Fuzzing: Revisiting Software Security

NAFIEZ @ ROOTCON 15, 2021

About Me

An independent security practitioner located in Malaysia with passion in vulnerability research, fuzzing, reverse engineering and exploit development. I play RC Drift.

Some of my notable works can be found here

- https://zeifan.my

Overview of Software Security

Why It Still Failed?

- Lack of Secure Development Lifecycle
- Ignorance from vendor by trying to avoid fixes
- Security is expensive
- Third party software developer do not follow mitigations built by Microsoft
- Security ain’t priority

What Are We Seeing Here?

- Microsoft has improved the security in their operating systems by killing and eliminating bug classes
- Exploit mitigations on different aspects, vulnerability become useless
- Finding vulnerability is HARD
- Competitor between researchers, vendors and boutique firm
- Exploit development costly

Hunting For Vulnerability

General (1/2)

- Started from lowest hanging fruit to the complex part. My previous work on hunting vulnerability in Antivirus covering various security issue and methods.
- Methods are similar, depending on the target
- Study other researchers write ups and analyzed from scratch to understand how it works. It helps to identify bugs and ideas on how to exploiting it.
- Reverse engineer patches and updates

General (2/2)

- Easiest way to hunt for vulnerability is the access to source code.
- However it is impossible to have access to source code when it comes to closed source program.

  Heavily involved in reverse engineering.
- Reverse engineering is HARD!
- One way to approach is to fuzz. Fuzzing is fun but hard too and may cause disappointment :D

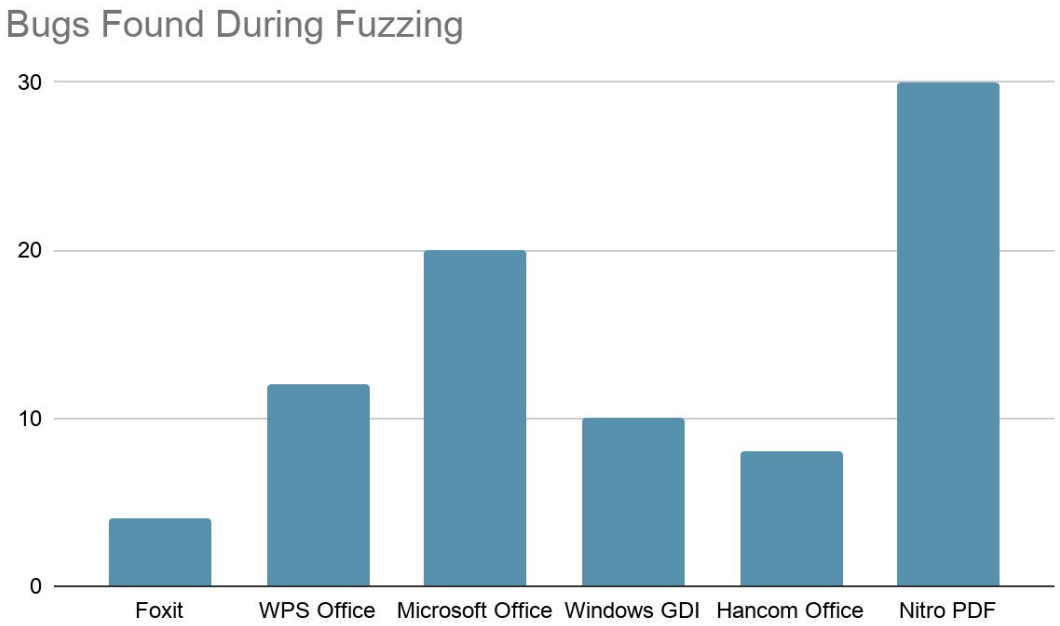
Fuzzing Approach (1/2)

- Rely on traditional method such file format fuzzing
  - Byte and Bit mutation FTW!
- Perform variant hunting
  - APIs, Functions, etc.
- Dumb & Smart Fuzzers
  - Custom & Public

Fuzzing Approach (2/2)

- Number of CVEs assigned
- Custom fuzzer built to work on specific cases such file format
    - Limited to the target itself
- Public fuzzer, we used any available fuzzers such as WinAFL and CERT BFF
- WinAFL supports coverage guided, APIs
- CERT BFF only file format, support custom Python plugin

Fuzzing Stats / Results



Public Fuzzer (1/2)

- WinAFL and CERT Basic Fuzzing Framework (BFF) are the main options on publicly available fuzzer
- WinAFL is powerful and smart fuzzer
  - Fast (depends on you harness) and it supports instrumentation too
- CERT BFF using traditional methods without coverage guided or instrumentation
  - It supports Python plugin and you can write your own fuzzer
  - The longer it runs, the slower it become :D
- Found numbers of vulnerability and assigned with CVEs for public record on vulnerability reported
- Next page shows the numbers of issue found
  - Not everything included due to some don’t have public advisory from vendors
  - I reported numbers of issue however only couple of it has CVEs assigned

Public Fuzzer (2/2)

MSRC Case 58680 - Windows GDI

MSRC Case 58593 - Windows GDI

MSRC Case 58745 - Windows GDI

MSRC Case 58843 - Windows GDI

CVE-2020-10222 - Nitro PDF Software

CVE-2020-10223 - Nitro PDF Software

CVE-2020-25290 - Nitro PDF Software

CVE-2019-19817 - Nitro PDF Software

CVE-2019-19818 - Nitro PDF Software

CVE-2019-19819 - Nitro PDF Software

* there are more…

Custom Fuzzer (1/5)

- File format fuzzing still effective these days, although it slow but we do found numbers of vulnerabilities
- Main idea is to find bug as much it can
- Heavily focus on C / C++ types of application
- Capable to fuzz complex software
- Able to catch bugs and minimize results
- Purely written in Python

Custom Fuzzer (2/5)

- Mutation on input file
  - e.g. file.exe input.test
- Covering bit flip
  - Randomize range values
  - Strings, special characters
- Detecting crashes via debugger, slow but it works :)
  - cdb, PyKD or WinAppDBG
  - Page Heap enabled

Custom Fuzzer (3/5)

