

应用程序安全性

2019年3月4日 21:01

■ 客户端IP白名单限制:

- 1、通过中间件，检查每个请求的远程IP地址
- 2、检查特定控制器或Action操作请求的远程IP地址
- 3、基于 Razor 页面的应用，检查 Razor 页面请求的远程IP地址

■ 重定向攻击

UrlHelperBase 类中的 IsLocalUrl 实现

来自 <<https://github.com/aspnet/AspNetCore/blob/master/src/Mvc/Mvc.Core/src/Routing/UrlHelperBase.cs>>

```
private IActionResult RedirectToLocal(string returnUrl)
{
    if (Url.IsLocalUrl(returnUrl))
    {
        return Redirect(returnUrl);
    }
    else
    {
        return RedirectToAction(nameof(HomeController.Index),
"Home");
    }
}
```

■ 跨站脚本 XSS 注入攻击

跨站点脚本 (XSS) 是一个安全漏洞，这会使第三方攻击者将 JS 代码注入到当前网页，当其他用户加载页面时，脚本将执行，攻击者可窃取 cookie、获取令牌 和 执行非法操作。

保护防御 XSS 应用程序

- 永远不会将不受信任的数据放入 HTML 标签内。
- 将数据放入 HTML 标签内时，首先进行 HTML 编码，JS脚本要进行脚本编码。

使用 Razor 语法进行 HTML 编码

```
var untrustedInput = "<\"123\">";
```

```
@untrustedInput
```

使用 Razor 的 JavaScript 编码

```
@using System.Text.Encodings.Web;
@inject JavaScriptEncoder encoder;

@{
    var untrustedInput = "<\`123\`>";
}

<script>
    document.write("@encoder.Encode(untrustedInput)");
</script>
```

在后台代码中使用 HTML、JavaScript 和 URL 编码器

```
public class HomeController : Controller
{
    HtmlEncoder _htmlEncoder;
    JavaScriptEncoder _javascriptEncoder;
    UrlEncoder _urlEncoder;

    public HomeController(HtmlEncoder htmlEncoder,
                           JavaScriptEncoder
javascriptEncoder,
                           UrlEncoder
urlEncoder)
    {
        _htmlEncoder = htmlEncoder;
        _javascriptEncoder = javascriptEncoder;
        _urlEncoder = urlEncoder;
    }
}

var example = "\"Quoted Value with spaces and &\"";
var encodedValue = _urlEncoder.Encode(example);
```

自定义编码器

默认情况下编码器限制为 Unicode 范围的安全字符，此行为影响 Razor TagHelper 和 HtmlHelper 呈现，因为它将使用编码器输出字符串。

<p>This link text is in Chinese: @Html.ActionLink("汉语/漢語", "Index")</p>

```
services.AddSingleton<HtmlEncoder>(
    HtmlEncoder.Create(allowedRanges: new[]
    { UnicodeRanges.BasicLatin,

        UnicodeRanges.CjkUnifiedIdeographs }));
```

默认情况下直接使用：System.Text.Encodings.Web.*Encoder.Default 类型编码

■ 防止跨站点请求伪造 (XSRF/CSRF) 攻击

微软 Antiforgery 防伪标记支持库

来自 <<https://github.com/aspnet/AspNetCore/tree/master/src/Antiforgery>>

防卫标记请求流程

- 1、服务器端生成防伪标记并发送到客户端
- 2、客户端提交表单时将防伪标记发送到服务器进行验证
- 3、服务器收到访问标记后进行验证，如果非法则拒绝请求

何是生成表单防伪标记？

满足以下任何一个条件，自动生成的防伪令牌，需要 [FormTagHelper](#) 支持。

- 操作属性为空 (action="")。
- 操作属性不提供 (<form method="post">)。

常见访问标记生成的写法

```
<form asp-controller="Manage" asp-action="ChangePassword" method="post">
    ...
</form>
```

```
@using (Html.BeginForm("ChangePassword", "Manage"))
{
    ...
}
```

```
<form action="/" method="post">
    @Html.AntiForgeryToken()
</form>
```

```
<input name="__RequestVerificationToken" type="hidden" value="CfDJ8NrAkS">
```

如何禁用防伪标记?

显式禁用防伪令牌，且asp-antiforgery属性

```
<form method="post" asp-antiforgery="false">
    ...
</form>
```

使用感叹号排除标记帮助程序

```
<!form method="post">
    ...
</!form>
```

从视图中删除 FormTagHelper 标记

```
@removeTagHelper Microsoft.AspNetCore.Mvc.TagHelpers.FormTagHelper,
Microsoft.AspNetCore.Mvc.TagHelpers
```

自定义防伪 AntiforgeryOptions 选项

```
services.AddAntiforgery(options =>
{
    // Set Cookie properties using CookieBuilder properties†.
    options.FormFieldName = "AntiforgeryFieldname";
    options.HeaderName = "X-CSRF-TOKEN-HEADERNAME";
    options.SuppressXFrameOptionsHeader = false;
});
```

选项	描述
Cookie	确定用于创建防伪 cookie 的设置。
FormFieldName	防伪系统用于呈现防伪令牌在视图中的隐藏的窗体字段的名称。
HeaderName	防伪系统使用的标头的名称。如果null，系统会认为只有窗体数据。
SuppressXFrameOptionsHeader	指定是否禁止显示生成X-Frame-Options标头。默认情况下，值为"SAMEORIGIN"生成标头。默认为 false。

手动生成防伪标识

```
public void Configure(IApplicationBuilder app, IAntiforgery antiforgery)
{
```

```

app.Use(next => context =>
{
    // The request token can be sent as a JavaScript-readable
cookie,
    // and Angular uses it by default.
    var tokens = antiforgery.GetAndStoreTokens(context);
    context.Response.Cookies.Append("XSRF-TOKEN",
tokens.RequestToken);

    return next(context);
});
}

```

服务端验证防伪标记

支持任何 HTTP 谓词方法

```

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> RemoveLogin(RemoveLoginViewModel account)
{
}

```

仅支持 POST 谓词，不支持：GET、HEAD、OPTIONS 和 TRACE

```

[Authorize]
[AutoValidateAntiforgeryToken]
public class ManageController : Controller
{
}

```

```

services.AddMvc(options =>
    options.Filters.Add(new AutoValidateAntiforgeryTokenAttribute()));

```

可通过 IgnoreAntiforgeryToken 忽略防伪标记验证。

```

[AutoValidateAntiforgeryToken]
public class ManageController : Controller
{
    [HttpPost]
    [IgnoreAntiforgeryToken]
    public async Task<IActionResult> DoSomethingSafe(SomeViewModel
model)
    {
        // no antiforgery token required
    }
}

```

```
}
```

使用 JavaScript 通过 AJAX 提交防伪令牌

```
xhttp.setRequestHeader("X-CSRF-TOKEN", csrfToken)
```

```
services.AddAntiforgery(options => options.HeaderName = "X-XSRF-TOKEN");
```

防伪标记扩展性

[IAntiForgeryAdditionalDataProvider](#)