



Google Search

I'm Feeling Lucky

先知社区

## 前言

2018年9月26日,开源[Closure](#)库(最初由谷歌创建并用于谷歌搜索)的一名开发人员创建了一个提交,删除了部分输入过滤。据推测,这是因为开发人员在用户界面设计方面出

2019年2月,安全研究员Masato

[Kinugawa](#)发现了这个漏洞,并将其报告给了Google。Google立即做出反应,并在2019年2月22日修复了漏洞,撤销了之前9月份做的修改。另一位安全专家[LiveOverflow](#)

## XSS如何发生的?

Closure库中的漏洞很难检测。它依赖于一种很少使用的技术,叫做突变XSS。突变XSS漏洞是由浏览器解释HTML标准的方式不同造成的。

由于浏览器的不同,很难对服务器上的用户输入进行清理。服务器不仅需要考虑浏览器之间的所有差异,还需要考虑它们的版本之间的所有差异。对输入进行过滤以防止XSS。有一个很好的客户端库用于防止XSS: DOMPurify。这个库也被Closure使用。然而, DOMPurify并不是万能法宝。在极少数情况下,需要额外的过滤操作才能确保万无一失。

```
9 closure/goog/html/sanitizer/safedomtreeprocessor.js View file
@@ -108,12 +108,9 @@ SafeDomTreeProcessor.prototype.processToString = function(html) {
108     newRoot.appendChild(newTree);
109     newTree = newRoot;
110 }
111 - // The XMLSerializer will add a spurious xmlns attribute to the root node.
112 - var serializedNewTree = new XMLSerializer().serializeToString(newTree);
113 - // Remove the outer span before returning the string representation of the
114 - // processed copy.
115 - return serializedNewTree.slice(
116     serializedNewTree.indexOf('>') + 1, serializedNewTree.lastIndexOf('</')
117 );
118 + // Serialized string of the sanitized DOM without root span tag
119 + return newTree.innerHTML;
120 /**
```

## DOMPurify的工作机制

DOMPurify使用template元素清除用户输入。浏览器以不同的方式处理div元素的innerHTML属性和template元素的相同属性。对于div元素,innerHTML在赋值后立即

DOMPurify的工作机制是获取用户输入,将其分配给模板元素的innerHTML属性,让浏览器解释它(但不执行它),从而避免潜在的XSS。然而正是因为这种操作,有可能导致

## 浏览器如何解释无效的HTML？

若要了解浏览器如何解释无效的HTML，请创建一个仅包含以下内容的HTML文档：

```
<div><script title="</div>">
```

当您在浏览器中打开它时，您将看到代码被解释为如下所示：

```
<html>
<head></head>
<body>
<div>
<script title="</div>"></script>
</div>
</body>
</html>
```

现在，尝试创建一个包含以下内容的HTML文档：

```
<script><div title="</script>">
```

浏览器以不同方式解释此代码：

```
<html>
<head>
<script><div title="</script>
</head>
<body>
">
</body>
</html>
```

造成这种差别的原因是，当浏览器遇到<script>标记时，它会从HTML解析器切换到JavaScript解析器，直到找到结束标记。

## noscript背后的邪恶

在大多数情况下，浏览器将始终以与环境无关的相同方式解释相同的文档。但是，有一种情况下，由于某些客户端环境的原因，这种行为可能会有所不同：这就是<noscript>。HTML规范规定，根据浏览器中是否启用JavaScript，必须对<noscript>标签进行不同的解释。这种不同的浏览器行为正是Masato Kinugawa的XSS Poc的关键一步。结果是，当将无效的HTML代码分配给template元素的innerHTML属性时(禁用了JavaScript)，以及分配给div元素的innerHTML属性时(启用了JavaScript)，

## PoC

payload

```
<noscript><p title="</noscript><img src=x onerror=alert(1)>">
```

如果禁用JavaScript,浏览器将以下方式解释有效负载：

```
<noscript>
<p title="</noscript><img src=x onerror=alert(1)>"></p>
</noscript>
```

```
</noscript><img src=x
onerror=alert(1)>
```

是title参数的值。因此，DOMPurify不会过滤输入内容，因为没有JavaScript，DOMPurify默认不存在XSS风险。

DOMPurify 1.0.10 "Dragonfish"

bower package 1.0.10 . npm package 1.0.10 . build passing

This is the demo for [DOMPurify](#), a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, SVG and MathML. The textarea below contains sample-payload - you can also add your own. Watch it sanitize on the console or in the Iframe below.