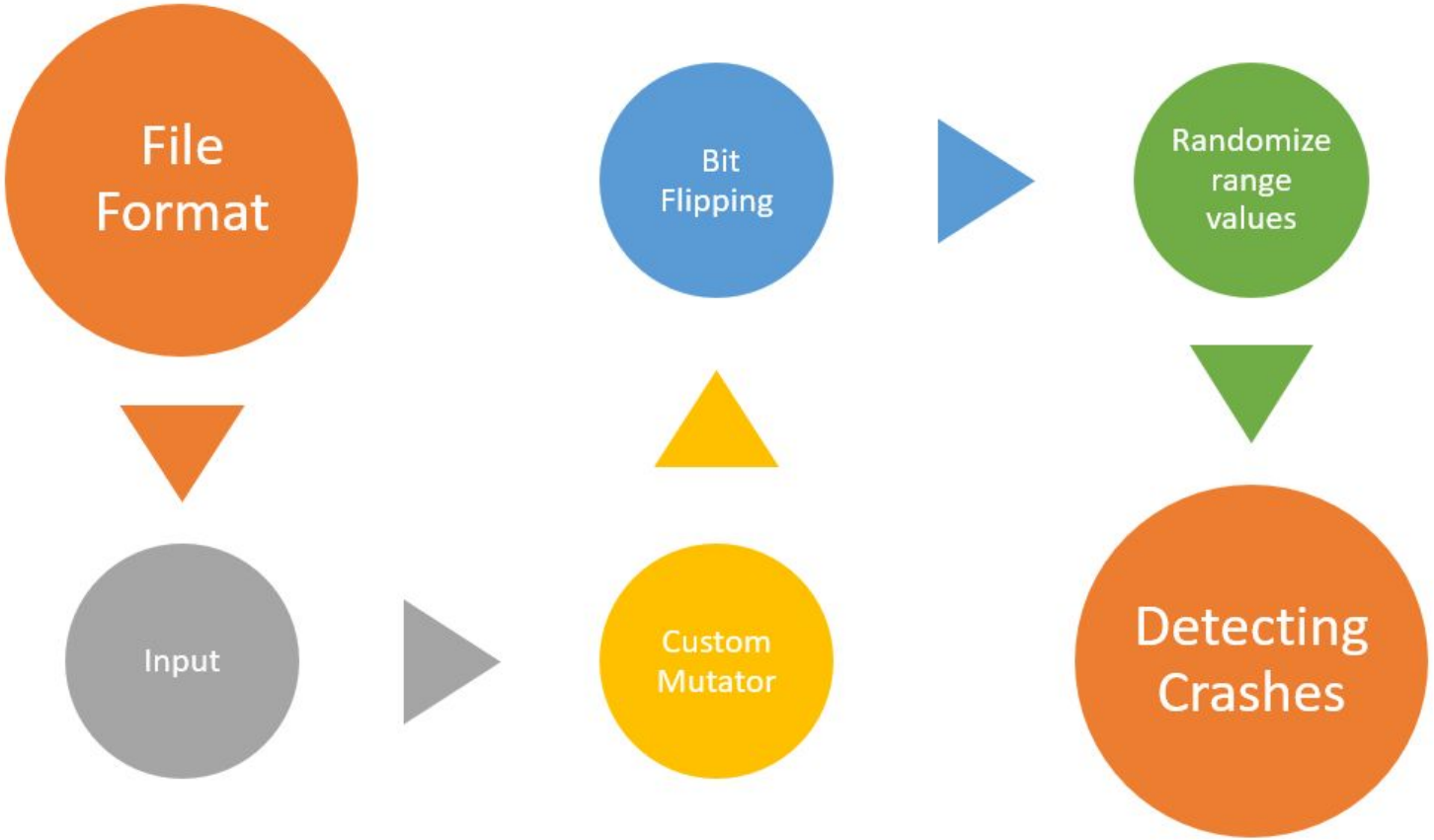
- Integers
  - Signed and Unsigned byte
  - Signed and Unsigned word
  - Signed and Unsigned dword
  - Signed and Unsigned qword
  - Negative numbers (ranging from 0x80000000 to 0xffffffff)
  - Positive numbers (ranging from 0x10000000 to 0x7fffffff)

Custom Fuzzer (4/5)

- Strings and ASCII
  - Large string s
  - Empty strings
  - Length tags modifications
  - NULL terminator
  - Append and prepend on tagged strings

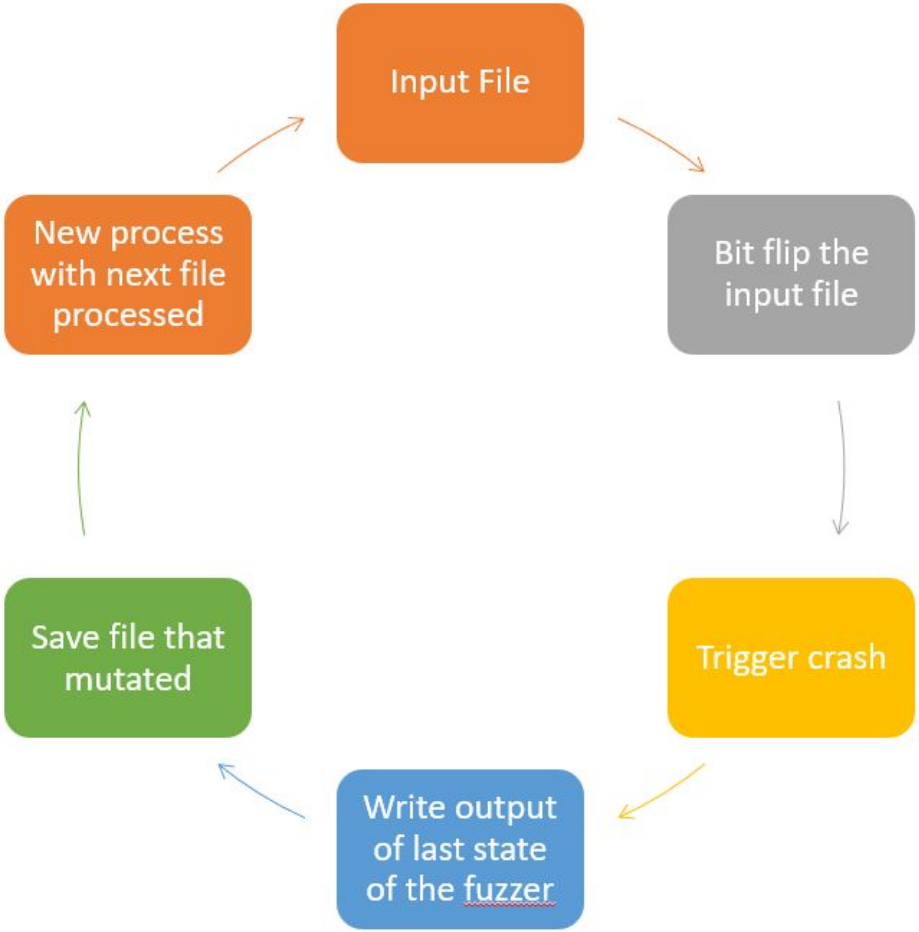
Custom Fuzzer (5/5)

