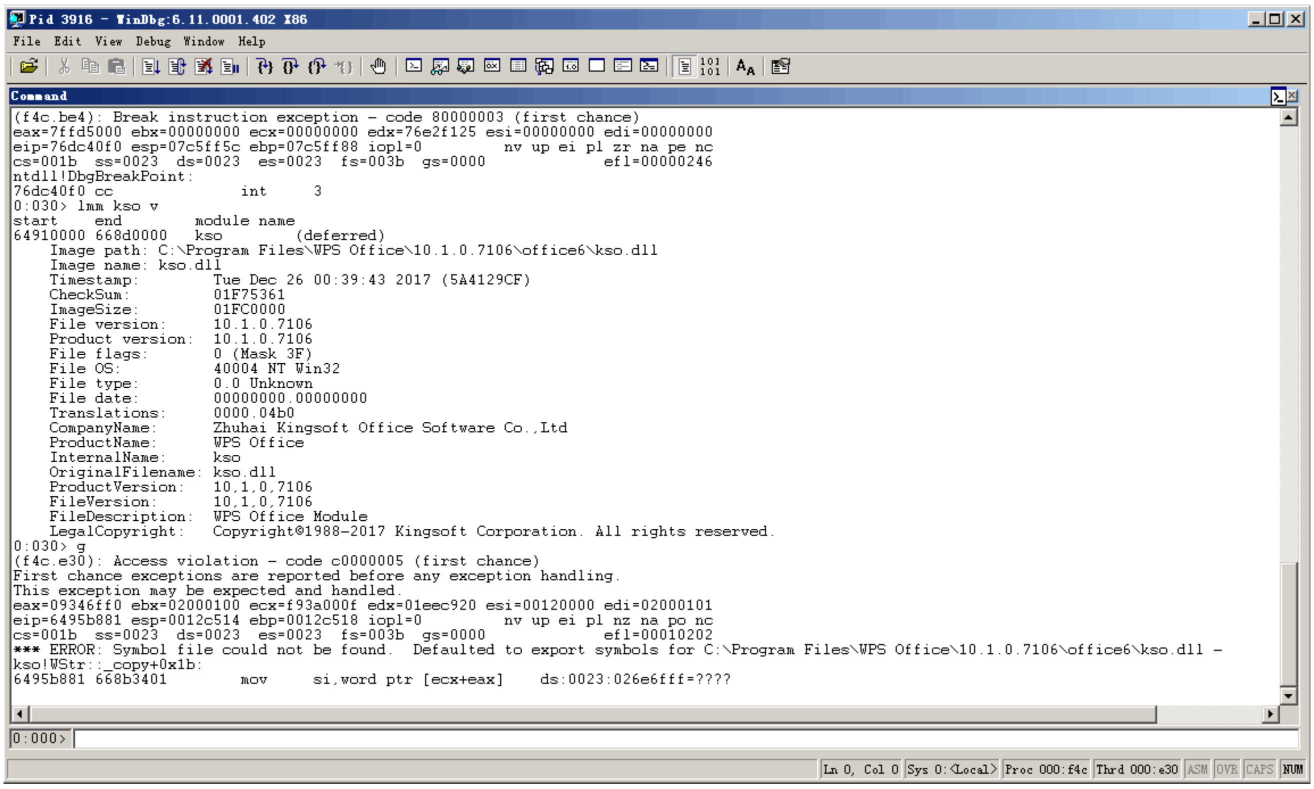
An issue was discovered in WPS Office

Here is an issue in WPS Office 10.2.0.5978 and 10.1.0.7106, and possibly have impacted other versions.

Remote attackers could leverage this vulnerability to cause a denial of service (application crash) via a crafted (a) web page, (b) office document, or (c) .rtf file.

It was discovered in the module of kso.dll.



```
(f4c.be4): Break instruction exception - code 80000003 (first chance)
eax=7ffd5000 ebx=00000000 ecx=00000000 edx=76e2f125 esi=00000000 edi=00000000
eip=76dc40f0 esp=07c5ff5c ebp=07c5ff88 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
ntdll!DbgBreakPoint:
76dc40f0 cc              int     3
0:030> lmm kso v
start    end        module name
64910000 668d0000   kso        (deferred)
    Image path: C:\Program Files\WPS Office\10.1.0.7106\office6\kso.dll
    Image name: kso.dll
    Timestamp:        Tue Dec 26 00:39:43 2017 (5A4129CF)
    CheckSum:         01F75361
    ImageSize:        01FC0000
    File version:     10.1.0.7106
    Product version:  10.1.0.7106
    File flags:       0 (Mask 3F)
    File OS:          40004 NT Win32
    File type:        0.0 Unknown
    File date:        00000000.00000000
    Translations:     0000.04b0
    CompanyName:      Zhuhai Kingsoft Office Software Co.,Ltd
    ProductName:      WPS Office
    InternalName:     kso
    OriginalFilename: kso.dll
    ProductVersion:   10,1,0,7106
    FileVersion:      10,1,0,7106
    FileDescription:  WPS Office Module
    LegalCopyright:   Copyright©1988-2017 Kingsoft Corporation. All rights reserved.
0:030> g
(f4c.e30): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=09346ff0 ebx=02000100 ecx=f93a000f edx=01eec920 esi=00120000 edi=02000101
eip=6495b881 esp=0012c514 ebp=0012c518 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program Files\WPS Office\10.1.0.7106\office6\kso.dll -
kso!WStr::_copy+0x1b:
6495b881 668b3401        mov     si,word ptr [ecx+eax]    ds:0023:026e6fff=????
```

