pdfunite of poppler-0.55.0 call stack exhaustion vulnerability due to program logic problem

Overview

This is a vulnerability in the program logic of the **pdfunite** util of poppler-0.55.0(linux), which will cause the program fall into a recursive and interactive call of two functions and eventually exhaust the stack space due to a large call stack. This vulnerability can cause Denial-of-Service attack.

Software and Environments

Software: poppler-0.55.0

Operating System: Ubuntu 14.04 x86_64 Desktop

```
pengjiaqi@ubuntu:~/Documents/crash/poppler-0.55.0/build-gcc/utils$ uname -a
Linux ubuntu 3.13.0-32-generic #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
```

Compiler: gcc

```
pengjiaqi@ubuntu:~/Documents/crash/poppler-0.55.0/build-gcc/utils$ gcc --version
gcc (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
```

Reproducing

The crash can be reproduced in the following way:

```
cd /* path of poppler source code */
mkdir build-gcc; cd build-gcc
../autogen.sh –disable-shared
make
./utils/pdfunite    /* path of PoC file */    1.pdf
```

Exception

```
pengjiaqi@ubuntu:~/Documents/crash/poppler-0.55.0/build-gcc/utils$ ./pdfunite PoC.pdf 1.pdf
Segmentation fault
```

Analysis

The crash situation is:

```
   0x7ffff6777b89 <_int_malloc+9>:      push   rbx
   0x7ffff6777b8a <_int_malloc+10>:     sub    rsp,0xb8
   0x7ffff6777b91 <_int_malloc+17>:     cmp    rsi,0xffffffffffffffbf
=> 0x7ffff6777b95 <_int_malloc+21>:     mov    QWORD PTR [rsp+0x8],rsi
   0x7ffff6777b9a <_int_malloc+26>:     ja     0x7ffff6778321 <_int_malloc+1953>
   0x7ffff6777ba0 <_int_malloc+32>:     mov    rax,rsi
   0x7ffff6777ba3 <_int_malloc+35>:     mov    ebp,0x20
   0x7ffff6777ba8 <_int_malloc+40>:     mov    rbx,rdi
[----------------------------------stack-----------------------------------]
Invalid $SP address: 0x7fffff7feff0
[--------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
_int_malloc (av=0x7ffff6ab9760 <main_arena>, bytes=0x5) at malloc.c:3302
3302    malloc.c: No such file or directory.
```

Then check the stack space of current process:

```
gdb-peda$ info proc mapping
process 75623
Mapped address spaces:

          Start Addr           End Addr       Size     Offset objfile
      0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
      0x7fffff800000     0x7fffffffe000   0x7ff000        0x0 [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

It is obvious that rsp (0x7fffff7feff0) in crash is smaller than the top of the whole stack, which is 0x7fffff800000, which causes this crash.

First, have a look at the call stack.
There are more than 60000 functions in crash stack. Here are some functions in the start and end of the crash stack:

```
gdb-peda$ bt 11
#0  _int_malloc (av=0x7ffff6ab9760 <main_arena>, bytes=0x5) at malloc.c:3302
#1  0x00007ffff677a6d0 in __GI___libc_malloc (bytes=0x5) at malloc.c:2891
#2  0x000000000047ad0e in gmalloc (checkoverflow=0x0, size=<optimized out>) at ../../goo/gmem.cc:110
#3  gmalloc (size=<optimized out>) at ../../goo/gmem.cc:120
#4  0x000000000047b2d2 in copyString (s=0x823990 "Page") at ../../goo/gmem.cc:316
#5  0x0000000000454009 in Object::copy (this=<optimized out>, obj=obj@entry=0x7fffff7ff170)
    at ../../poppler/Object.cc:109
#6  0x00000000004130a8 in Dict::getValNF (this=this@entry=0x823470, i=i@entry=0x0, obj=obj@entry=0x7fffff7ff170)
    at ../../poppler/Dict.cc:299
#7  0x000000000045c53d in PDFDoc::markDictionnary (this=0x822d20, dict=0x823470, xRef=0x83ef00,
    countRef=countRef@entry=0x83ff50, numOffset=0x0, oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3)
    at ../../poppler/PDFDoc.cc:1572
#8  0x000000000045c87d in PDFDoc::markObject (this=this@entry=0x822d20, obj=0x7fffff7ff280,
    xRef=xRef@entry=0x83ef00, countRef=countRef@entry=0x83ff50, numOffset=numOffset@entry=0x0,
    oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3) at ../../poppler/PDFDoc.cc:1599
#9  0x000000000045c560 in PDFDoc::markDictionnary (this=0x822d20, dict=0x823470, xRef=0x83ef00,
    countRef=countRef@entry=0x83ff50, numOffset=0x0, oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3)
    at ../../poppler/PDFDoc.cc:1572
#10 0x000000000045c87d in PDFDoc::markObject (this=this@entry=0x822d20, obj=0x7fffff7ff390,
    xRef=xRef@entry=0x83ef00, countRef=countRef@entry=0x83ff50, numOffset=numOffset@entry=0x0,
    oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3) at ../../poppler/PDFDoc.cc:1599
```