- Detecting crashes could trigger false alarm
- Split out the result by performing a better filtering
  - Check last exceptions e.g. address NULL or has something on memory / register
- Important info
  - Access violation
    - Last crash disassembly code, Register value, Stack trace (sometimes inaccurate)
  - Manual verification, debugging FTW


File Format
Bit Flipping
Randomize range values
Input
Custom Mutator
Detecting Crashes

What’s Inside the Custom Fuzzer?

- No special code or techniques, just a “copy cat” code from the Internet with major modification
- Bit flip FTW :)
- No taint or guided features, fully file format fuzzer
- Initial idea is to build a framework, but looks hard LOL
- It caught real vulnerability on complex software such Microsoft Office
- Too slow but satisfied with its results XD

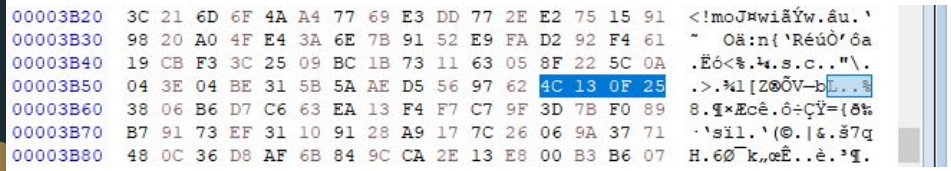

Input File
Bit flip the input file
Trigger crash
Write output of last state of the fuzzer
Save file that mutated
New process with next file processed

Test Case - Fuzzing Example

- Targeting Hancom Word processing application. Vulnerability reported to KISA.
- Corpus size around 25 KB
- Bit Flip mutation
    - Covering random range of values starting from 0x0 until 0x7FFFFFFF
- Bug found after 4 hours running, there are three different vulnerability found
- Example of fuzzing test case in next page

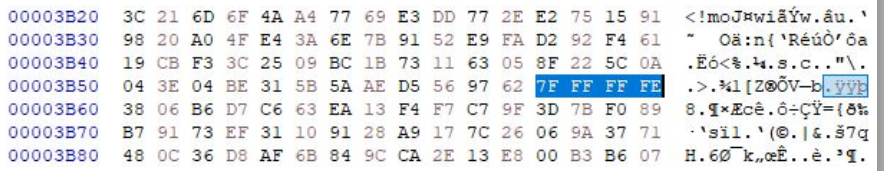
File Format Fuzzed

Original File



```
00003B20  3C 21 6D 6F 4A A4 77 69 E3 DD 77 2E E2 75 15 91  <!moJ¤wiãÝw.âu.'
00003B30  98 20 A0 4F E4 3A 6E 7B 91 52 E9 FA D2 92 F4 61  ˜  Oä:n{'RéúÒ'ôa
00003B40  19 CB F3 3C 25 09 BC 1B 73 11 63 05 8F 22 5C 0A  .Ëó<%.¼.s.c.."\.
00003B50  04 3E 04 BE 31 5B 5A AE D5 56 97 62 4C 13 0F 25  .>.¾1[Z®ÕV—bL..%
00003B60  38 06 B6 D7 C6 63 EA 13 F4 F7 C7 9F 3D 7B F0 89  8.¶×Æcê.ô÷ÇŸ={ð‰
00003B70  B7 91 73 EF 31 10 91 28 A9 17 7C 26 06 9A 37 71  ·'sï1.'(©.|&.š7q
00003B80  48 0C 36 D8 AF 6B 84 9C CA 2E 13 E8 00 B3 B6 07  H.6Ø¯k„œÊ..è.³¶.
```

Mutated File



```
00003B20  3C 21 6D 6F 4A A4 77 69 E3 DD 77 2E E2 75 15 91  <!moJ¤wiãÝw.âu.'
00003B30  98 20 A0 4F E4 3A 6E 7B 91 52 E9 FA D2 92 F4 61  ˜  Oä:n{'RéúÒ'ôa
00003B40  19 CB F3 3C 25 09 BC 1B 73 11 63 05 8F 22 5C 0A  .Ëó<%.¼.s.c.."\.
00003B50  04 3E 04 BE 31 5B 5A AE D5 56 97 62 7F FF FF FE  .>.¾1[Z®ÕV—b.ÿÿþ
00003B60  38 06 B6 D7 C6 63 EA 13 F4 F7 C7 9F 3D 7B F0 89  8.¶×Æcê.ô÷ÇŸ={ð‰
00003B70  B7 91 73 EF 31 10 91 28 A9 17 7C 26 06 9A 37 71  ·'sï1.'(©.|&.š7q
00003B80  48 0C 36 D8 AF 6B 84 9C CA 2E 13 E8 00 B3 B6 07  H.6Ø¯k„œÊ..è.³¶.
```

When the Word started to process and parse for the contents of the DOC file, crash will trigger due to malformed contents on the file formatting.

