登录

ASP.NET Web应用程序中的高风险漏洞

惊鸿一瞥最是珍贵 / 2019-08-07 07:23:00 / 浏览数 3852 安全技术 WEB安全 顶(0) 踩(0)

前言

在本博客中，我们将讨论在ASP.NET应用程序中发现的本地文件包含和SQL注入的漏洞以及如何缓解它们。
当Web应用程序允许用户从Web服务器读取任何文件而不考虑其扩展名时，ASP.NET
Web应用程序中会出现本地文件包含(LFI)漏洞。这可能导致泄露敏感信息，当与其他漏洞(如远程执行)结合使用时，会导致恶意用户获得对Web服务器的完全控制。
当用户输入未被清理并作为参数发送到SQL语句时，会发生SQL注入漏洞。利用SQL注入漏洞，恶意用户可能会篡改数据，导致会话劫持(帐户接管)或在数据中注入有害脚本
开发人员必须接受编码标准方面的培训，使得他们能够以安全的方式编写代码。代码审查程序必须到位，以便能够在开发/测试阶段早期识别此类漏洞。测试阶段还必须包括

ASP.net应用程序中的SQL注入

```
xxxxxxxxxxxxxxxxxxxxxxx ASP.NET Code Snippet start xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx


<%@ Page Title="" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeFile="Login.aspx.cs" Inherits="Login
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" Runat="Server">
   <form id="Form1" runat="server">
       <table>
           <tr>
               <td>User:</td><td><asp:TextBox ID="user" runat="server"></asp:TextBox></td>
           </tr>
           <tr>
               <td>Password:</td><td><br /><asp:textbox TextMode="password" id="pass" runat="server" /></td>
           </tr>
           <tr>
               <td><asp:button Text="Login" runat="server" id="btn" onClick="submit"/></td>
           </tr>
       </table>
   </form>
</asp:Content>

//server side Submit button method

protected void submit (object sender, EventArgs e)
{
   string query1 = "Select username, password from admin where username <> 'admin' and password = '" + pass.Text.Trim() + "' "

   try
       {
           DataSet dSet1 = new DataSet();
           dSet1 = fetchWebDB(query1);
       }

   DataSet fetchWebDB(string query)
       {
           // connect to data source
           OleDbConnection myConn = new OleDbConnection("Provider=SQLOLEDB;Data Source=.;Initial Catalog=AMS;User ID=sa;Passwo
           myConn.Open();
           // initialize dataadapter with query
           OleDbDataAdapter myAdapter = new OleDbDataAdapter(query, myConn);

           // initalize and fill dataset with query results
```

```
            DataSet myData = new DataSet();
            myAdapter.Fill(myData);
            myConn.Close();

            // return dataset
            return myData;
        }
}
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ASP.Net Code snippet end xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

上面提到的代码容易受到SQL注入攻击，因为开发人员在将数据发送到SQL语句之前没有对用户输入进行清理。User和Pass变量的值按原样传递给SQL语句，这是禁止的。用
必须先对用户输入进行编码，然后才能将其保存到数据库中。
其次，开发人员使用"系统管理员"用户连接MS
SQL数据库来获取数据。在这种情况下，所有SQL语句都将在"sa"(系统管理员)用户的上下文中执行。在这种情况下，所有SQL语句都将在"sa"(系统管理员)用户的上下文中执
SQL
Server中，"sa"用户具有最高级别的权限。如果恶意用户获得对此应用程序的访问权限，或者以某种方式知道"sa"用户的密码，则他可以执行一些特定操作，这些操作可以帮
第三，不建议在应用程序页面中保留连接字符串，因为源代码泄露也会暴露连接字符串。建议将其加密然后保存在
`<connectionstring/> key`下的`web.config`中。如果恶意用户访问web.config文件，他也无法解密连接字符串。

如何加密ASP.net Web应用程序中的连接字符串？

建议系统管理员/开发人员在将应用程序移至生产环境之前，应在web.config中加密连接字符串和MachineKey。在配置文件中应始终避免使用明文密码，因为它们会成为W
MachineKey本身必须加密，因为此密钥用于加密/解密和验证ASP.NET Cookie和防伪令牌。
MachineKey和连接字符串一样重要，因为一旦恶意用户获得了这个密钥，他就可以创建一个经过身份验证的cookie，允许他以任何用户的身份登录。
可以通过以下命令加密web.config中的MachineKey元素：

