site and in there would be the 'random' 5 digit number defining their session.



5 digits are nowhere near enough characters to define a session.

5 digits are nowhere near enough characters to define a session. The random function may be sufficient, although computers cannot do true random in the mathematical definition of the term, the fact that it is only 5 digits long means that values for other users can be easily guessed.

If an attacker obtained a cookie and changed their 40251 ID value to something else, say 51298 or 81652 etc, would they obtain the session of another currently logged on user? Or, as an extension to that, would they obtain the session of a user that logged on some time ago, but whose session is still valid?

Identification

By analysing any session management systems in place for the given web application, you can attempt to identify what value the application developers intended to be used to define the session. This could be in the cookie, most likely, but may also be found in the content of the web page or in the URL.

Once you have found the session ID value, log off of the application and log back on again. Compare the session ID values again. Does it change? How is the value constructed, 0-9/a-z/A-Z etc? What happens if you change your ID value to something else?

This process can be made easier by having multiple browsers with multiple, individual sessions on the application. Then you can analyse the different ID values and change them to reflect one another.

As with most application security flaws, another effective method of identifying weaknesses is to bring in the application developers and discuss the vulnerability with them. This has the secondary advantage of raising their security awareness too.

another effective method of identifying weaknesses is to bring in the application developers and discuss the vulnerability with them.

ensure that your session management system is complex, random and long.

Countermeasures

Simple! Ensure that your session management system is complex, random and long. If you are using a vendor supplied session management system, such as ASPSESSIONID's, then confirm that it works as expected. Do not fall into the trap of expecting vendors to supply secure products. Confirm everything that executes on your applications.

If developing in-house session management systems, use an algorithm that combines many pieces of data together to create the ID value, such as:

- ⇒ Username
- ⇒ Time Stamp
- ⇒ MAC of client system
- ⇒ IP of client system
- ⇒ Random string from server/application.

Combine these pieces of data together and hash it to produce a unique (within reason!) value. This value could then be used as a session ID. This system would make it extremely difficult to brute force and successfully guess another users session ID.

The next part of this series concentrates on **Improper error handling** and how they are used in exploitation.