Example Minimize Results

```
Null: False

Access violation (first chance) at hwordapp!0x18e2e

Registers:
eax=baadf041 ebx=baadf041 ecx=00000000 edx=00000005 esi=014fcda8 edi=014fccd0
eip=69248e2e esp=014f9bb8 ebp=014f9bc4 iopl=0         no up ei pl nz na po cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00210203

Stack trace:
Frame    Origin
014F9BC4 BAADF04169D20B9D (0xbaadf04169d20b9d)
```

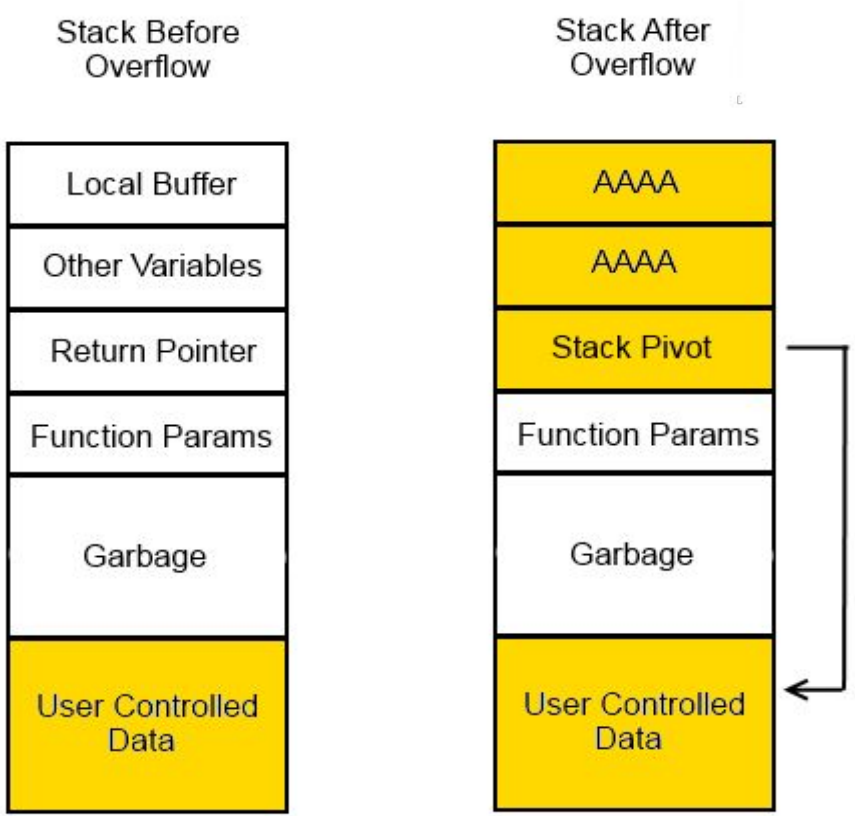
State the Art & Understanding the Attack Surface

Before that...

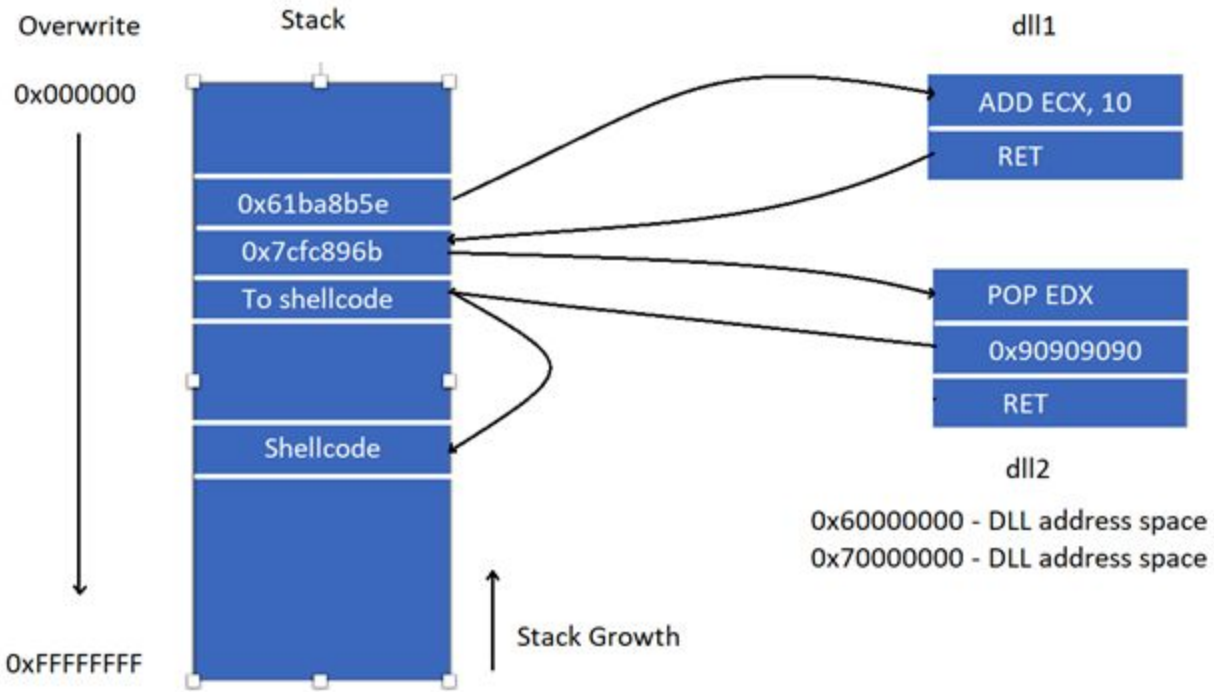
- We need to understand how the mitigations takes place
- Are the targets are really protected with the current mitigations?
- How far can we demonstrate the impact of the bug? Exploitable? Partially exploitable? Non-exploitable?
- Understand your target is very important

State the Art

- Finding memory corruption used to be easy with exploitable results
- Historically exploited for decades (we can observe in the wild exploits)
- Attack surface is always there it’s just the matter of the understanding how it works
- Heavily involved reverse engineering process

