16:05 Fast Cash for Useless Bugs!

by EA

Hello neighbors,

I come to you with a short story about useless crashes turned useful.

Every one of us who has ever looked at a piece of code looking for vulnerabilities has ended up finding a number of situations which are more than simple bugs but just a bit too benign to be called a vulnerability. You know, those bugs that lead to process crashes locally, but can't be exploited for anything else, and don't bring a remote server down long enough to be called a Denial Of Service.

They come in various shapes and sizes from simple `assert()`s being triggered in debug builds only, to null pointer dereferences (on certain platforms), to recursive stack overflows and many others. Some may be theoretically exploitable on obscure platform where conditions are just right. I'm not talking about those here, those require different treatment.[12]

The ones I'm talking about are the ones we are dead sure can't be abused and by that virtue might have quite a long life. I'm talking about all those hundreds of thousands of null pointer dereferences in MS Office that plagued anybody who dared fuzz it, about unbounded recursions in PDF renderers, and infinite loops in JavaScript engines. Are they completely useless or can we squeeze just a tiny bit of purpose from their existence?

As I advise everybody should, I've been keeping these around, neatly sorting them by target and keeping track of which ones died. I wouldn't say I've been stockpiling them, but it would be a waste to just throw them away, wouldn't it?

Anyway, here are some of my uses for these useless crashes – including a couple of examples, all dealing with file formats, but you can obviously generalize.

Testing Debug/Fuzzing Harness The first use I came up with for long lived, useless crashes in popular targets is testing debugging or fuzzing harnesses. Say I wrote a new piece of code that is supposed to catch crashes in Flash that runs in the context of a browser. How can I be sure my tool actually catches crashes if I don't have a proper crashing testcase to test it with?

Of course CDB catches this, but would your custom harness? It's simple enough to test. From a standpoint of a debugger, crashing due to null pointer dereference or heap overflow is the same. It's all an "Access Violation" until you look more closely – and it's always better to test on the actual thing than on a synthetic example.

```
# cdb flashplayer_26_sa.exe flash_crasher.swf
CommandLine: flashplayer_26_sa.exe flash_crasher.swf
(784.f3c): Break instruction exception - code 80000003 (first chance)
eax=00000000 ebx=00000000 ecx=001ef418 edx=777f6c74 esi=fffffffe edi=00000000
eip=778505d9 esp=001ef434 ebp=001ef460 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!LdrpDoDebuggerBreak+0x2c:
778505d9 cc              int     3
0:000> g
(784.f3c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file not found.  Defaulted to export symbols for FlashPlayer.exe -
eax=00f6c3d0 ebx=00000000 ecx=00000000 edx=0372b17d esi=00000000 edi=02d1b020
eip=0187b6c9 esp=001eb490 ebp=00f6c3d0 iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
FlashPlayer!IAEModule_IAEKernel_UnloadModule+0x25a559:
0187b6c9 8b11            mov     edx,dword ptr [ecx]  ds:0023:00000000=????????
0:000>
```

[12]The author has generously donated a collection of useless bugs. `unzip pocorgtfo16.pdf useless_crashers.zip` and then extract that archive with a password of "`pocorgtfo`".

Test for Library Inclusion Ok, what else can we do? Another instance of use for useless crashes that I've found is in identifying if certain library is embedded in some binary you don't have source or symbols for. Say an application renders TIFF images, and you suspect it might be using libtiff and be in OSS license violation as it's license file never mentions it. Try to open a useless libtiff crash in it, if it crashes chances are it does indeed use libtiff. A more interesting example might be some piece of code for PDF rendering. There are many many closed and open source PDF SDKs out there, what are the chances that the binary you are looking at employs it's own custom PDF parser as opposed to Poppler, MuPDF, PDFium or Foxit SDKs?

Leadtools, for example, is an imaging SDK that supports indexing PDF documents. Let's test it:

```
$./testing/LEADTOOLS19/Bin/Lib/x64/lfc \
    ./foxit_crasher/ ./junk/ -m a
Error -9 getting file information from
./foxit_crasher/8c...d174b1f189.pdf
$
```

" * * * It was a Dreaded Ordeal for Us to Meet the Lordly Hardware Clerk * * * to Purchase Five Cents Worth of Copper Burrs.

[13]Version 2017-08-23 23-34-32 shown here.

The test crash for Foxit doesn't seem to crash it, instead it just spits out an error. Let's try another one:

```
$ ./testing/LEADTOOLS19/Bin/Lib/x64/lfc \
    ./mupdf_crasher/ ./junk/ -m a
lfc: draw-path.c:520: fz_add_line_join:
Assert "Invalid line join"==0 failed.
Aborted (core dumped)
$
```

Would you look at that; it's an assertion failure so we get a bit of code path, too! Doing a simple lookup confirms that this code indeed comes from MuPDF which Leadtools embeds.

