GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials (Security 401)"
at http://www.giac.org/registration/gsec

The Challenge of Security When Deploying an ASP.NET Application to the Internet

© SANS Institute 2005, Author retains full rights.

Jared Waltzer
November 16, 2004
GIAC Security Essentials Certification (GSEC)
Practical Assignment Version 1.4c, Option 1

© SANS Institute 2005 Author retains full rights.

Table of Contents

Abstract 3
Introduction 3
Authenticating the User 4
  Windows Authentication 4
  Passport Authentication 5
  Forms Authentication 6
  Authentication and Resources not Controlled by ASP.NET 6
Securing Communication 7
Prevent Specific Attacks 7
  Buffer Overflow 8
  Cross site Scripting prevention 8
  SQL Injection Prevention 9
  Canonicalization 9
Do Not Leak Information to the Client 10
Conclusion 10
References 12

© SANS Institute 2005, Author retains full rights

© SANS Institute 2005 Author retains full rights.

Abstract

The purpose of this paper is to outline certain security concerns that need to be addressed when deploying an ASP.NET application on the World Wide Web. In particular, it will cover how to prevent sensitive data from falling into the wrong hands by means of authentication and secure communication. It also explains how sensitive information can be protected by preventing common security attacks and not leaking data when an error condition arises in the application.

Introduction

With ASP.NET, Microsoft has provided a powerful framework to develop web applications for company intranets, extranets, and the Internet. However, there are several things to consider before deploying an application to the World Wide Web.

One consideration is what Authentication mechanism will be used. Some sites may be designed to be accessible to any visitor, and contain data we will allow anyone to see. But, often, we will also need to restrict certain resources and data to specific users. To know whether someone should be able to access a certain resource, we first have to know who they are by means of authentication. However, the more secure authentication mechanisms available to ASP.NET do not always work well in an Internet setting.

Another consideration is how to get sensitive data from the server to the client. This is extremely critical as this information is not being transferred over a private network, but over the public Internet where there is an even greater chance of someone trying to intercept our sensitive information.

By putting our application on the Internet we are also greatly increasing its exposure to specific hacking attacks such as buffer overflows, cross-site scripting (XSS), SQL injection, and Canonicalization. These attacks, if successful, could allow sensitive information to fall into the wrong hands. Thus an ASP.NET application on the Internet needs to take precautions to prevent these attacks.

Finally, if an application should error or fail, it can reveal sensitive information. Thus, care should be exercised so that the application does not provide a client with valuable information as a result of an error condition.

While these matters are hardly the only security considerations when deploying an ASP.NET application to the Internet, they are certainly critical.

© SANS Institute 2005 Author retains full rights.

Authenticating the User

If the pages you create for your website are to be public and are designed to be accessible to any visitor, the default authentication and authorization settings for ASP.NET pages may be sufficient. They allow anyone to access the pages from anywhere on the network or Internet. (Anderson)

However, there is often content that we don't want readily available to the public in general. To determine whether a visitor should have access to a specific page or resource on your site, you first have to determine who it is that is trying to access that resource. To do this, the client's identity must be validated. This act is called authentication. (Peiris)

ASP.NET provides three types of authentication in addition to the Default IIS ("None") authentication. They are:

1) Windows built-in ("Windows") authentication,

2) Passport-based ("Passport") authentication, and

3) Forms-based ("Forms") authentication

(Anderson)

The challenge of choosing a method for authentication for an ASP.NET Internet application is choosing one that will be compatible with the greatest number of potential clients, while at the same time protecting sensitive resources and data, including the client's credentials.

Windows Authentication

Windows authentication provides the most secure way of controlling access and securing your ASP.NET applications. With Windows authentication, IIS performs the initial authentication. Requested resources are then accessed under the context of this account. For many reasons Windows authentication is an excellent choice for authenticating users on a corporate intranet. (Anderson)

However, it does present some issues for the Internet. One is that for each user that will use the Internet, an account needs to be created in Windows, and the user needs to be provided with that username and password so that they may be able to log onto the site. This means that you need to know in advance which users will be accessing your site. (Anderson)

© SANS Institute 2005 Author retains full rights.

