


Wrapping the Converter within Foxit Reader

 thezdi.com/blog/2017/10/17/wrapping-the-converter-within-foxit-reader

2017年10月17日

SUBSCRIBE

PDF readers are an essential part of our daily workflow nowadays. Many of these readers support more features than just navigating a PDF. For example, a lot of the PDF readers support image conversion, including Adobe Acrobat Pro (DC) and Foxit Reader (and PhantomPDF). And by image conversion, I'm specifically referring to converting an image file to a PDF file.

While this feature is quite helpful, it does come with a security cost, and that price tag is pretty high. In the past few years, we at the ZDI have seen a spike in image parsing vulnerabilities targeting PDF readers, many of which include targeting the image conversion feature. Foxit Reader is one of the more interesting PDF readers as it is able to convert multiple image formats to PDF documents.

This blog talks about how to fuzz the image conversion feature of Foxit without fuzzing the whole application. Specifically, we'll demonstrate this by writing a small wrapper that allows us to fuzz faster and analyze cases quicker.

Foxit's Image Conversion

NOTE: All the analysis in this section was done with Foxit Reader V7.34.

When we first started getting image conversion vulnerability reports in Foxit, I noticed that most of the crashes happened inside `ConvertToPDF_x86.dll`, which is where most of the conversion/parsing happens.

For example, here's the initial crash of ZDI-17-039 (dump below from Foxit version 7.34):

```
0:000> r
eax=1bd76efc ebx=1bd61dd0 ecx=1bd6e000 edx=1bd76f27 esi=000000ff edi=000000d4
eip=54d74c5c esp=001e9eec ebp=001e9ef4 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
ConvertToPDF_x86!CreateFXPDFConvertor+0x2da16c:
54d74c5c 8a19          mov     bl,byte ptr [ecx]          ds:0023:1bd6e000=??
0:000> kb
# ChildEBP RetAddr  Args to Child
WARNING: Stack unwind information not available. Following frames may be wrong.
```

Forget about the actual vulnerability; we're mostly interested in how Foxit Reader communicates with `ConvertToPDF_x86.dll` .

Let's first have a look at the functions exported by `ConvertToPDF_x86.dll` and then start debugging from there:

Name	Address	Ordinal
CreateFXPDFConverter	000000001000AAF0	1
DestorFXPDFConverter	0000000010008E30	2
DllEntryPoint	00000000101496A6	[main entry]

CreateFXPDFConverter is pretty simple. It allocates an object of size `0x1BDCh` and then initializes it:

Here's the initialization:

In the debugger, the object returned from `CreateFXPDFConverter` looks like the following:

Setting breakpoints on the functions above and continuing execution ends up hitting the breakpoint on the second function:

The function takes a single argument. In this case, it's `0x2` . Note that in order to call this function properly, we'd need to pass the `this` pointer in ECX.

Continuing the execution, we end by hitting the breakpoint set at function `0x5062a2d0` :

This function takes an argument, which is the name of the printer device that Foxit installed.

Continuing the execution ends up breaking at `0x5062b060` :

The function takes three arguments. The first argument points to the path of the file to be parsed. The second and third arguments are set to zero.

Finally, continuing the execution triggers the crash:

Summarizing this process:

The `CreateFXPDFConverter()` method returns a pointer that points to a vtable that contains four methods. After the vtable is obtained, the following three calls are made:

Call 1: `this->method2(0x2)`

Call 2: `this->method3("Foxit Reader PDF Printer")`

Call 3: `this->method0(IMAGE_PATH,0,0)`

Putting it all together in Ctypes

A while back, my colleague Jasiel Spelman blogged about this topic and released Python code that helps to call arbitrary code from Python. The code can be used in this case to call functions and pass the `this` pointer in the ECX register.

So, here's how I implemented this in `ctypes`.

Load the library (clearly this part isn't rocket science):

Call `CreateFXPDFConverter` and grab the object returned:

Get the vtable address and store the methods:

Call `this->method2(0x2)` :

Call `this->method3("Foxit Reader PDF Printer")` :

And finally call `this->method0(IMAGE_PATH,0,0)` :

We can use the script implemented for fuzzing purposes. I plugged this in my own framework and you can notice the speed. Here's a short video demonstrating it:

Watch Video At: <https://youtu.be/Gf2eyPKswYY>

It's a lot faster than fuzzing the whole app, right?

Differences between `ConvertToPDF_x86.dll` in 7.3.4 and 8.3.1

Foxit has made some quiet changes to `ConvertToPDF_x86.dll`. For example, `ConvertToPDF_x86` now exports more functions:

Also, the vtable has more methods to explore.

The same principle should still apply for v8.3.1, with some minor issues to overcome. I'll likely cover this topic in future blog posts.

Conclusion

Fuzzing components of an application separately is definitely worthwhile. It's much faster, and at least from an analyst perspective, it makes case analysis easier to manage. The details applied in this blog post can be applied to other software, as well. We have implemented something similar for the Windows PDF Library that helps us analyze cases faster, rather than going through Edge. Keep following the ZDI blog for more analysis and tips as we continue our Foxit series.

Have you tried fuzzing individual application components? Curious how to take this method and apply it elsewhere? Share your experiences and ask any questions on twitter [@abdhariri](#), and follow the team for the latest in exploit techniques and security patches.

- Foxit

- [Research](#)
- [Techniques](#)

[BACK TO THE BLOG](#)

[Share](#)