```
#61641 0x000000000045c560 in PDFDoc::markDictionnary (this=0x822d20, dict=0x823470, xRef=0x83ef00,
    countRef=countRef@entry=0x83ff50, numOffset=0x0, oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3)
    at ../../poppler/PDFDoc.cc:1572
#61642 0x000000000045c87d in PDFDoc::markObject (this=this@entry=0x822d20, obj=0x7fffffffd990,
    xRef=xRef@entry=0x83ef00, countRef=countRef@entry=0x83ff50, numOffset=numOffset@entry=0x0,
    oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3) at ../../poppler/PDFDoc.cc:1599
#61643 0x000000000045c560 in PDFDoc::markDictionnary (this=0x822d20, dict=0x823470, xRef=0x83ef00,
    countRef=countRef@entry=0x83ff50, numOffset=0x0, oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3)
    at ../../poppler/PDFDoc.cc:1572
#61644 0x000000000045c87d in PDFDoc::markObject (this=this@entry=0x822d20, obj=obj@entry=0x7fffffffdaa0,
    xRef=xRef@entry=0x83ef00, countRef=countRef@entry=0x83ff50, numOffset=numOffset@entry=0x0,
    oldRefNum=oldRefNum@entry=0x3, newRefNum=newRefNum@entry=0x3) at ../../poppler/PDFDoc.cc:1599
#61645 0x000000000045caa6 in PDFDoc::markPageObjects (this=0x822d20, pageDict=pageDict@entry=0x823470,
    xRef=xRef@entry=0x83ef00, countRef=countRef@entry=0x83ff50, numOffset=0x0, oldRefNum=0x3, newRefNum=0x3)
    at ../../poppler/PDFDoc.cc:1713
#61646 0x000000000040bd8e in main (argc=argc@entry=0x3, argv=argv@entry=0x7fffffffddc8)
    at ../../utils/pdfunite.cc:375
#61647 0x00007ffff6719f45 in __libc_start_main (main=0x40b6e0 <main(int, char**)>, argc=0x3, argv=0x7fffffffddc8,
    init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>, stack_end=0x7fffffffddb8)
    at libc-start.c:287
#61648 0x000000000040d099 in _start ()
```

Here, PDFDoc::markObject() and PDFDoc::makeDictionnary() are calling each other. Then in the last time of calling makeDictionnary(), the stack is exhausted, then the program crashes!
So, we analysis following this sequence:
main() -> markPageObjects() -> markObject() -> markDictionary()

in main():

```
365       Dict *pageDict = page.getDict();
366       Dict *resDict = docs[i]->getCatalog()->getPage(j)->getResourceDict();
367       if (resDict) {
368         Object *newResource = new Object();
369         newResource->initDict(resDict);
370         pageDict->set("Resources", newResource);
371         delete newResource;
372       }
373       pages.push_back(page);
374       offsets.push_back(numOffset);
375       docs[i]->markPageObjects(pageDict, yRef, countRef, numOffset, refPage->num, refPage->num);
```

in markPageObjects():

```
1698 void PDFDoc::markPageObjects(Dict *pageDict, XRef *xRef, XRef *countRef, Guint numOffset,
     int oldRefNum, int newRefNum)
1699 {
1700   pageDict->remove("OpenAction");
1701   pageDict->remove("Outlines");
1702   pageDict->remove("StructTreeRoot");
1703
1704   for (int n = 0; n < pageDict->getLength(); n++) {
1705     const char *key = pageDict->getKey(n);
1706     Object value; pageDict->getValNF(n, &value);
1707     if (strcmp(key, "Parent") != 0 &&
1708         strcmp(key, "Pages") != 0 &&
1709         strcmp(key, "AcroForm") != 0 &&
1710         strcmp(key, "Annots") != 0 &&
1711         strcmp(key, "P") != 0 &&
1712         strcmp(key, "Root") != 0) {
1713       markObject(&value, xRef, countRef, numOffset, oldRefNum, newRefNum);
1714     }
1715     value.free();
1716   }
1717 }
```

crash happeds when n=2

value = pageDict.entries[2].val

The first argument of markObject() is &value and value comes from page->getValNF(n, &value) =>
value = pageDict.entries[n].val = **pageDict.entries[2].val**

in markObject():