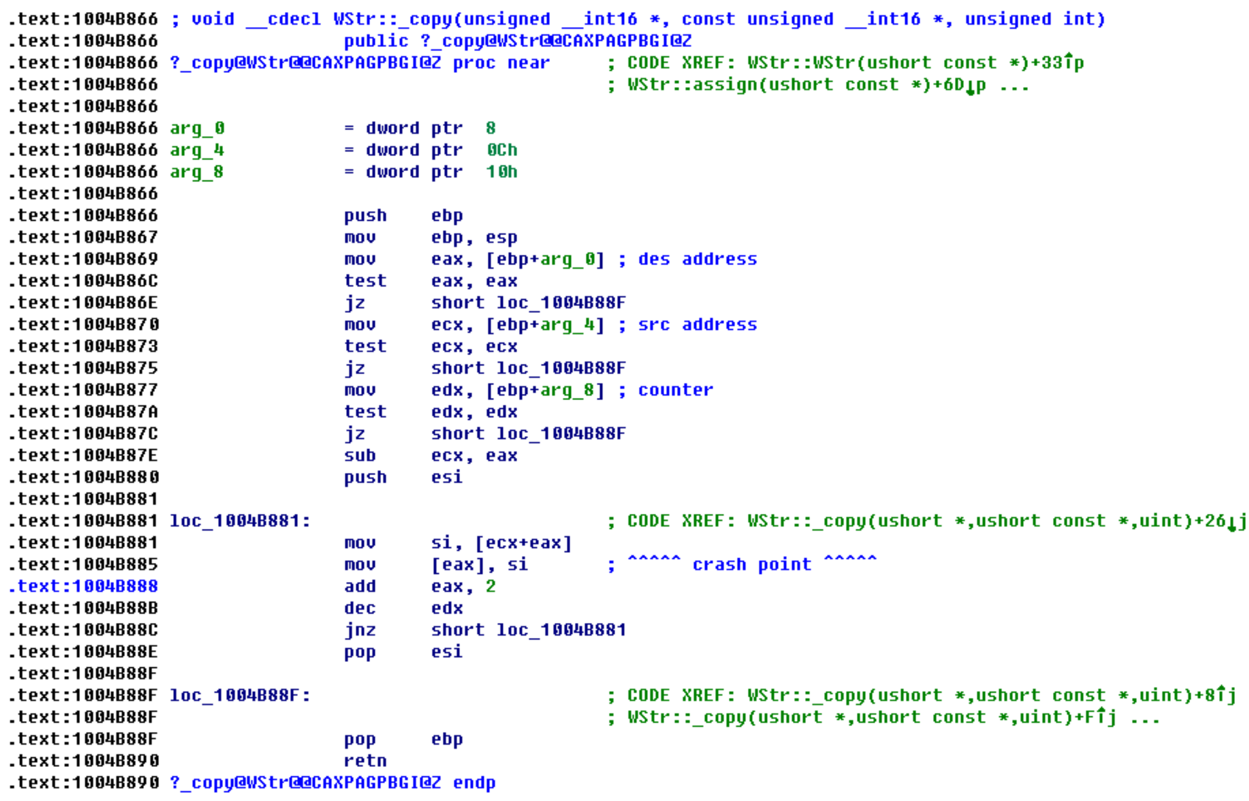
Looking at the stack of calls.

```
0:000> k
ChildEBP RetAddr
WARNING: Stack unwind information not available. Following frames may be wrong.
0012c518 64a19543 kso!WStr::_copy+0x1b
0012c538 6d2fb3d0 kso!WStr::assign+0x72
```

At the crash point, calculating its offset from the starting address of kso.dll.