```
aspnet_regiis -pe "system.web/machineKey" -app "[Your Application Name]"
```

aspnet_regiis.exe位于Web服务器上的以下路径：

```
c:\Windows\Microsoft.NET\Framework\v4.0.30319\aspnet_regiis.exe
```

要使用RSA密钥容器加密连接字符串，必须遵循以下方法
我们将使用现成的RSA密钥容器，当安装.Net时，默认情况下会创建"NetFrameworkConfigurationKey"
首先，要读取RSA密钥容器，必须授权Web应用程序的ASP.NET身份：

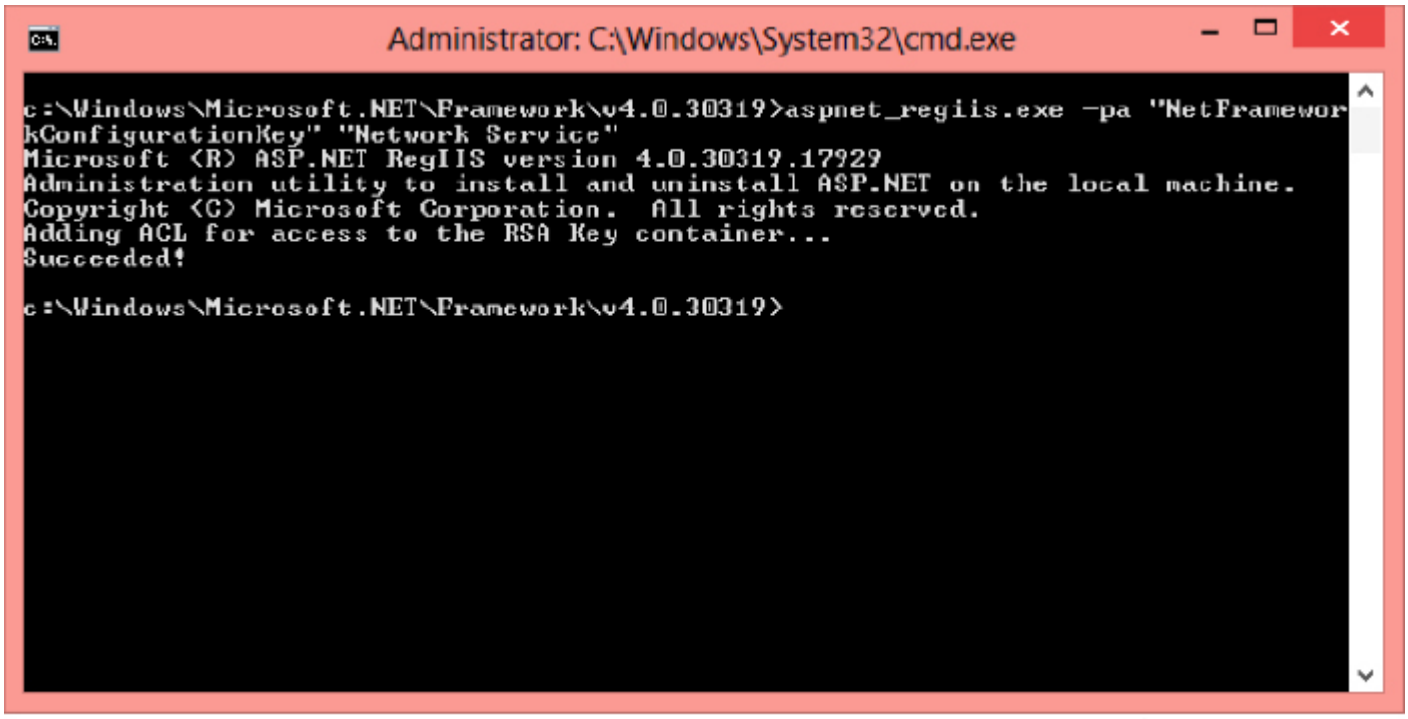

Image 1: Web App Authorization

加密web.config中的敏感数据元素"connectionstring"，可以运行以下命令：