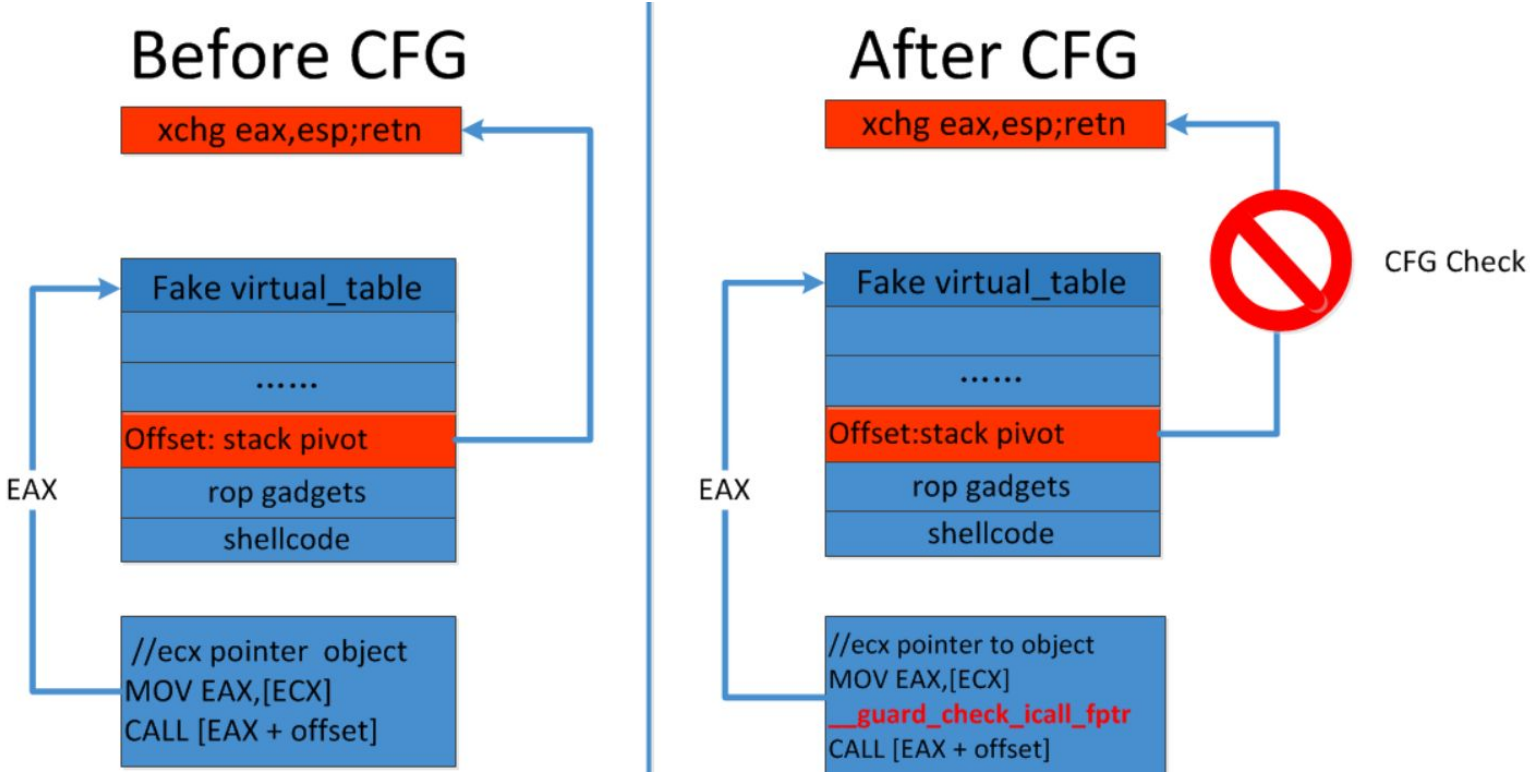


https://ctf-wiki.github.io/ctf-wiki/pwn/windows/stackoverflow/figure/demo2-1.png



ROP in action - Code Reuse Attack

https://jnet.i.lithium.com/t5/image/serverpage/image-id/18394i1F93FD8C873152B4/image-size/large?v=1.0&px=999



https://www.blackhat.com/docs/asia-17/materials/asia-17-Li-Cross-The-Wall-Bypass-All-Modern-Mitigations-Of-Microsoft-Edge.pdf

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	DEP	ASLR	Control Flow Guard	Stack Protection
MSACCESS.EXE		46,792 K	89,404 K	23128	Microsoft Access	Microsoft Corporation	Enabled (permanent)	ASLR		Disabled

Name	Description	Company Name	Path	ASLR	Control Flow Gu...
MSAIN.DLL	Microsoft Access International...	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\1033\MSAIN.DLL	ASLR	
GFX.DLL	Microsoft Office Graphics	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\GFX.DLL	ASLR	
IEAWSDC.DLL	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\IEAWSDC.DLL	ASLR	
IVY.DLL	Microsoft Ivy	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\IVY.DLL	ASLR	
MSACCESS.EXE	Microsoft Access	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\MSACCESS.EXE	ASLR	
MSOARIA.DLL	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\MSOARIA.DLL	ASLR	
msvcp140.dll	Microsoft® C Runtime Library	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\msvcp140.dll	ASLR	CFG
OART.DLL	Microsoft OfficeArt	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\OART.DLL	ASLR	
vcruntime140.dll	Microsoft® C Runtime Library	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\Office16\vcruntime140.dll	ASLR	CFG
MSOINTL.DLL	Office International Resources	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\1033\...	ASLR	
msointl30.dll	Office International Resources	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\1033\...	ASLR	
OFFICE.ODF	Microsoft Office culture data dll	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\Culture...	ASLR	
MSO.DLL	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\MSO....	ASLR	
Mso20win32client.dll	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\Mso20...	ASLR	
Mso30win32client.dll	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\Mso30...	ASLR	
MSO40UIRES.DLL	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\MSO4...	ASLR	
Mso40UIwin32client....	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\Mso40...	ASLR	
Mso50win32client.dll	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\Mso50...	ASLR	
Mso98win32client.dll	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\Mso98...	ASLR	
MSO99LRES.DLL	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\MSO9...	ASLR	
MSORES.DLL	Microsoft Office component	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\MSOR...	ASLR	
RICHED20.DLL	RichEdit Version 8.0	Microsoft Corporation	C:\Program Files (x86)\Microsoft Office\root\VFS\ProgramFilesCommonX86\Microsoft Shared\OFFICE16\RICHE...	ASLR	

```
.text:10032750 __alloca_probe  proc near               ; CODE XREF: pfnWinEventProc+B↑p
.text:10032750                                         ; sub_1001B900+B↑p ...
.text:10032750                 push    ecx
.text:10032751                 lea     ecx, [esp+4]
.text:10032755                 sub     ecx, eax
.text:10032757                 sbb     eax, eax
.text:10032759                 not     eax
.text:1003275B                 and     ecx, eax
.text:1003275D                 mov     eax, esp
.text:1003275F                 and     eax, 0FFFFF000h
.text:10032764
.text:10032764 cs10:                                   ; CODE XREF: __alloca_probe+29↓j
.text:10032764                 cmp     ecx, eax
.text:10032766                 jb      short cs20
.text:10032768                 mov     eax, ecx
.text:1003276A                 pop     ecx
.text:1003276B                 xchg    eax, esp
.text:1003276C                 mov     eax, [eax]
.text:1003276E                 mov     [esp+0], eax
.text:10032771                 retn
.text:10032772 ; ---------------------------------------------------------------------------
.text:10032772
.text:10032772 cs20:                                   ; CODE XREF: __alloca_probe+16↑j
.text:10032772                 sub     eax, 1000h
.text:10032777                 test    [eax], eax
.text:10032779                 jmp     short cs10
```