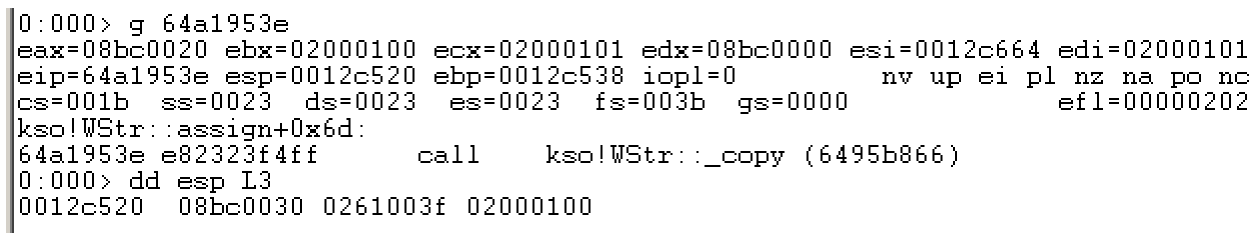
```
0:000> ? 6495b881 - kso
Evaluate expression: 309377 = 0004b881
```

In kso!WStr::_copy() function, arg_0 is the destination memory address, arg_4 is the source memory address, and arg_8 stands for the counter.



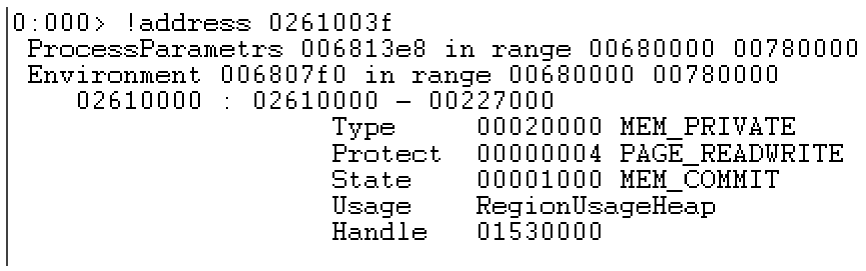
```
.text:1004B866 ; void __cdecl WStr::_copy(unsigned __int16 *, const unsigned __int16 *, unsigned int)
.text:1004B866                 public ?_copy@WStr@@CAXPAGPBGI@Z
.text:1004B866 ?_copy@WStr@@CAXPAGPBGI@Z proc near     ; CODE XREF: WStr::WStr(ushort const *)+33↑p
.text:1004B866                                         ; WStr::assign(ushort const *)+6D↓p ...
.text:1004B866
.text:1004B866 arg_0           = dword ptr  8
.text:1004B866 arg_4           = dword ptr  0Ch
.text:1004B866 arg_8           = dword ptr  10h
.text:1004B866
.text:1004B866                 push    ebp
.text:1004B867                 mov     ebp, esp
.text:1004B869                 mov     eax, [ebp+arg_0] ; des address
.text:1004B86C                 test    eax, eax
.text:1004B86E                 jz      short loc_1004B88F
.text:1004B870                 mov     ecx, [ebp+arg_4] ; src address
.text:1004B873                 test    ecx, ecx
.text:1004B875                 jz      short loc_1004B88F
.text:1004B877                 mov     edx, [ebp+arg_8] ; counter
.text:1004B87A                 test    edx, edx
.text:1004B87C                 jz      short loc_1004B88F
.text:1004B87E                 sub     ecx, eax
.text:1004B880                 push    esi
.text:1004B881
.text:1004B881 loc_1004B881:                           ; CODE XREF: WStr::_copy(ushort *,ushort const *,uint)+26↓j
.text:1004B881                 mov     si, [ecx+eax]
.text:1004B885                 mov     [eax], si       ; ^^^^^ crash point ^^^^^
.text:1004B888                 add     eax, 2
.text:1004B88B                 dec     edx
.text:1004B88C                 jnz     short loc_1004B881
.text:1004B88E                 pop     esi
.text:1004B88F
.text:1004B88F loc_1004B88F:                           ; CODE XREF: WStr::_copy(ushort *,ushort const *,uint)+8↑j
.text:1004B88F                                         ; WStr::_copy(ushort *,ushort const *,uint)+F↑j ...
.text:1004B88F                 pop     ebp
.text:1004B890                 retn
.text:1004B890 ?_copy@WStr@@CAXPAGPBGI@Z endp
```

When it calls kso!WStr::_copy() function, looking at the 3 arguments.



```
0:000> g 64a1953e
eax=08bc0020 ebx=02000100 ecx=02000101 edx=08bc0000 esi=0012c664 edi=02000101
eip=64a1953e esp=0012c520 ebp=0012c538 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
kso!WStr::assign+0x6d:
64a1953e e82323f4ff       call    kso!WStr::_copy (6495b866)
0:000> dd esp L3
0012c520  08bc0030 0261003f 02000100
```

Checking the size of the source block, as follows:

