内容安全策略（CSP）：每个Web开发人员都必须知道的

这是一个全面的内容安全策略（CSP）指南。如果您是一名web开发者，CSP是一个重要的概念，需要了解，以保护您的用户免受跨站点脚本（XSS）注入攻击。

在本文中，我们将学习（几乎）关于内容安全策略（CSP）的所有内容。即使你以前从未听说过CSP，也不知道它是什么，我保证，在文章的最后，你会有一个坚实的理解和具体的，一步一步的计划，以保护您的网站和应用程序免受跨站点脚本（XSS）注入攻击。

问题：跨站点脚本如何注入

大多数Web应用程序从数据库中获取数据，将其插入到HTML模板并发送到浏览器。攻击者通过网站上的表单提交恶意JavaScript到数据库中。在下一个请求中，服务器将错误的脚本发送到浏览器，浏览器会认为该脚本来自您的服务器。

让我们想象一下，您是一名黑客，想要窃取电子商务网站用户的信用卡号码。

你是怎么办到的

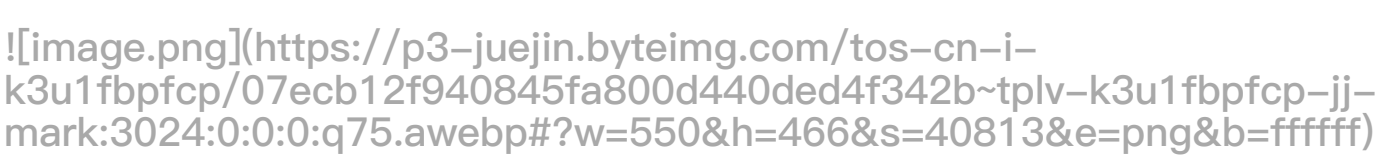
您可以做的一件事是编写一些JavaScript，在用户输入其信用卡详细信息时记录支付页面上的每笔交易。

由于浏览器执行页面上的任何JavaScript，而JavaScript可以读取、写入或修改网站的任何部分，如果您能找到一种方法在电子商务网站上插入您的糟糕的JavaScript代码，那么用户的信用卡号码就不再安全了。