Additionally, when choosing Windows authentication, you additionally need to choose how IIS will perform the initial authorization. The choices are Basic, Digest, or Integrated Windows. Basic Authentication, which is based on a proposed Internet standard, is supported by most browsers found on the Internet. It requires the user to supply credentials in the form of a user name and a password to prove their identity, and then the user's credentials are transmitted from the browser to the Web server in an unencrypted Base 64 encoded format. Because the credentials are passed unencrypted, however, the user names and passwords can easily be intercepted with a network sniffer. (Anderson)

Digest authentication is like Basic authentication, but more secure in that instead of transmitting the user's credentials in unencrypted form, it sends a hash of the credentials. The drawback is that Digest Authentication requires Internet Explorer 5.0 or later and specific server configuration. Integrated Windows Authentication uses a cryptographic exchange based on Kerberos or NTLM to confirm the identity of the user. It is also only supported only by Internet Explorer. (Anderson)

As we can see, the two most secure methods are often not feasible for a site on the Internet because they severely limit compatibility with the potential target audience by requiring Microsoft's Internet Explorer. Although, a corporation may mandate the use of Internet Explorer for all users on its corporate intranet, the World Wide Web presents a different situation. Users may want to access your site with any one of a number of alternative browsers including, but not limited to, Netscape, Mozilla, Firefox, Opera, or Safari.

What is the solution for using Windows Authentication on the Internet? Basic Authentication provides the best compatibility with the largest numbers of users, but it should never be used alone because of its use of unencrypted credentials. However, when combined with a secure channel (most commonly SSL) to encrypt the transmission, it becomes feasible for Internet applications. When using Basic authentication, IIS should be configured to require that SSL be used not only on the logon page, but all subsequent requests because credentials are passed with each subsequent request. (Meier)

Passport Authentication

Passport authentication uses Microsoft's "Passport Service" to authenticate users on any passport-enabled site, anywhere on the Internet. It enables the use of a single-sign-on across multiple sites on the internet, even ones that you may not provide yourself. When a user logs onto a participating site, their browser sends the credentials to the passport service, which then authenticates them and places a secure cookie on their machine allowing them to access participating sites for the length of the session.

© SANS Institute 2005 Author retains full rights.

Unfortunately, passport authentication requires a paid subscription to the service, and special software installed on the web server. (Anderson) This will no doubt limit its use for many potential Internet web applications.

Forms Authentication

Forms-based ("Forms") authentication is the most popular authentication choice for applications that will be deployed to the web. Unauthenticated requests are redirected to a form page where the user can provide their login credentials. Upon authenticating the request, the system issues a cookie that contains their credentials that the browser then sends with all subsequent requests. (Anderson) This form of authentication is also often referred to as cookie-based authentication.

Forms authentication has many advantages for Internet applications. For one, forms-based authentication allows you ditch the ugly Windows Logon dialog, and replace it with an attractive custom form. (Anderson) Also, it is compatible with a large number of web browsers. Finally, it is ideal because it does not require users to have Windows accounts--any data store can be used to validate user credentials, including a SQL Server database, which is perhaps the most common solution. (Meier)

Although Forms Authentication has many points in its favor, there are still considerations that should be kept in mind. Again, with this solution, you will want to use it in combination with SSL, at least to protect the initial logon credentials. On subsequent requests, you will also want to use SSL to protect the forms authentication ticket. Alternatively, you can encrypt the form's authentication ticket by configuring the protection attribute of the <forms> element to "All" or "Encrypt" in the web.config file, and by using the Encrypt method of the FormsAuthentication class to encrypt the ticket. (Meier)

Authentication and Resources not Controlled by ASP.NET

One thing to remember when using an ASP.NET authentication mechanism, is that the authentication process used by ASP.NET only applies to resources associated with ASP.NET. This means that access control is only applied to files defined as "application" files. These include such files as .aspx pages, .ascx components, and .cs and .vb code files. It does not apply to resources such as images, Word documents, zip files, PDF files, and other types of files. (Anderson)

This presents an obvious security problem if you have if you have non-application files that you wish to protect. Although, the user would likely not know where such resources are located, relying on obscurity is not a good security

© SANS Institute 2005 Author retains full rights.

practice. Potentially, a user could guess the path to such an unsecured resource, effectively circumventing authentication.

A good practice would be to never place anything in the Web root that you aren't absolutely comfortable letting a hacker see unless the file extension is mapped to an ISAPI handler that restricts access to files such as the previously mentioned .aspx, .ascx, .cs, and .vb files. (Adams) When using Windows authentication, you could restrict access to such files by using standard Windows techniques and ACLs. (Anderson) To secure images and other such files under Forms authentication, you may need to create a special gatekeeper such as described at http://www.codeproject.com/aspnet/imagehandler.asp. (Coleman)

Securing Communication

Restricting access to certain users is one way of protecting sensitive information. However, often sensitive data will be passed between an authenticated client and a web application over the Internet. This sensitive data could include a user's credentials as mentioned in the previous section, possibly credit card numbers, banking transactions, or any other piece of sensitive information. Certainly, transmitting such information over a public resource such as the Internet in an unencrypted form would be unwise, as it would be susceptible to interception. To guard against unwanted disclosure of and unauthorized modification of the data, there must be secure communication between the client and server application. (Meier)

A basic requirement for security on the Internet is privacy. That is, it is essential that sensitive data transmitted over the web remains private and confidential. This is usually accomplished by means of encryption, (Meier) and the standard for such secured communication on Internet is SSL. That remains true when the information is served over the World Wide Web by an ASP.NET application. SSL is used to establish the secure channel between client and server.