Sanitize textarea value, then write result to console

Sanitize textarea value, then write result to DOM

Sanitize textarea value, then use \$(elm).html()

Timings: 1ms 1ms 1ms 1ms 2ms

Dirty HTML

```
<noscript><p title="</noscript><img src=x onerror=alert(1)>">
```

Clean HTML

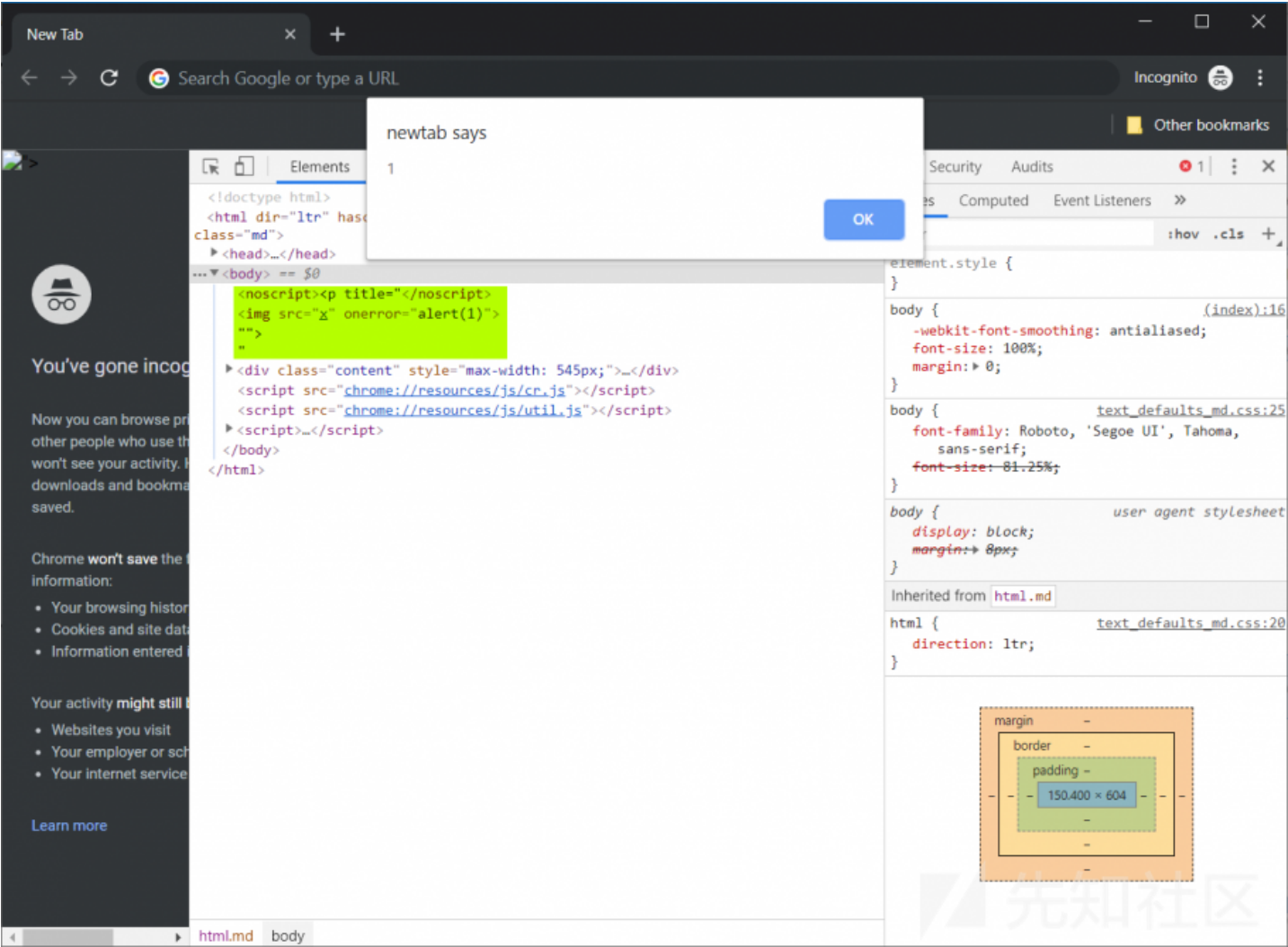
```
<p title="</noscript><img src=x onerror=alert(1)>"></p>
```

但是，如果启用了JavaScript(对于浏览器使用的div元素)，浏览器将以下方式解释payload：

```
<noscript><p title="</noscript>

" ">
"
```

noscript元素提前结束，img元素被完整地解释，包括onerror属性的JavaScript内容。



结论

很难说以前是否有其他人发现此漏洞，以及该漏洞是否已经用于任何恶意目的。由于Closure库也用于其他Google产品，因此此漏洞可能会影响Gmail，地图，文档和其他服务。

https://www.acunetix.com/blog/web-security-zone/mutation-xss-in-google-search/?chat\_vt=1

点击收藏 | 1 关注 | 1

上一篇：ROIS CISCN 全国大学生信... 下一篇：ROIS CISCN 全国大学生信...

1. 0 条回复
- 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

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