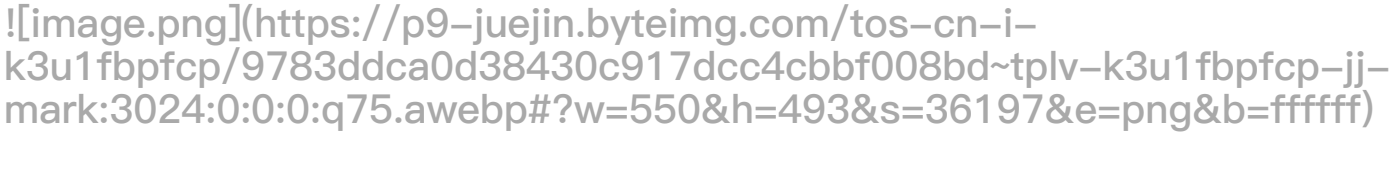
但是浏览器要执行这个JavaScript，它是网站代码的一部分。

你怎么能在电子商务网站上获得这种糟糕的JavaScript代码，以便用户的浏览器在他们付款时执行它？

你注意到这个电子商务网站上有一个表单，任何人都可以添加对产品的评论。所以你创建了一个帐户，而不是一个真实的评论，你在文本区域键入以下代码。

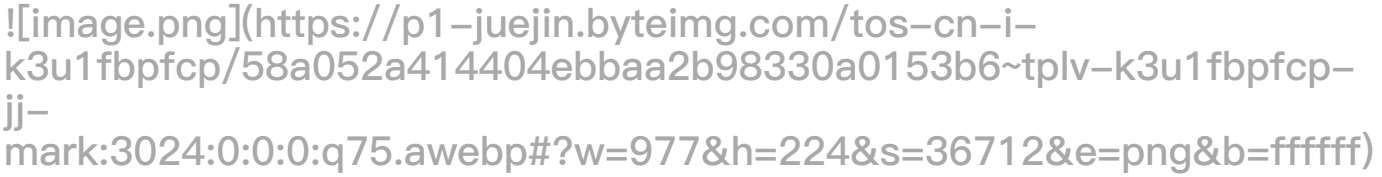


点击提交按钮后，您会看到弹出的警告窗口，显示您刚刚输入的消息。

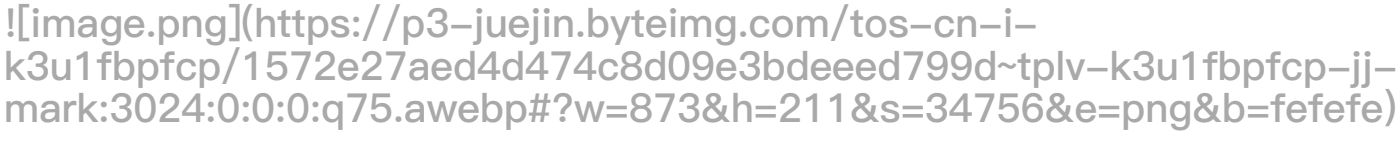


怎么会这样？

在你提交评论后，浏览器将虚假评论发送到电子商务网站的后端服务器，后者将其以纯文本形式保存在数据库中。



接下来，在对该请求的响应中，后端将您重定向到相同的产品页面，该页面刷新并从后端获取相同的虚假评论。后端脚本立即从数据库中获取存储的评论并将其发送到浏览器。



在收到包含虚假评论的HTML时，浏览器认为这是您想要执行的脚本，并立即执行它，并向您显示警告框。

更重要的是，由于评论是公开的，并且对网站的每个用户都是可见的，因此包含注入脚本的相同HTML将发送给每个其他用户，用相同的警报迎接他们！

您编写了一个精心设计的脚本，记录用户的点击，并发出一个获取请求，将所有数据发布到您的其他网站，这将保存所有这些信息，以便您稍后利用。或者，您可以通过访问用户的会话信息来完全劫持用户的会话，这样您就可以以该用户的身份远程登录。

然后，您以相同的方式将其插入数据库，在产品页面上添加另一个虚假评论。

恭喜您，您成功实现了跨站点脚本攻击！

攻击者可以利用站点脚本漏洞攻击从数据库获取用户提交数据的任何页面。攻击者将向数据库中注入错误的JavaScript代码，这些代码稍后将由网站获取，并在用户访问网站时由用户的浏览器执行。

XSS是最常见的安全漏洞。攻击者可以尝试将此脚本注入他们可以提交表单的任何地方，例如文章的标题或正文、产品评论或评论。如果网站不阻止XSS，可以完全访问网站上的数据。

现在从电子商务网站开发者的角度来思考。如何缓解这种攻击?

解决方案一：转义HTML字符

防止跨站点脚本攻击的一个简单解决方案是获取所有用户提交的文本，并将其中的任何特殊控制字符（如 `<` ， `>` 等）替换为相应的实体编码。这被称为字符转义。

例如，HTML

```
<script>alert('hello world')</script>
```

```

转换为

```
<script>alert(’hello world’)</script>
```

当浏览器看到转义的内容时，它会像对待普通文本一样识别并呈现它，但不会像对待HTML标记一样对待它。换句话说，它不会执行转义的 `&lt;script&gt;` 标记中的代码，这正是我们想要的。

通常，您需要在HTML的上下文中转义以下字符，以便它们添加任何特殊行为。

1. 小于符号（<） `&amp;lt;`
2. 大于符号（>），`&amp;gt;`
3. 双引号（“），`&amp;quot;`
4. 单引号（’），`&amp;#39;`
5. (&），`&amp;amp;`

通常，您不需要担心转义特殊字符，因为大多数现代Web框架都会为您处理它。

在许多情况下，确保用户提交的HTML内容被正确地转义需要很长的路要走。

解决方案二：内容安全策略
---

*CSP背后的基本思想是阻止内联脚本执行，并提供允许的可信内容源列表（脚本，样式表，字体，插件等）。到浏览器。即使攻击者注入了他们的坏脚本，浏览器也不会执行它，因为CSP阻止了内联脚本的执行。*

XSS攻击的根本原因是浏览器能够执行HTML中包含的任何和所有内联JavaScript。如果有一种方法可以告诉浏览器不要执行任何内联JavaScript，以及任何从外部域加载的脚本，那会怎么样?
```