```
aspnet_regiis -pe "connectionStrings" -app "[Your Application Name]"
```

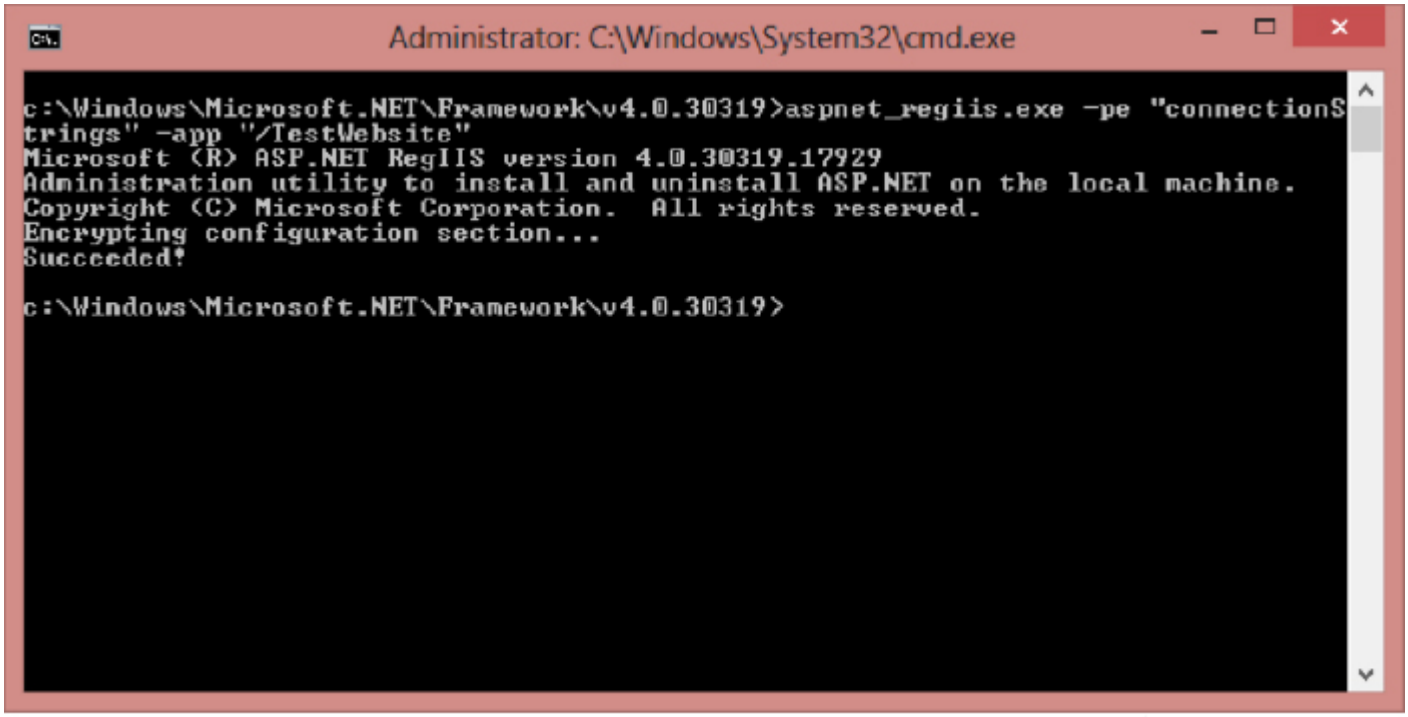

Image 2: Encripting Sensitive Data Element

运行完命令后，打开web.config

Web.config

