عنوان: **مقابله با XSS ؛ یکبار برای همیشه!**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۳۰ ۰۱:۱۹:۰۰
آدرس: www.dotnettips.info
گروه‌ها: Security

ASP.NET به صورت پیش فرض در مقابل ارسال هر نوع تگی عکس العمل نشان می‌دهد و پیغام خطای یافتن خطری بالقوه را گوشزد می‌کند. اما بین خودمان باشد، همه این قابلیت را خاموش می‌کنند! چون در یک برنامه واقعی نیاز است تا مثلا کاربران تگ html هم ارسال کنند. برای نمونه یک ادیتور متنی پیشرفته را درنظر بگیرید. خاموش کردن این قابلیت هم مساوی است با فراهم کردن امکان ارسال تگ‌های مجاز و در کنار آن بی دفاع گذاشتن برنامه در مقابل حملات XSS.
توصیه هم این است که همه جا از توابع مثلا HtmlEncode و موارد مشابه حتما استفاده کنید. ولی باز هم خودمونیم ... چند نفر از شماها اینکار را می‌کنید؟!
بهترین کار در این موارد وارد شدن به pipe line پردازشی ASP.NET و دستکاری آن است! اینکار هم توسط HttpModules میسر است. به عبارتی در ادامه می‌خواهیم ماژولی را بنویسیم که کلیه تگ‌های ارسالی کوئری استرینگ‌ها را پاک کرده و همچنین تگ‌های خطرناک موجود در مقادیر ارسالی فرم‌های برنامه را هم به صورت خودکار حذف کند. اما هنوز اجازه بدهد تا کاربران بتوانند تگ HTML هم ارسال کنند.
مشکل! در ASP.NET مقادیر ارسالی کوئری استرینگ‌ها و همچنین فرم‌ها به صورت NameValueCollection در اختیار برنامه قرار می‌گیرند و ... خاصیت IsReadOnly این مجموعه‌ها در حین ارسال، به صورت پیش فرض true است و همچنین غیرعمومی! یعنی به همین سادگی نمی‌توان عملیات تمیزکاری را روی مقادیر ارسالی، پیش از مهیا شدن آن جهت استفاده در برنامه اعمال کرد. بنابراین در ابتدای کار نیاز است با استفاده از قابلیت Reflection ، اندکی در سازوکار داخلی ASP.NET دست برد، این خاصیت فقط خواندنی غیرعمومی را برای مدت کوتاهی false کرد و سپس مقصود نهایی را اعمال نمود. پیاده سازی آن را در ادامه مشاهده می‌کنید:

```
using System;
using System.Collections.Specialized;
using System.Reflection;
using System.Text.RegularExpressions;
using System.Web;
using Microsoft.Security.Application;


namespace AntiXssMdl
{
  public class AntiXssModule : IHttpModule
  {
      private static readonly Regex _cleanAllTags = new Regex("<[^>]+>", RegexOptions.Compiled);
      public void Init(HttpApplication context)
      {
          context.BeginRequest += CleanUpInput;
      }


      public void Dispose()
      { }


      private static void CleanUpInput(object sender, EventArgs e)
      {
          HttpRequest request = ((HttpApplication)sender).Request;
          if (request.QueryString.Count > 0)
          {
              //تمیزکاری مقادیر کلیه کوئری استرینگ‌ها پیش از استفاده در برنامه
              CleanUpAndEncode(request.QueryString, allowHtmltags: false);
          }


          if (request.HttpMethod == "POST")
          {
              //تمیزکاری کلیه مقادیر ارسالی به سرور
              if (request.Form.Count > 0)
              {
                  CleanUpAndEncode(request.Form, allowHtmltags: true);
              }
```

```
        }
    }


    private static void CleanUpAndEncode(NameValueCollection collection, bool allowHtmltags)
    {
        //اندکی دستکاری در سیستم داخلی دات نت
        PropertyInfo readonlyProperty = collection
                                            .GetType()
                                            .GetProperty("IsReadOnly",
                                                         BindingFlags.Instance |
BindingFlags.NonPublic);
        readonlyProperty.SetValue(collection, false, null);//IsReadOnly=false


        for (int i = 0; i < collection.Count; i++)
        {
            if (string.IsNullOrWhiteSpace(collection[i])) continue;


            if (!allowHtmltags)
            {
                //در حالت کوئری استرینگ دلیلی برای ارسال هیچ نوع تگی وجود ندارد
                collection[collection.Keys[i]] =
                   AntiXss.HtmlEncode(_cleanAllTags.Replace(collection[i], string.Empty));
            }
            else
            {
                //قصد تمیز سازی ویوو استیت را نداریم چون در این حالت وب فرم‌ها از کار می‌افتند
                if (collection.Keys[i].StartsWith("__VIEWSTATE")) continue;
                //در سایر موارد کاربران مجازند فقط تگ‌های سالم را ارسال کنند و مابقی حذف می‌شود
                collection[collection.Keys[i]] = Sanitizer.GetSafeHtml(collection[i]);
            }
        }


        readonlyProperty.SetValue(collection, true, null);//IsReadOnly=true
    }
  }
}
```