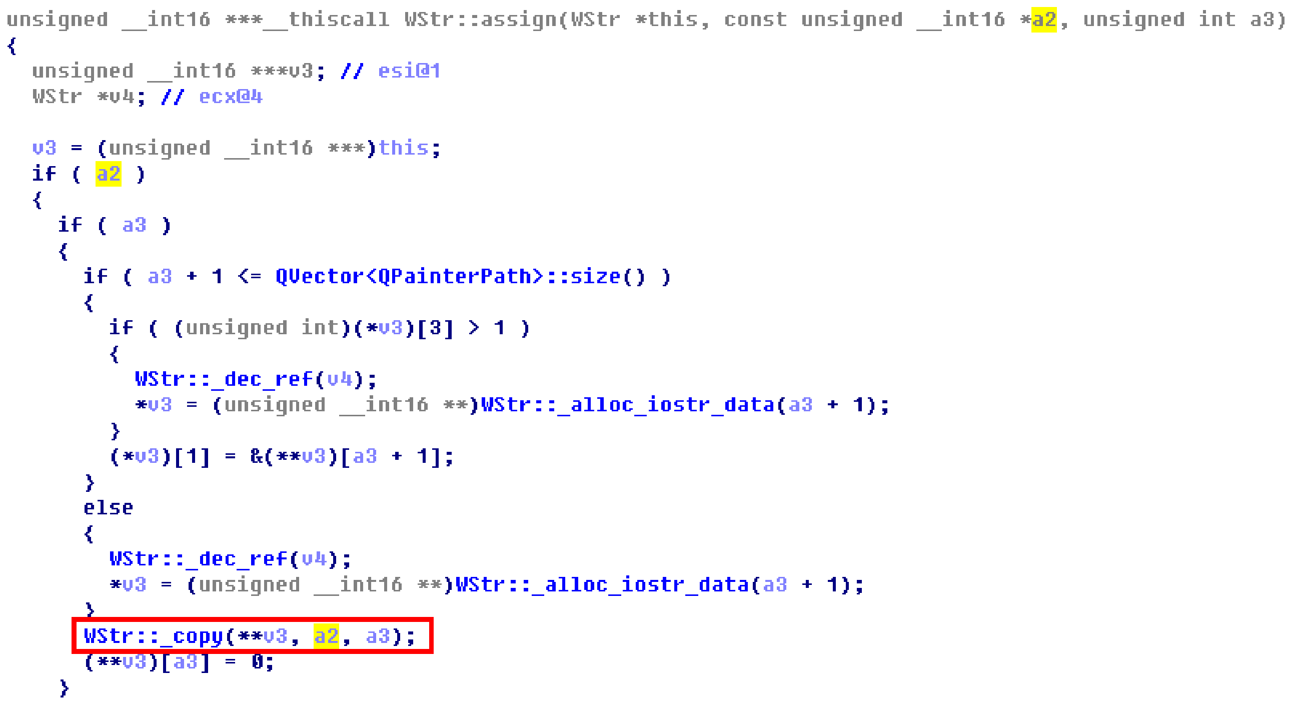


```
0:000> !address 0261003f
 ProcessParametrs 006813e8 in range 00680000 00780000
 Environment 006807f0 in range 00680000 00780000
    02610000 : 02610000 - 00227000
                    Type     00020000 MEM_PRIVATE
                    Protect  00000004 PAGE_READWRITE
                    State    00001000 MEM_COMMIT
                    Usage    RegionUsageHeap
                    Handle   01530000
```

The size of the source memory block is 0x00227000. The counter passing to kso!WStr::_copy() function is 0x02000100, and each copy needs 2 bytes. So the size of existence is much smaller than the requied.

In kso!WStr::_copy() function, when source space is exhausted, triggered access violation.

Finally, in a nutshell, the reason of the issue is that there is no valid check of the size of the source memory block before calling the kso!WStr::_copy() function, and the issue arises from kso!WStr::assign() function.



```
unsigned __int16 ***__thiscall WStr::assign(WStr *this, const unsigned __int16 *a2, unsigned int a3)
{
  unsigned __int16 ***v3; // esi@1
  WStr *v4; // ecx@4

  v3 = (unsigned __int16 ***)this;
  if ( a2 )
  {
    if ( a3 )
    {
      if ( a3 + 1 <= QVector<QPainterPath>::size() )
      {
        if ( (unsigned int)(*v3)[3] > 1 )
        {
          WStr::_dec_ref(v4);
          *v3 = (unsigned __int16 **)WStr::_alloc_iostr_data(a3 + 1);
        }
        (*v3)[1] = &(**v3)[a3 + 1];
      }
      else
      {
        WStr::_dec_ref(v4);
        *v3 = (unsigned __int16 **)WStr::_alloc_iostr_data(a3 + 1);
      }
      WStr::_copy(**v3, a2, a3);
      (**v3)[a3] = 0;
    }
```