```
<?xml version="1.0"?>

<configuration>
  <connectionStrings configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>Rsa Key</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>tY23UMWaO+QQ6+T8GB1ZccDqZUllxTVkByHzhM15oGWFCbOCCKbyevuL8Yxl4COBmto8/SokVMCV
          </CipherData>
        </EncryptedKey>
      </KeyInfo>
      <CipherData>
        <CipherValue>QaOex36XuY29/ArcpejLdCqNQaDIsQiXeU4n0cvJvbqeTFaqzMAo/ItX7lvCZk+VSQxgWNkIOPdYveYU
      </CipherData>
    </EncryptedData>
  </connectionStrings>

  <appSettings configProtectionProvider="RsaProtectedConfigurationProvider">
    <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
          <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
          <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
            <KeyName>Rsa Key</KeyName>
          </KeyInfo>
          <CipherData>
            <CipherValue>D+GXLinTA424SCJ1psspmHnguhGcFW5fYWOVPQkdKiDZOkTEYG+vsv61QUaqTotoOu7/35tmrL3a
          </CipherData>
```

Image 3: Web Configuration

另一种方法是 - 在连接字符串中添加“Trusted connection = true”：

```
<connectionStrings>
<add name=”DefaultConnection” connectionString=”Server=localhost;Database=mydb;Trusted_Connection=True”
</connectionStrings>
```

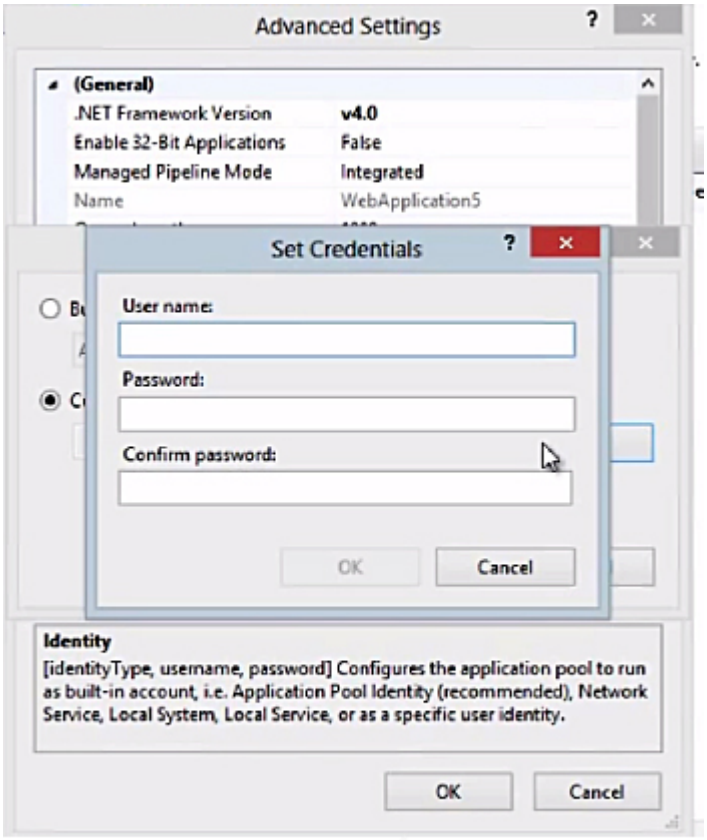

Image 4: IIS Screenshot

上面的屏幕截图是从Internet Information
Services（IIS）获取的，关于选择用于运行Web应用程序的应用程序池。可以将自定义帐户(服务帐户)配置为应用程序池标识运行。Web应用程序现在将使用此标识连接到数
Server，以便它可以运行SQL查询并将数据保存在数据库中。
如果恶意用户收集了不同的凭据，他可能会使用这些凭据来强制在Web服务器级别或操作系统级别运行的其他服务，例如FTP或SMB服务。可以通过对任何用户控制的参数强
另一件需要注意的重要事情是禁用对网站的匿名访问并启用身份验证，以便以后可以跟踪访问期间发生的事件。
此外，应在IIS中禁用网站的目录列表：
打开IIS管理器。
选择网站以禁用文件列表。
双击IIS部分中的目录浏览图标。
管理员还应限制对具有机密/配置相关数据的特定目录的访问。可以拒绝访问位于特定文件夹中的特定文件：