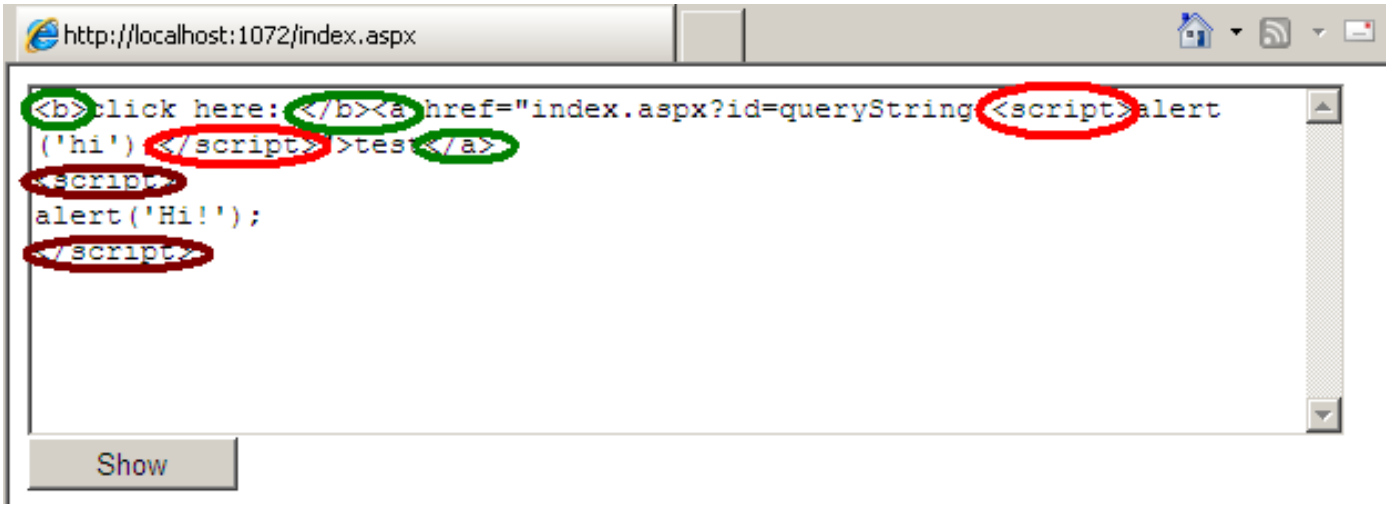
در این کلاس از کتابخانه AntiXSS مایکروسافت استفاده شده است. آخرین نگارش آن‌را از اینجا دریافت نمائید. نکته مهم آن متد Sanitizer.GetSafeHtml است. به کمک آن با خیال راحت می‌توان در یک سایت، از یک ادیتور متنی پیشرفته استفاده کرد. کاربران هنوز می‌توانند تگ‌های HTML را ارسال کنند؛ اما در این بین هرگونه سعی در ارسال عبارات و تگ‌های حاوی حملات XSS پاکسازی می‌شود.

و یک وب کانفیگ نمونه برای استفاده از آن به صورت زیر می‌تواند باشد (تنظیم شده برای IIS6 و 7):

```
<?xml version="1.0"?>
<configuration>
<system.web>
  <pages validateRequest="false" enableEventValidation="false" />
  <httpRuntime requestValidationMode="2.0" />
  <compilation debug="true" targetFramework="4.0" />
  <httpModules>
    <add name="AntiXssModule" type="AntiXssMdl.AntiXssModule"/>
  </httpModules>
</system.web>


<system.webServer>
  <validation validateIntegratedModeConfiguration="false"/>
  <modules>
    <add name="AntiXssModule" type="AntiXssMdl.AntiXssModule"/>
  </modules>
</system.webServer>
</configuration>
```

برای مثال به تصویر زیر دقت کنید. ماژول فوق، فقط تگ‌های سبز رنگ را (حین ارسال به سرور) مجاز دانسته، اسکریپت ذیل لینک را کلا حذف کرده و تگ‌های موجود در کوئری استرینگ را هم نهایتا (زمانیکه در اختیار برنامه قرار می‌گیرد) حذف خواهد کرد.