https://twitter.com/zeifan/status/1298074650098819072/photo/1

After...

- Finding bugs ain’t easy task nowadays
- Modern exploitation is hard and expensive
- One has to chain multiple bugs to achieve powerful exploitation
- Exploit development costs continue growing

Attack Surface

- Bugs are exist, the entry point is important to hunt
- Patches and fixes let us understand what have been fix previously and could introduce another bug
- Variant hunting indeed important however it’s pretty hard without proper guided fuzzing
- Input, processing and parsing are the common attack surface

Real World Vulnerabilities

- Found numbers of bug on various software
- In this talk, I'll present case study on Microsoft Access and Hancom Word Processor
- Microsoft did a great job on fixing and future plan release to eliminate the bugs that reported
- Fun fact about Hancom, I reported vulnerability to KISA however no further updates / news from them on the reported bug

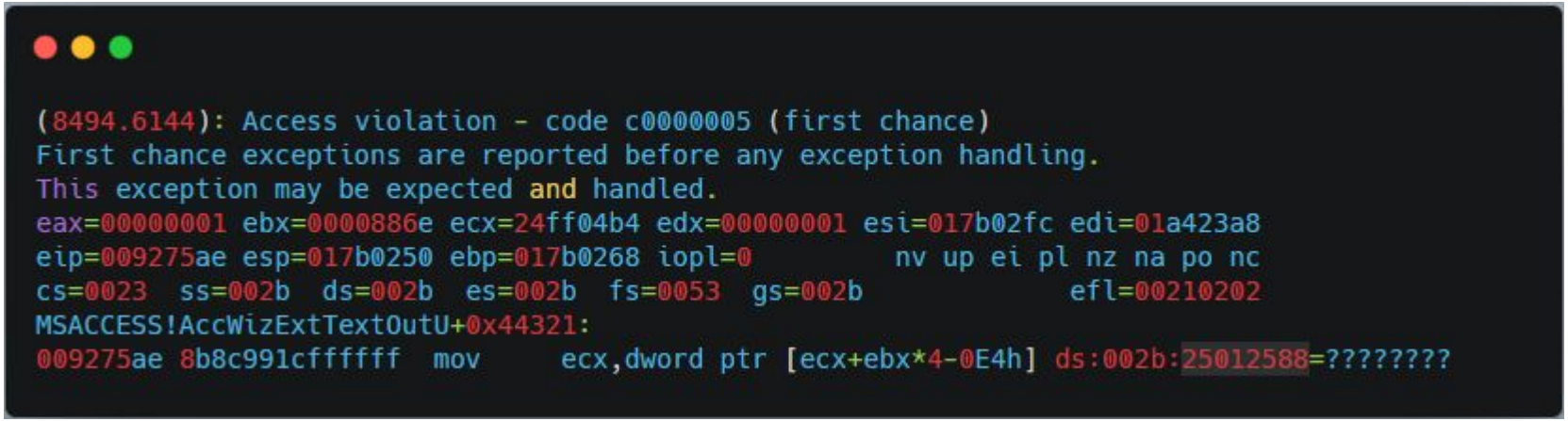
Case Study #1 : (CVE-2020-16957) Microsoft Access Connectivity Engine Remote Code Execution Vulnerability