SSL configuration occurs on the IIS Web server, and requires a server authentication certificate to be installed on the server. All connections between a browser and the server that pass sensitive information, including any unencrypted user credentials, should be secured with SSL.

Prevent Specific Attacks

A number of attacks may come upon an ASP.NET application that is exposed to the Internet. Many of these attacks are done by embedding malicious strings in query strings, form fields, cookies, and HTTP headers. These attacks may

© SANS Institute 2005 Author retains full rights.

include buffer overflow, cross-site scripting (XSS), SQL injection, and Canonicalization, among others. (Meier)

Fortunately, the risk from such attacks can be minimized with proper care. In fact risk from all of these attacks can be practically eliminated with proper input validation. An application should validate and clean all user input before processing it, and should never directly output user input to the browser. ASP.NET's powerful built-in validator controls can be very helpful in this regard to clean user input before processing it. (Adams) Input should be validated for type, length, format and/or range. (Krishnan)

Buffer Overflow

In a buffer overflow attack, an attacker sends too much data, and if the program does not check the size of data, the extra or overflowed data may be executed as if it were a set of command instructions. Fortunately, ASP.NET minimizes the threat of this happening, because in managed .NET, code array bounds are automatically checked. Thus, some may feel that there is no need for concern with buffer overflows in ASP.NET. However, ASP.NET is still susceptible to buffer overflows where managed code calls unmanaged APIs or COM objects. (Krishnan)

Thorough input validation will eliminate the risk of buffer overflows. Additionally, a good practice is to limit the application's use of unmanaged code by sticking to .NET's managed API whenever possible, and when calling unmanaged code check all values that are being passed. (Krishnan)

Cross site Scripting prevention

In a cross-site scripting (XSS) attack, someone could possibly be able to hijack information from visitors of your site by injecting client-side scripting into your application. If you do not validate the input, you could be allowing your site to become a tool for someone looking to do a malicious act such as hijacking cookies, or a user's session information. (Santry)

Again ASP.NET's validator controls can be helpful in cleaning user input. Additionally, another form of protection is provided by ASP.NET 1.1, and that is page validation. This is done by adding a "ValidateRequest" attribute to the page itself, or in the web.config. "ValidateRequest" checks for any type of script or html characters. With this validation in place, if someone were to enter a script into your application, no damage would occur, only an error would be displayed. (Santry)

© SANS Institute 2005 Author retains full rights.

SQL Injection Prevention

SQL injection attacks again take advantage of code that does not filter input that is being entered into a form. A vulnerability can exist when direct user input is used to generate dynamic SQL that is executed on the back-end. An attacker could enter SQL commands, or partial SQL commands, into a login form and receive potentially sensitive information. If the correct SQL was entered into a login form, a hacker could possibly receive a positive logon allowing access to your site. Inserting malicious SQL into any form that is not validated could possibly reveal database structure, user names and passwords, or other sensitive information from the database. Again the first key to preventing this attack is to validate all user input to make sure the data entered matches the type of data expected. (Santry)

You can add another layer of protection when using numeric fields in a database-driven application, by actually casting the variables to a numeric type before using them. If a user places something nonnumeric into the field an exception will be thrown. (Adams)

SQL injection becomes a risk when using dynamically generated SQL. Thus, to prevent SQL injection, avoid dynamically generated SQL in your code. Instead use parameterized queries which make SQL injection impossible. (Santry)

Stored Procedures offer even more protection. By using a stored procedure you can provide greater control of how the database is accessed. Execution of stored procedures should be limited to specific accounts, and those accounts should be allowed to only execute stored procedures. You do not need to provide the account any other permission, such as write access. Thus any interaction to the database is always done using SQL in a stored procedure which you wrote. (Santry)

Canonicalization

Canonicalization is the process by which various equivalent forms of a name can be resolved to a single standard name, or the "canonical" name. For example, on a specific computer, the names c:\dir\test.dat, test.dat, and ..\..\test.dat might all refer to the same file. Through canonicalization such names are mapped to a name similar to c:\dir\test.dat. (Ballard)

An application is susceptible to canonicalization issues when it makes a security decision based on the name of a resource that has been received as input. Files, paths, and URL are all susceptible to this type of vulnerability. To prevent this, avoid accepting file names as input, and if there is a need for accepting input, convert the name to its canonical form before providing security decisions.

© SANS Institute 2005 Author retains full rights.