دریافت نسخه جدید و نهایی این مثال

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۳۰ ۱۰:۰۰:۰۳

جهت تکمیل بحث،
موارد زیر هم باید به لیست صرفنظر شونده‌ها اضافه شوند (همانند همان سطر ViewState که در کد آمده)

,LASTFOCUS, __EVENTTARGET__
,EVENTARGUMENT, __VIEWSTATE, __SCROLLPOSITIONX, __SCROLLPOSITIONY__
VIEWSTATEENCRYPTED, __ASYNCPOST__

این مورد برای ASP.NET Webforms ضروری است اما برای ASP.NET MVC خیر.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۳۰ ۱۰:۴۴:۰۱

مثال رو بر همین اساس به روز کردم که از همان آدرس قبلی آن قابل دریافت است.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۲/۳۱ ۱۰:۴۱:۳۷

مثال رو مجددا به روز کردم. تمیزکاری کوکی‌ها هم لحاظ شد. کلا این ماژول به ورودی‌های کاربر (کوکی‌ها (که با ابزار قابل تغییر هستند)، کوئری استرینگ‌ها و مقادیر ورودی در فرم‌ها حین ارسال به سرور) حساسیت دارد.

نویسنده: Ramin
تاریخ: ۱۳۹۰/۰۳/۰۱ ۱۴:۴۱:۳۰

مثل همیشه بی نقص و عالی بود ، واقعا بابت به اشتراک گذاری دانسته هایتان متشکرم

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۰۲ ۱۹:۲۵:۲۰

کتابخانه امنیتی مایکروسافت خطوط جدید را حذف می‌کند: (+)
به همین جهت مثال مجددا به روز شد تا این مشکل وجود نداشته باشد

نویسنده: مقدسی
تاریخ: ۱۳۹۰/۰۳/۰۶ ۱۱:۳۲:۳۵

واقعآ متشکرم و خسته نباشید .

نویسنده: Orion
تاریخ: ۱۳۹۰/۰۳/۰۷ ۱۵:۱۷:۵۰

جناب نصیری. خیلی ممنون از این مطلب ارزشمندت. خیلی دنبالش بودم. فقط به یه مشکلی در Webformها برخوردم. اونم اینه که شما اگر کاراکتر ampersand در مثلاً تکست‌باکستون داشته باشید، استفاده از وب ماژول باعث خراب شدن نتیجه میشه. چیکارش میشه کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۰/۰۳/۰۷ ۱۸:۳۸:۰۰

کاری که ماژول مایکروسافت انجام می‌ده علاوه بر حذف موارد زائد، تبدیل متن به XHTML استاندارد است و همچنین اعمال انواع و اقسام encoding؛ به همین جهت این نوع تبدیلات رو شما مشاهده می‌کنید ولی ... مطلوب کار ما نیست. در کل به خاطر این مسایل من ماژول مایکروسافت رو کنار گذاشتم (هر چند از لحاظ تشخیص حملات عالی است اما فعلا قابل تنظیم نیست و یک ضرب هر کاری که دوست دارد انجام می‌دهد). از یک روش دیگر استفاده کردم که سبک‌تر است و این مشکلات را هم ندارد (+) . این روش بر اساس white list عمل می‌کند. یعنی می‌گه یک سری تگ html از نظر من مجاز است و مابقی خطرناک‌ها همه باید حذف شوند.
مثال به روز شد لطفا آن‌را دریافت کنید.

نویسنده: Milad Bahari
تاریخ: ۱۳۹۰/۰۳/۲۵ ۱۳:۱۲:۰۸

بسیار عالی، می دونید مشکل اصلی اینجاست که توسعه دهنده ها دو قانون ساده رو فراموش می کنند. الف) پاک سازی ورودی و سپس استفاده ب) پاک سازی خروجی و سپس استفاده

نویسنده: جلال
تاریخ: ۱۳۹۱/۰۳/۲۹ ۱۳:۳۸

سلام
بابت به اشتراک گذاری سورس این ماژول ممنون. اما گویا این ماژول نشت حافظه داره. شما در تابع Dispose اشاره گر به تابع رو حذف نکردید. از اونجایی که IModule اشاره گری به HttpApplication نداره بنابراین من اونو از تابع Init گرفتم

```
private HttpApplication _context = null;
public void Init(HttpApplication context)
{
   _context = context;
   _context.BeginRequest += CleanUpInput;
}
public void Dispose()
{
    _context.BeginRequest -= CleanUpInput;
}
```