```
1585 void PDFDoc::markObject(Object* obj, XRef *xRef, XRef *countRef, Guint numOffset, int oldRefNum,
     int newRefNum)
1586 {
1587   Array *array;
1588   Object obj1;
1589
1590   switch (obj->getType()) {
1591     case objArray:
1592       array = obj->getArray();
1593       for (int i=0; i<array->getLength(); i++) {
1594         markObject(array->getNF(i, &obj1), xRef, countRef, numOffset, oldRefNum, newRefNum);
1595         obj1.free();
1596       }
1597       break;
1598     case objDict:
1599       markDictionnary(obj->getDict(), xRef, countRef, numOffset, oldRefNum, newRefNum);
1600       break;
```

The first argument of makeDictionnary() is
obj->getDict() = value.dict = pageDict.entries[2].val.dict

in markDictionnary():

```
1566 void PDFDoc::markDictionnary (Dict* dict, XRef * xRef, XRef *countRef, Guint numOffset, int oldRe
     fNum, int newRefNum)                          dict = pageDict.entries[2].val.dict = pageDict
1567 {
1568   Object obj1;
1569   for (int i=0; i<dict->getLength(); i++) {
1570     const char *key = dict->getKey(i);
1571     if (strcmp(key, "Annots") != 0) {
1572       markObject(dict->getValNF(i, &obj1), xRef, countRef, numOffset, oldRefNum, newRefNum);
1573     } else {
1574       Object annotsObj;                  when i=2, due to dict = pageDict,
1575       dict->getValNF(i, &annotsObj);     the calling is the same as before  ->  loop !
1576       if (!annotsObj.isNull()) {
1577         markAnnotations(&annotsObj, xRef, countRef, 0, oldRefNum, newRefNum);
1578         annotsObj.free();
1579       }
1580     }
1581     obj1.free();
1582   }
1583 }
```

dict = pageDict.entries[2].val.dict

the first argument of markObject() is dict->getValNF(i, &obj1), which is dict.entries[i].val

After analysis, we get dict=pageDict (which will be analyzed later),

then when i=2, the argument will be **pageDict.entries[2].val**, which is the same as the last call of markObject().

So, the program will fall into a loop call of above several functions and won't terminate!

After above analysis, we know that **the root cause is:**

dict = **pageDict.entries[2].val.dict = pageDict** (which can be verified as below)

```
gdb-peda$ p pageDict
$178 = (Dict *) 0x823470
```

```
gdb-peda$ p pageDict.entries[2].val.dict
$181 = (Dict *) 0x823470
```

Then ,we analyze how pageDict comes from:

```
367       if (resDict) {
368         Object *newResource = new Object();
369         newResource->initDict(resDict);
370         pageDict->set("Resources", newResource);
371         delete newResource;
372       }
373       pages.push_back(page);
374       offsets.push_back(numOffset);
375       docs[i]->markPageObjects(pageDict, yRef, countRef, numOffset, refPage->num, refPage->num);
```

pageDict->set("Resources",newResource) will copy newResource.dict (=resDict) to pageDict.entries[i].val.dict whose pageDict.entries[i].key=="Resources".

Here, pageDict.entries[2].key="Resources":

```
gdb-peda$ p pageDict.entries[2].key
$182 = 0x83ea80 "Resources"
```

So, it will set pageDict.entries[2].val.dict to be resDict.

According to the shotscreen below, we know resDict = pageDict:

```
gdb-peda$ p resDict
$184 = (Dict *) 0x823470
```

then pageDict.entries[2].val.dict = pageDict

So, **the cause becomes: resDict = pageDict**.

```
354   for (i = 0; i < (int) docs.size(); i++) {
355     for (j = 1; j <= docs[i]->getNumPages(); j++) {
356       PDFRectangle *cropBox = NULL;
357       if (docs[i]->getCatalog()->getPage(j)->isCropped())
358         cropBox = docs[i]->getCatalog()->getPage(j)->getCropBox();
359       docs[i]->replacePageDict(j,
360         docs[i]->getCatalog()->getPage(j)->getRotate(),
361         docs[i]->getCatalog()->getPage(j)->getMediaBox(), cropBox);
362       Ref *refPage = docs[i]->getCatalog()->getPageRef(j);
363       Object page;
364       docs[i]->getXRef()->fetch(refPage->num, refPage->gen, &page);
365       Dict *pageDict = page.getDict();
366       Dict *resDict = docs[i]->getCatalog()->getPage(j)->getResourceDict();
```

pageDict = docs[0].xref.entries[docs[0].catalog.pageRefs[0]->num]->obj.dict
resDict = docs[0].catalog.pages[0].attrs.resources.dict

Both of pageDict and resDict come from docs[0], which is built from the PoC file. However, how the two variables are related to the PoC file is too complicated to analyze, which we omit here. But, it is certain that we can craft a pdf file which will make pageDict and resDict equal to each other and then will cause an infinite call of two functions, leading to an exhaustion of call stack and eventually program crash.

Author

Name: Jiaqi Peng of VARAS@IIE
Organization: IIE (http://iie.ac.cn/)