```
<location path="confidential">
    <system.webServer>
        <security>
            <authorization>
                <remove users="*" roles="" verbs="" />
                <add accessType="Allow" roles="Administrators" />
            </authorization>
        </security>
    </system.webServer>
</location>>
```

缓解措施：

有时，开发人员认为他们可以通过使用存储过程从SQL注入漏洞中拯救Web应用程序。然而，编写得不好的存储过程可能是导致web应用程序中的SQL注入漏洞的原因。
考虑以下存储过程：

```
ALTER PROCEDURE [dbo].[SearchLeaves]
 @searchleavetype VARCHAR(50) = ''
AS
BEGIN
DECLARE @query VARCHAR(100)
SET @query = 'SELECT * FROM LEAVES WHERE category LIKE ''%' + @searchleavetype + '%''';
EXEC(@query)
END
```

在上面的存储过程中，字符串连接是一个问题，可以利用存储过程从数据库中的任何表中读取数据。
参数化可以解决上述问题：可以重写上述存储过程来缓解SQL注入漏洞。

```
ALTER PROCEDURE [dbo].[SearchLeaves]
 @searchleavetype NVARCHAR(50) = ''
AS
BEGIN
 DECLARE @query NVARCHAR(100)
 DECLARE @msearch NVARCHAR(55)

 SET @msearch = '%' + @searchleavetype + '%'
 SET @query = 'SELECT * FROM LEAVES WHERE category LIKE @search'
 EXEC sp_executesql @query, N'@search VARCHAR(55)', @msearch
END
```

可以创建添加“%”字符的新字符串，然后将此新字符串作为参数传递给SQL语句。
缓解SQL注入漏洞的另一个操作是只向负责执行SQL语句的当前用户授予有限的权限。
让我们假设有一个网页，它显示员工的休假情况。可以在数据库中创建新的SQL用户。通过映射自定义角色，用户只能执行存储过程。通过在此用户的上下文中执行存储过程
考虑创建以下角色和用户以减轻SQL注入：

```
CREATE ROLE CustomFetchDataRole
CREATE LOGIN TestUser WITH PASSWORD = '$Passw0rd@123##’
ALTER ROLE CustomFetchDataRole ADD MEMBER [TestUser]
IF NOT EXISTS (SELECT * FROM sys.database_principals WHERE name = N' TestUser ')
BEGIN
 CREATE USER [TestUser] FOR LOGIN [TestUser]

END;
GO

GRANT EXECUTE ON dbo.uspGetLeavesInfo TO TestUser
GRANT EXEC ON TYPE::DBO.MyTableType TO [CustomFetchDataRole]
```

仅授予对数据库有限的权限可以在保护数据方面起到很大的作用。恶意用户将无法直接访问任何表，即使他能够以某种方式登录到SQL
Server。现在，存储过程(考虑参数化)要么将预期的数据返回到网页，要么不返回任何内容。
另一件需要记住的重要事情是，数据库管理员应该禁用“xp_cmdshell”存储过程。数据库用户不应具有启用此存储过程的权限。开发人员永远不应使用此存储过程，因为恶意
数据库管理员应启用SQL Server审核日志，以保留登录审核，SQL Server审核，SQL跟踪，DML，DDL和登录触发事件的日志。
考虑到攻击的严重性，SQL注入是最危险的攻击，它可以危害整个数据库服务器和Web服务器。可以通过对存储过程进行参数化并仅向SQL
DB用户授予有限的权限来阻止此攻击。
SQL
DB管理员应创建不同的用户，并仅允许他们访问服务器上的特定数据库。只允许有限数量的用户访问单个数据库。每个用户必须仅具有访问数据库的有限权限。
应禁用Sysadmin(SA)用户，而应创建具有精简权限的自定义系统管理员用户，这样，如果恶意用户以某种方式获得了对数据库的访问权限，他应该无法使用“SA”用户获得系

ASP.NET（本地文件包含）

LFI漏洞使得恶意用户能够访问Web服务器上的其他文件。这是OWASP 10漏洞列表中列出的最关键漏洞之一。
下面是允许下载某些文档的示例ASP.NET代码。