Additionally, make sure the filenames that are received are well-formed and within your application's directory hierarchy. (Krishnan)

Canonicalization issues are involved in a recently reported on security vulnerability in ASP.NET that could allow an attacker to gain access to secured content. If the attacker includes a backslash ("\") in the URL they can bypass authentication and directly access a resource. Microsoft recommends that Web site owners and developers programmatically check for canonicalization problems, or install a recently released HTTP module that will protect all ASP.NET applications on the server against known URL canonicalization problems. (Microsoft)

Do Not Leak Information to the Client

Secure exception handling can be used to prevent valuable system-level information from being returned to the client. In the event of a failure, do not expose unnecessary information such as stack trace details. Instead, return generic errors to the client. Detailed error messages can be sent to an error log with the client only receiving a minimal message. (Meier)

Use exception handling to catch exceptions, which will prevent your application from being left in an inconsistent state that may lead to information disclosure. (Meier)

Before placing an ASP.NET web application into production on the Internet, first check the configuration. Disable tracing and debugging support, and make sure the "customErrors" tag in the web.config file is not set to "off." This will help prevent leaks of information such as filenames and paths, and possibly even source code, when an error occurs in the application. (Adams) You likely do not want your users helping you debug your application, so why would you want to provide them with debugging information. (Santry)

Conclusion

When deploying an application on the World Wide Web it is important that visitors be able to access information that we decide they should have access to, while preventing access to sensitive information to which they should not have access.

This will require implementing an authentication mechanism in ASP.NET that is compatible with a wide variety of web browsers, and yet able to protect sensitive data. Additionally, it will require the use of secure channel to transmit sensitive

© SANS Institute 2005 Author retains full rights.

information such as user credentials. Most commonly this will mean the use of Forms authentication in combination with SSL. In the case of Windows authentication, this will usually mean using standard Basic authentication in combination with SSL. Additional precautions are required to prevent unauthorized access to non-application files such as images and PDF documents due to limitations in ASP.NET.

Diligent user input validation is essential to minimize risks from attacks such as buffer overflow, cross-site scripting (XSS), SQL injection, and Canonicalization. Additionally, errors want to be handled in a graceful manner by use of exceptions, and ASP.NET should be configured not to reveal sensitive information in error messages.

By addressing these challenges at the ASP.NET application level, your ASP.NET application is closer to being ready for the World Wide Web.

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.

References

Richard J. Anderson, Brian Francis, Alex Homer, Robert Howard, David Sussman, and Karli Watson. “Professional ASP.NET 1.0: Securing ASP.NET Applications, Part 1.“ 24 May 2004. URL: http://securityadvisor.info/doc/14117 (12 November 2004).

Peiris, Gayan. “Developing Secure Web Sites with ASP.NET and IIS (Part I).” 4 March 2003. URL: http://www.c-sharpcorner.com/Code/2003/March/SecureSiteWithASPNET.asp (12 November 2004).

J.D. Meier, Alex Mackman, Michael Dunner, and Srinath Vasireddy. “Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication.” November 2002. URL: http://msdn.microsoft.com/security/securecode/dotnet/default.aspx?pull=/library/en-us/dnnetsec/html/secnetlpMSDN.asp (12 November 2004).

J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan. “Improving Web Application Security: Threats and Countermeasures.” June 2003. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp (12 November 2004).

Adams, Lamont. “Learn to secure your ASP.NET applications with these tips.” 8 July 2002 URL: http://builder.com.com/5100-6387-1044869.html (12 November 2004).

Coleman, James. ”Securing Images under Forms-Based Authentication in ASP.NET Applications.” 22 October 2002. URL: http://www.codeproject.com/aspnet/imagehandler.asp (12 November 2004).

Krishnan, V.R. “Securing your ASP.NET web application.” 31 August 2004. URL: http://www.c-sharpcorner.com/Code/2004/Sept/securewebappl.asp (12 November 2004).

Santry, Patrick. “Securing Your ASP.NET App Against Cross-site Scripting (XSS) Attacks.” URL: http://www.wwwcoder.com/main/parentid/258/site/2885/68/default.aspx (12 November 2004).

Santry, Patrick. “Preventing SQL Injection Attacks.” URL: http://www.wwwcoder.com/main/parentid/258/site/2966/68/default.aspx (12 November 2004).

© SANS Institute 2005 Author retains full rights.

Ballard, Paul. “Canonicalization Security Vulnerability in ASP.NET.” 6 October 2004. URL: http://www.theserverside.net/news/thread.tss?thread_id=29248 (12 November 2004).

Microsoft Corporation. “What You Should Know About a Reported Vulnerability in Microsoft ASP.NET.” 5 October 2004 updated 15 October 2004. URL: http://www.microsoft.com/security/incident/aspnet.mspx (12 November 2004).

Microsoft Corporation. “Checklist: Securing ASP.NET.” January 2004. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secmod/html/secmod98.asp (12 November 2004).

© SANS Institute 2005, Author retains full rights.

© SANS Institute 2005 Author retains full rights.