Summary

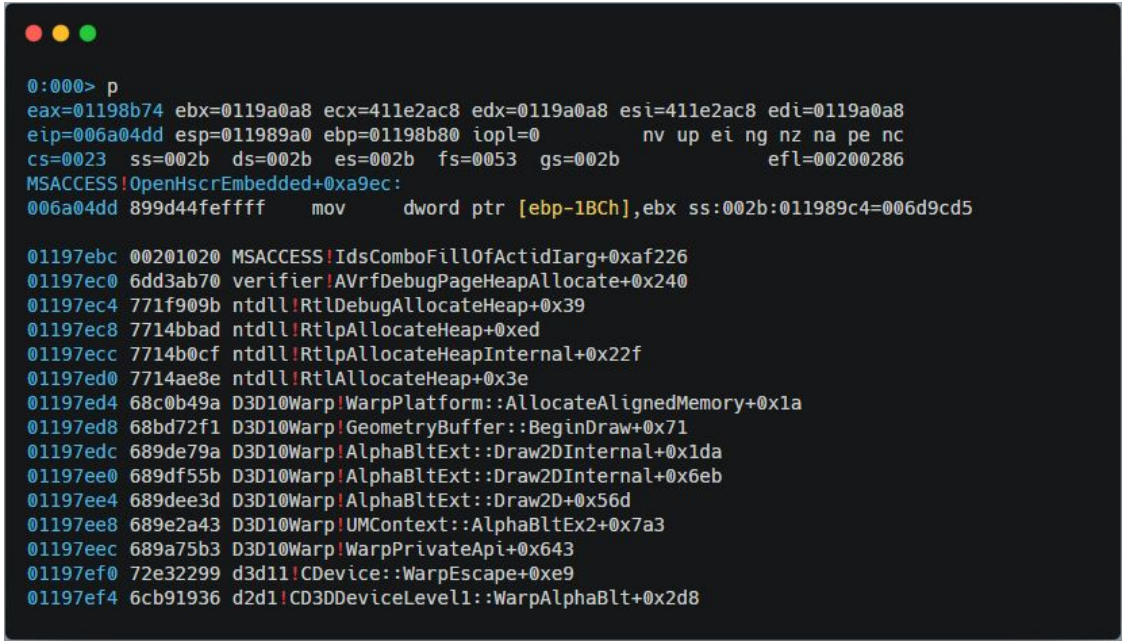
The bug was found with my custom fuzzer and Microsoft acknowledge me on their portal along with the CVE-2020-16957. The idea fuzzing Microsoft Access is by feed the fuzzer with 10MB+ file size

A heap corruption was detected when handling a specially crafted Access database and the bug reproducible on Windows 10 x64 version 1909. Affected version of Microsoft Access 2016 with version 16.0.13029.20308.

Analysis (1/2)


(8494.6144): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=0000886e ecx=24ff04b4 edx=00000001 esi=017b02fc edi=01a423a8
eip=009275ae esp=017b0250 ebp=017b0268 iopl=0 nv up ei pl nz na po nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00210202
MSACCESS!AccWizExtTextOutU+0x44321:
009275ae 8b8c991cffffff mov ecx,dword ptr [ecx+ebx*4-0E4h] ds:002b:25012588=????????

Analysis (2/2)


0:000> p
eax=01198b74 ebx=0119a0a8 ecx=411e2ac8 edx=0119a0a8 esi=411e2ac8 edi=0119a0a8
eip=006a04dd esp=011989a0 ebp=01198b80 iopl=0 nv up ei ng nz na pe nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00200286
MSACCESS!OpenHscrEmbedded+0xa9ec:
006a04dd 899d44feffff mov dword ptr [ebp-1BCh],ebx ss:002b:011989c4=006d9cd5
01197ebc 00201020 MSACCESS!IdsComboFillOfActidIarg+0xaf226
01197ec0 6dd3ab70 verifier!AVrfDebugPageHeapAllocate+0x240
01197ec4 771f909b ntdll!RtlDebugAllocateHeap+0x39
01197ec8 7714bbad ntdll!RtlpAllocateHeap+0xed
01197ecc 7714b0cf ntdll!RtlpAllocateHeapInternal+0x22f
01197ed0 7714ae8e ntdll!RtlAllocateHeap+0x3e
01197ed4 68c0b49a D3D10Warp!WarpPlatform::AllocateAlignedMemory+0x1a
01197ed8 68bd72f1 D3D10Warp!GeometryBuffer::BeginDraw+0x71
01197edc 689de79a D3D10Warp!AlphaBltExt::Draw2DInternal+0x1da
01197ee0 689df55b D3D10Warp!AlphaBltExt::Draw2DInternal+0x6eb
01197ee4 689dee3d D3D10Warp!AlphaBltExt::Draw2D+0x56d
01197ee8 689e2a43 D3D10Warp!UMContext::AlphaBltEx2+0x7a3
01197eec 689a75b3 D3D10Warp!WarpPrivateApi+0x643
01197ef0 72e32299 d3d11!CDevice::WarpEscape+0xe9
01197ef4 6cb91936 d2d1!CD3DDeviceLevel1::WarpAlphaBlt+0x2d8

Case Study #2 - (MSRC Case 60509) Microsoft Access 2016 Heap-Based Out-of-Bounds Read

Summary

An Out-of-Bounds Read vulnerability has been detected when handling a specially crafted Access database. The following crash was observed in Microsoft Access 2016 with Windbg. The vulnerability was found during fuzzing activity.

Microsoft consider this bug as moderate info disclosure meaning no fix and it will only included in the next product cycle (not the monthly patch)

Analysis (1/2)

```
(9b8.2fec): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00c9d840 ecx=00000001 edx=00000001 esi=1ece34d4 edi=0174ae0f
eip=65dd2cfd esp=0174ad44 ebp=0174ad64 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00210202
VCRUNTIME140!memcpy+0x51d:
65dd2cfd 8a06            mov     al,byte ptr [esi]         ds:002b:1ece34d4=??
```