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۳/۲۹ ۱۳:۴۹

طول عمر HttpModule با طول عمر HttpApplication یکی است. به این معنا که تنها یک وهله از آن در زمان آغاز برنامه وب تولید و این وهله به صورت خودکار در زمان پایان عمر برنامه وب (ری استارت شدن سرور، recycle شدن آن توسط IIS و مواردی از این دست)، dispose خواهد شد. بنابراین ضرورتی به پیاده سازی متد Dispose در اینجا وجود ندارد.
اگر این مدیریت طول عمر خودکار نمی‌بود، بله ... بهتر بود که اینکار انجام می‌شد.

نویسنده: جلال
تاریخ: ۱۳۹۱/۰۳/۲۹ ۱۴:۳

ممنون. تابحال HttpModule ننوشته بودم و از این موضوع خبر نداشتم. به هرحال تمرین خوبیه که همیشه رویدادها رو پس از اتمام کار در این نوع موارد Unsubscribe کنیم.

نویسنده: هادی دانش پژوه
تاریخ: ۱۳۹۱/۰۵/۲۳ ۱۲:۲۸

با سلام من تو ارسال نظرات کدی مینوسم که عبارت Script از متنی که کاربر ارسال کرده حذف بشه! آیا همین کافی نیست؟

نویسنده: وحید نصیری

تاریخ: ۱۳۹۱/۰۵/۲۳ ۱۲:۳۶

خیر. موضوع پیچیده‌تر از این بررسی‌های ساده ابتدایی است. اطلاعات بیشتر (+) در مورد ورودی‌های پیچیده‌تر.

نویسنده: علی قشقایی
تاریخ: ۱۳۹۱/۰۵/۲۳ ۱۳:۸

در سطر اول فرمودید "همه این قابلیت را خاموش می‌کنند! ". این قابلیت رو چطور میشه خاموش کرد؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۳ ۱۳:۱۵

به احتمال زیاد تابحال در برنامه‌های وب خودتون نیاز به ارسال html به همراه تگ‌های آن نداشتید، ولی در کل:
- غیرفعال کردن validateRequest در سطح یک صفحه

```
<%@ Page
Language="C#"
        AutoEventWireup="true"
        CodeBehind="editpage.aspx.cs"
        validateRequest="false"
        Inherits="MyProject.UI.editpage" %>
```

- در سطح کل برنامه

```
<system.web>
    <pages validateRequest="false" enableEventValidation="true" />
    <httpRuntime requestValidationMode="2.0" />
</system.web>
```

البته asp.net mvc از این لحاظ پیشرفته‌تر است؛ چون اجازه می‌ده در سطح یک خاصیت فقط این بررسی رو خاموش کرد با بکارگیری ویژگی AllowHtml آن.

نویسنده: علی قشقایی
تاریخ: ۱۳۹۱/۰۵/۲۳ ۱۵:۵۲

در فایل مثالی که قرار دادید از کتابخانه AntiXSS استفاده نشده!

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۲۳ ۱۶:۴۲

توضیح دادم اینجا

نویسنده: saman
تاریخ: ۱۳۹۱/۰۷/۱۹ ۹:۲۶

سلام جناب نصیری خیلی عالی بود. فقط یه سوال و درخواست داشتم . اگر براتون مقدور هست کد VB.net رو هم اینجا بذارین.
من کد رو تبدیل کردم ولی وقتی استفاده می‌کنم این پیغام رو میده :
("<A potentially dangerous Request.Form value was detected from the client (TextBox1="<script.

در حالی که Validate request رو False کردم.
و اینکه این کد روی VB9 اجرا نمیشه چون Collection رو ساپورت نمیکنه ولی VB10 مشکل نداره. راه حلی برای این موضوع هم دارین ؟
ممنونم ازتون

موفق *باشید*

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۹ ۱۳:۲۲

این ماژول و کدهای اون رو به روز کردم و از اینجا قابل دریافت است:
HtmlCleaner.zip

تفاوت‌ها:
- برای دات نت سه و نیم کامپایل شده. فقط فایل dll رو به پروژه خودتون cs یا vb اضافه کنید.
- متد ToSafeHtml کلاس HtmlSanitizer برای کار با تگ‌های مشخص شده با حروف کوچک و بزرگ بهبود یافته (الان در همین سایت جاری استفاده می‌شود).
- الزامی نیست حتما از AntiXssModule آن استفاده کنید. کلا هرجایی که Allow Html دارید، متد ToSafeHtml را برای پاکسازی اطلاعات فراخوانی کنید (در MVC و یا در وب فرم‌ها).
- کلاس PersianProofWriter هم به آن اضافه شده (جزئی از ToSafeHtml است). یک سری از مسایل مانند نیم فاصله‌ها رو به صورت خودکار اصلاح می‌کند؛ به همراه اصلاح ی و ک فارسی.