```
http://abcd.com/<vuln.page>?<vuln query string>

Consider the below code:

public partial class Downloads_Download : System.Web.UI.Page
{
 string sBasePath="";
 protected void Page_Load(object sender, EventArgs e)
 {

   try
   {
     string filename = Request.QueryString["fname"];
     string sBasePath = System.Web.HttpContext.Current.Request.ServerVariables["APPL_PHYSICAL_PATH"];
     if (sBasePath.EndsWith("\\"))
       sBasePath = sBasePath.Substring(0, sBasePath.Length - 1);
     sBasePath = sBasePath + "\\" + ConfigurationManager.AppSettings["FilesPath"] + "\\" + filename;
     Response.AddHeader("content-disposition", String.Format("attachment;filename={0}", filename));
     Response.WriteFile(sBasePath);
   }
   catch (Exception ex)
   {
Response.Redirect("~/Error.aspx?message=" + ex.Message.ToString() + " path=" + sBasePath);
   }
```

```
 }
}
```

让我们来浏览一下上面的代码，找出它的问题所在。
“fname”参数在没有任何验证或清理的情况下被接受。此代码容易受到目录遍历漏洞和信息泄漏的影响，这些漏洞可能会泄露导致进一步威胁的敏感数据。
应该以只允许下载特定文件扩展名的方式编写代码。Content type: application / octet-stream不安全，不应设置为switch
case函数中定义的默认content-type。
看看下面的代码，它比之前的代码更安全：

```
protected void Page_Load(object sender, EventArgs e)
 {
   //string file = Server.MapPath(HttpUtility.UrlEncode(functions.RF("file")));
   string file = Server.MapPath(functions.RQ("file"));
   if (File.Exists(file))
   {
     Regex reg = new Regex(@"\.(\w+)$");
     string ext = reg.Match(file).Groups[1].Value;
     switch (ext)
     {
       case "xls":
         Response.ContentType = "application/vnd.ms-excel";
         break;
       case "xlsx":
         Response.ContentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";
         break;
       case "ppt":
         Response.ContentType = "application/vnd.ms-powerpoint";
         break;
       case "pptx":
         Response.ContentType = "application/vnd.openxmlformats-officedocument.presentationml.presentation";
         break;
       default:
     Response.ContentType = " application/pdf ";
         break;
     }
     byte[] buffer = File.ReadAllBytes(file);
     Response.OutputStream.Write(buffer, 0, buffer.Length);
     Response.AddHeader("Content-Disposition", "attachment;filename=" + Path.GetFileName(file));
   }
   else
   {
     Response.Write("This file Extension is not allowed");
   }
 }
```

在上面的代码中，用户只能下载扩展名为xls，xlsx，ppt，pptx和pdf的文件。
开发者已经为每个文件扩展名设置了一个“content type”，这意味着对于PDF文件类型，content type是application/pdf，并且只允许下载PDF文件。
MIME(多用途互联网邮件扩展)内容类型描述内容/文件的媒体类型，帮助浏览器正确处理和显示内容。在下载或上传文件时，应用程序必须描述文件的“content type”，否则，浏览器将不知道要处理的预期内容与报告的MIME类型不同。
默认content type不应为“application/octet-stream”。开发人员不应在其代码中使用此content type，除非是有别的目的。
除了上述问题外，ASP.NET Web应用程序中还存在其他严重漏洞。虽然所提到的缓解步骤不足以保护ASP.NET
Web应用程序的整体安全，但通过遵循这些步骤，可以至少消除ASP.NET Web应用程序中的基本漏洞。我们将在即将发布的博客中讨论剩余的高风险漏洞。

■■■■https://www.gspann.com/resources/blogs/high-risk-vulnerabilities-found-in-asp.net-web-applications

上一篇：代码审计之旅 下一篇：SURF路由器安全漏洞研究
1. 0 条回复
   - 动动手指，沙发就是你的了！

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板