As another example, there is a tool called PSPDFKit[13] which is more complete PDF manipulation SDK (as opposed to PDFKit) for macOS and iOS. Do they rely on PDFKit at all or on something completely different? Let's try with their demo application.

```
(lldb) target create "PSPDFCatalog"
Current executable set to 'PSPDFCatalog'.
(lldb) r pdfkit_crasher.pdf
Process 53349 launched: 'PSPDFCatalog'
Process 53349 exited with status = 0
(lldb)
```

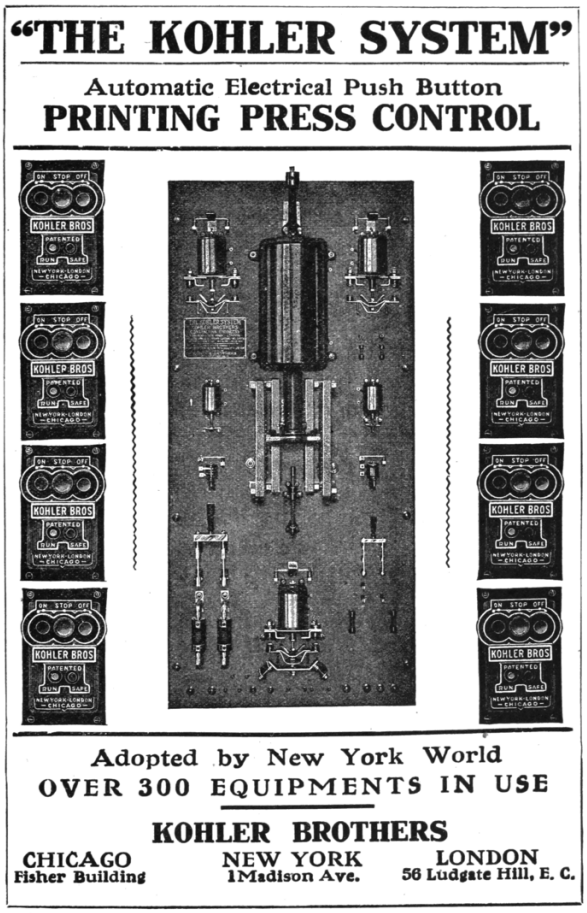
Nothing out of the ordinary, so let's try another test.

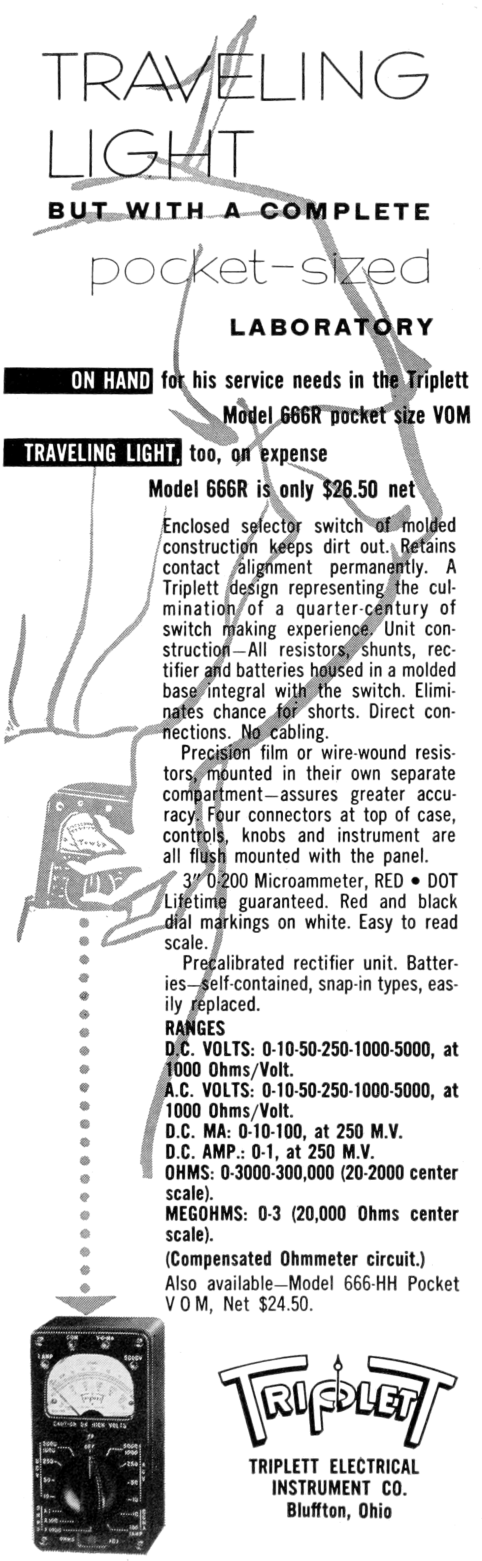
```
(lldb) r pdfium_crasher.pdf
Process 53740 launched: 'PSPDFCatalog-macOS'
Process 53740 stopped
* thread #2: tid = 0x2060fc, ...
stop reason = EXC_BAD_ACCESS
(code=2, address=0x700009a76fc8)
libsystem_malloc.dylib`
    szone_malloc_should_clear:
->0x7fff9737946d+395: callq 0x7fff9737a770
                ; tiny_malloc_from_free_list
  0x7fff97379472 <+400>: movq   %rax, %r9
  0x7fff97379475 <+403>: testq  %r9, %r9
  0x7fff97379478 <+406>: movq   %r12, %rbx
```

Now ain't that neat! It seems like PSPDFKit actually uses PDFium under the hood. Now we can proceed to dig into the code a bit and actually confirm this (in this case their license also confirms this conclusion).

What else could we possibly use crashes like these for? These could also be useful to construct a sort of oracle when we are completely blind as to what piece of code is actually running on the other side. And indeed, some folks have used this before when attacking different online services, not unlike Chris Evans' excellent writeup.[14] What would happen if you try to preview above mentioned PDFs in Google Docs, Dropbox, Owncloud, or any other shiny web application? Could you tell what those are running? Well that could be useful, couldn't it? I wouldn't call these tests conclusive, but it's a good start.

I'll finish this off with a simple observation. No one seems to care about crashes due to infinite recursion and those tend to live longest, followed of course by null pointer dereferences, so one of either of those is sure to serve you for quite some time. At least that has been the case in my very humble experience.





[14] Black Box Discovery of Memory, Scary Beast Security blog, March 2017.