您可以使用HTTP响应上的 `Content-Security-Policy` 头来实现这一点，它规定了内容的安全策略。从本质上讲，这个header做了两件事：

1. 指示浏览器阻止所有内联脚本执行
2. 提供一个安全域列表，从那里加载外部资源，防止所有其他来源。

虽然它通常用于脚本，但它也控制其他资源，如样式表，字体，插件等。

实现严格的CSP可以防止浏览器从任何其他位置获取脚本。这使得很难在你的网站中注入糟糕的JavaScript代码。即使他们以某种方式注入它，浏览器也不会执行它，因为它是内联的。

所有现代浏览器都支持CSP。由于跨站点脚本攻击是由于浏览器上不必要的JavaScript执行而发生的，因此锁定所有外部和内联JavaScript对防止XSS攻击大有帮助。

当您的网站实现严格的CSP时，攻击者将无法在网站上运行任何错误的JavaScript。

那么，它是如何工作的？

这是一个标准CSP头的例子。

```
Content-Security-Policy: script-src 'self' https://safe-external-site.com; style-src 'self'
```

在上面的头文件中，术语 `script-src` 和 `style-src` 是分别为JavaScript和样式表指定有效源的指令。

这两个指令都告诉浏览器不要执行任何内联JavaScript或样式表。

1. 脚本策略告诉浏览器只运行从同一个域（ `self` ）或从域 `safe-external-site.com` 获取的脚本。
2. 样式策略告诉它只允许来自相同域的样式表。不允许使用外部样式表。

您也可以在 `<head>` 元素下的 `<meta>` 标记中提供策略，而不是使用HTTP头。

下面是通过 `<meta>` 标签提供的内容安全策略。

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self' https://safe-external-site.com">
```

“unsafe-inline”允许您绕过CSP

如果出于某种原因，您必须允许内联JavaScript或样式，请在相应的指令上使用 `unsafe-inline` 值。例如，以下策略允许所有内联JavaScript，因此违背了CSP的基本目的。

```
Content-Security-Policy: script-src 'self' 'unsafe-inline' https://safe-external-site.com
```

一些重要的CSP指令

到目前为止，我们已经看到了 `script-src` 和 `style-src` ，它们为脚本和样式表指定了有效的源。这里还有一些其他重要的属性。

* `default-src` ： 定义获取所有资源的默认策略。当缺少更具体的策略时，浏览器将回退到 `default-src` 策略指令。示例： `default-src 'self' trusted-domain.com`
* `img-src` ： 定义图像的有效来源，例如 `img-src 'self' img.mydomain.com`
* `font-src` ： 定义字体资源的有效来源。
* `object-src` ： 定义插件和外部内容的有效源，如或元素。
* `media-src` ： 定义音频和视频的有效源。

当您确实需要内联样式和样式时，请使用nonce属性

术语 `nonce` 指的是只使用一次。

在CSP的上下文中， `nonce` 是 `<script>` 和 `<style>` 标记上的一个属性，它允许特定的内联 `script` 和 `style` 元素，同时仍然阻止所有其他内联脚本和样式。这可以让你避免使用 `unsafe-inline` 指令，它允许所有的内联执行，是不安全的。

使用 `nonce` 属性可以让您保持CSP的基本优势，同时允许您执行特定内联元素。

**它是如何工作的？ **

* 对于每个请求，服务器都会创建一个无法猜测的随机base64编码字符串。这是随机数值，例如 `dGhpcyBpcyBhIG5v==`
* 服务器会将此nonce值插入到您希望允许的所有内联脚本和样式元素中，作为 `nonce` 属性的值。

```
<script nonce="dGhpcyBpcyBhIG5v==">.. ..</script>

<style nonce="dGhpcyBpcyBhIG5v==">.. ..</style>
```

* 服务器还在 `script-src` 和 `style-src` 指令的 `Content-Security-Policy` 头中插入相同的nonce值。