به صورت خلاصه:
فقط از متد ToSafeHtml کلاس HtmlSanitizer آن به صورت دستی و در موارد لازم که HTML از کاربر دریافت می‌شود، استفاده کنید.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۹ ۱۳:۳۰

- توضیحات رفع خطای فوق در اینجا: (^)
- نگارش DLL این پروژه که قابل استفاده در VB هم هست بدون نیاز به تبدیل کدها در اینجا: (^)

نویسنده: اردوان دژپناه
تاریخ: ۱۳۹۲/۰۹/۰۹ ۱۶:۶

در ورژن AntiXss Library4.2 2013 برای تنظیمات فقط باید در فایل Web.config کد زیر را اضافه کرد
در بخش system.web

```
<httpRuntime executionTimeout="180" encoderType="Microsoft.Security.Application.AntiXssEncoder,
AntiXssLibrary" />
```

نویسنده: ایران من
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۸:۱۲

با سلام؛ میشه لطف کنید و تو یه پروژه نحوه استفاده از این ماژول رو آموزش بدین ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۸:۴۲

پروژه IRIS از HtmlCleaner استفاده کرده.

نویسنده: شهاب
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۹:۴

ممنون. اما من میخوام تو صفحاتم از tinymce استفاده کنم. ابته تو winforms 4.5 نه mvc. اینو چه کنم.

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۹:۱۰

- HtmlCleaner وابستگی به MVC ندارد.
- اصلا ماژول نیست و نیازی نیست داخل web.config ثبت شود.
- یکبار باید کامنت‌ها را کامل مطالعه کنید تا علت رسیدن به HTML Cleaner نهایی مشخص شود.
+ کمی بالاتر توضیح دادم «فقط از متد ToSafeHtml کلاس HtmlSanitizer آن به صورت دستی و در موارد لازم که HTML از کاربر دریافت می‌شود، استفاده کنید. »

نویسنده: شهاب
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۹:۱۱

در واقع زمانی که در حال دریافت اطلاعات از tinymce هستم بایستی مقدار دریافتی را به این روش چک کنم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۵ ۱۹:۱۳

بله. با استفاده از متد ToSafeHtml تمیز و بعد در بانک اطلاعاتی ذخیره‌اش کنید.

نویسنده: محمد
تاریخ: ۱۳۹۲/۰۹/۲۵ ۲۱:۳۵

سلام؛ در دات نت 4 به بعد که AntiXSS معرفی شد ، آیا این رفرنس اضافه شده به طور خودکار اینکار را انجام نمیدهد؟ با وجود کدی که شما زحمت کشیدین ، پس کار AntiXss چی هست ؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۹/۲۵ ۲۱:۴۳

- موتور سیستم اعتبارسنجی خودکار ورودی‌ها در ASP.NET قابل تغییر است (حالا چه نگارش قدیمی آن باشد، چه AntiXss جدید). اما این موتور به صورت پیش فرض به هر نوع تگی عکس العمل نشان می‌دهد. یعنی شما نمی‌توانید متن HTMLایی ارسال کنید بدون خاموش کردن یا تغییر بسیاری از پیش‌فرض‌های آن (مقدمه بحث). مثلا در ASP.NET MVC با استفاده از ویژگی AllowHtml می‌شود فقط یک خاصیت را جهت ارسال HTML مجاز دانست (از این لحاظ نسبت به Webforms بهتر عمل می‌کند؛ چون مجبور نیستید به ازای یک صفحه یا حتی کل سایت این اعتبارسنجی را خاموش کنید).
- با استفاده از کلاس‌ها و متدهای AntiXss هم می‌شود دستی HTML را تمیز کرد و خروجی امن گرفت (کاری که ابتدا در کدهای متن انجام شده بود و هنوز هم قابل مشاهده است). اما این خروجی تهیه شده توسط AntiXss، یک سری اضافات و دخل و تصرف‌هایی دارد که خوشایند نیست. کامنت‌های مطلب را از ابتدا مطالعه کنید تا به سیر منطقی رسیدن به کتابخانه Clean HTML تهیه شده برسید.