```
Content-Security-Policy: script-src 'nonce-dGhpcyBpcyBhIG5v=='; style-src 'nonce-dGhpcyBpcyBhIG5v=='
```

由于攻击者无法猜测令牌，因此他们无法注入错误的JavaScript。

总而言之，特定脚本和样式元素上的 `nonce` 属性指示浏览器允许这些元素中的内联执行，同时仍然阻止所有其他没有 `nonce` 属性的内联脚本和样式。

它告诉浏览器这些元素不是黑客注入的（因为他们无法猜测nonce值），而是服务器故意插入的，所以它们可以安全执行。

采用CSP：从报告开始，但不强制执行CSP违规行为

如果你有一个巨大的遗留代码库，到处都是PHP/HTML文件中的内联脚本，那该怎么办？最重要的是，您不知道有多少地方使用了内联脚本和外部脚本。怎么办？

由于CSP默认情况下阻止所有内联脚本并阻止所有外部源，因此您必须删除/重构所有内联JavaScript和外部源，或者在策略中显式允许它们。

如果您立即启用严格的策略来阻止所有内联或外部脚本和样式表，那么您就有麻烦了。浏览器既不会加载外部脚本/样式，也不会执行内联脚本。

有一个优雅的解决方案来解决这个难题。您可以使用仅报告模式，而不是立即设置严格的CSP并冒着生产中出现任何崩溃的风险。

使用 `Cotent-Security-Policy-Report-Only` 头，您可以告诉浏览器只报告，但不执行策略。

因此，浏览器仍将加载外部脚本并执行任何内联JavaScript，但它也将使用 `report-to` 指令向策略中定义的端点报告所有违规。

```
Content-Security-Policy-Report-Only: script-src 'self'; report-to https://mysite.com/csp-violations
```

端点可以是应用程序中的一个路由，您可以在其中编写代码来解析违规数据并将其保存到数据库中，或者您可以使用第三方服务，如 `report-uri.com`，它可以在浏览器的表中很好地显示违规数据，并支持过滤和排序。

如何实现严格的CSP
----

1. 从 `Content-Security-Policy-Report-Only` 头开始，迭代地处理策略。密切违规报告，并在出现问题时进行修复，直到不再有违规报告。
2. 重构代码以删除内联JavaScript，并将其移动到在 `<script>` 元素上使用 `src` 属性加载的JavaScript文件中。重构包含内联事件处理程序（如 `onclick` 和 `javascript:` URI）的HTML。
3. 对于必须执行内联脚本或使用内联CSS的地方，在服务器上生成 `nonce` 令牌，将其放在脚本和样式元素的 `nonce` 属性上。
4. 彻底测试。在部署到生产环境之前，让您的测试团队了解CSP，并要求他们注意控制台中的任何CSP违规报告。

无论如何，将JavaScript与HTML分离是一种最佳实践，因为它可以清理HTML并保持代码库的组织性。

这里有几个例子，说明你将进行的更改类型。

* **将随机数添加到 `<script>` 和 `<style>` 元素**

```
<script src=”my_script.js”></script>
<script>alert(’hello world’)</script>

<style>
  body {
      background-color: green;
  }
</style>
```

成为:

```
<script nonce=”token” src=”my_script.js”></script>
<script nonce=”token”>alert(’hello’)</script>
```

```
<style nonce="token">
  body {
      background-color: green;
  }
</style>
```

重构内联JavaScript处理程序和URI

通常，旧网站可以包含使用 `onclick` 或 `onerror` 回调的事件处理程序。它们容易受到XSS攻击。重构它，以便从JavaScript代码块添加处理程序。理想情况下，您应该将其移动到单独的JavaScript文件中。

```
<script>
    function handle() {
      // click handler code
    }
</script>

<button onclick="handle();">Click Me</button>
```

成为:

```
<button id="submit-btn">Click Me</button>

<script nonce="token">
document.addEventListener('DOMContentLoaded', function () {
  document.getElementById('submit-btn')
          .addEventListener('click', () => {
            // click handler code
          });
});
</script>
```

重要的是要记住，策略是按页面定义的。

原文： [Content Security Policy (CSP): Everything You Should Know (writesoftwarewell.com)](http://cxyroad.com/”https://www.writesoftwarewell.com/content-security-policy/”)

原文链接: https://juejin.cn/post/7352146376632467490