Analysis (2/2)

```
STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
0174ad64 008786ca 0174e440 0174e440 0174ae14 VCRUNTIME140!memcpy+0x51d
0174ad7c 008840b8 0174e440 00c9d840 9520ad47 MSACCESS!MSAU_OfficeGetTcDIB+0x24e2c
0174b660 008849d2 9520a41b 16392240 0174bfcc MSACCESS!MSAU_OfficeGetTcDIB+0x3081a
0174bf3c 008849d2 9520d33f 0174ce1c 1ecd04b0 MSACCESS!MSAU_OfficeGetTcDIB+0x31134
0174c818 00884d0b 9520d67b 164070a0 00008000 MSACCESS!MSAU_OfficeGetTcDIB+0x31134
0174cd5c 00885298 11be5838 11be5838 0174f4f8 MSACCESS!MSAU_OfficeGetTcDIB+0x3146d
0174e3d8 00883378 11be5838 16347730 0174f4f8 MSACCESS!MSAU_OfficeGetTcDIB+0x319fa
0174e730 0070f678 11be5838 11be5838 0174f4f8 MSACCESS!MSAU_OfficeGetTcDIB+0x2fada
0174e864 0070f7b0 11be5838 0174f4f8 00008000 MSACCESS!AccWizExtTextOutU+0x8c3eb
0174e89c 0070df18 0174f4f8 00000000 00000001 MSACCESS!AccWizExtTextOutU+0x8c523
0174f610 0070f87c 11be5838 0174fdd8 00000000 MSACCESS!AccWizExtTextOutU+0x8ac8b
0174f678 00832d0d 11be5838 0174fdd8 00000202 MSACCESS!AccWizExtTextOutU+0x8c5ef
0174fda0 00626558 00000202 0174fdd8 00000080 MSACCESS!MSAU_GetSizeList+0x252be
0174fef8 00625a63 00000000 0c272430 00000000 MSACCESS!MSAU_ErrSortStringArray+0x114c0
01750208 00630861 952123df 00000000 00000b86 MSACCESS!MSAU_ErrSortStringArray+0x109cb
017538f8 0062d6d9 01753934 00000001 00000100 MSACCESS!MSAU_ErrSortStringArray+0x1b7c9
01755120 0062d4b2 00124204 00000000 00000004 MSACCESS!MSAU_ErrSortStringArray+0x18641
01755354 00b6ea06 9521484b 01759804 00bb4da3 MSACCESS!MSAU_ErrSortStringArray+0x1841a
0175536c 628860f1 122bf430 ab72d767 0175b938 MSACCESS!OpenHscrEmbedded+0x118f15
...cut here...
0175ff04 00000000 00458f60 01516000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

Case Study #3 - (MSRC Case) Microsoft Access 2016 Out-of-Bounds Read Vulnerability

Summary

An Out-of-Bounds Read vulnerability has been detected when handling a specially crafted Access database. The following crash was observed in Microsoft Access 2016 with Windbg. The vulnerability was found during fuzzing activity.

Microsoft does not consider this OOB Read as exploitable.

Analysis (1/2)

```
(32f4.2de4): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=1b0d2001 ebx=00000280 ecx=00000001 edx=00000280 esi=1b0d2000 edi=155bbc00
eip=6df2317e esp=018f0670 ebp=018f06b8 iopl=0         nv up ei pl nz na pe cy
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00210207
VCRUNTIME140!memcpy+0x4e:
6df2317e f3a4            rep movs byte ptr es:[edi],byte ptr [esi]
```

Analysis (2/2)

```
STACK_TEXT:
018f0674 62cf82d0 155bb981 1b0d1d81 00000280 VCRUNTIME140!memcpy+0x4e
018f06b8 62cda1ad 1b0d1d81 00000280 018f0778 ACECORE+0x382d0
018f06e4 62ce3cad 00000e21 154fe990 018f0778 ACECORE+0x1a1ad
018f0714 62ce39b5 018f0778 23fb4f0f 018f0778 ACECORE+0x23cad
018f0754 62ce3731 00000000 00000002 018f0778 ACECORE+0x239b5
018f07a0 62ce06e2 13634220 153f0c90 00000001 ACECORE+0x23731
018f07c0 62cf0ae6 13634220 000007ff 00000001 ACECORE+0x206e2
018f09ec 62cf8703 04000000 018f0aec 00000004 ACECORE+0x30ae6
018f0ae4 62cf845e 10000000 15506c38 00000000 ACECORE+0x38703
018f0b18 62cf8094 13634220 000000fe 10000000 ACECORE+0x3845e
018f0bf0 62cf502f 62e312dc 62e312f0 00000000 ACECORE+0x38094
018f1080 62cf4952 13634220 1538b348 0a0e9d90 ACECORE+0x3502f
018f10a4 62cf8010 13634220 1538b348 0a0e9d90 ACECORE+0x34952
018f10c4 005640e5 13634220 1538b348 0a0e9d90 ACECORE+0x38010
018f12c8 00ba2a9e 00000000 130cded0 130cdea0 MSACCESS!CreateIExprSrvObj+0x1674
018f12e4 005622a5 130cdea0 018f2108 018f1494 MSACCESS!OpenHscrEmbedded+0x4f6e8
018f1400 0063c738 130cdea0 018f2108 018f1494 MSACCESS!AccessLoadString+0x624e
```

Case Study #4 - (MSRC Case 62010) Microsoft Access 2016 Heap Corruption

Summary

An heap corruption (invalid pointer) has been detected when handling a specially crafted Access database. The following crash was observed in Microsoft Access 2016 and 2019 with Windbg. The vulnerability was found during fuzzing activity.

No fix for this issue as Microsoft stated user are required to run VBScript.

Analysis (1/2)

```
(e80.558): 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 could not be found.  Defaulted to export symbols for C:\Program Files
(x86)\Common Files\Microsoft Shared\VBA\VBA7.1\VBE7.DLL -
eax=004f06d0 ebx=0000d040 ecx=17747000 edx=00000000 esi=17747000 edi=0c0c693e
eip=5f6337f4 esp=004f0664 ebp=004f06a0 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00210202
VBE7!DllUnregisterServer+0x348fb:
5f6337f4 8a01            mov     al,byte ptr [ecx]          ds:002b:17747000=??
```

Analysis (2/2)

```
bp 5f6337f4 ".if (poi(esi) == 0x17747000) {} .else {gc}"

0:000> bp 5f6337f4 ".if (poi(esi) == 0x17747000) {} .else {gc}"
0:000> g
eax=004f93d0 ebx=0000d06f ecx=1774effe edx=00000000 esi=1774effe edi=0c0c2d83
eip=5f6337f4 esp=004f92a4 ebp=004f92e0 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00200202
VBE7!DllUnregisterServer+0x348fb:
5f6337f4 8a01            mov     al,byte ptr [ecx]          ds:002b:1774effe=00

1774effe 00 00 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..??????????????
1774f010 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
1774f022 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
1774f034 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ????????????????
```

Case Study #5 - Hancom Word Out-of-Bounds Read Vulnerability

Summary

An heap out-of-bounds read vulnerability exists in Hancom Word software that is caused when the Office software improperly handles objects in memory while parsing specially crafted Office files. An attacker who successfully exploited the vulnerability remotely and could run arbitrary code in the context of the current user. Failure could lead to denial-of-service. Product and version affected was Hancom Office 2020 with version 11.0.0.1. The vulnerability was found with fuzzing.

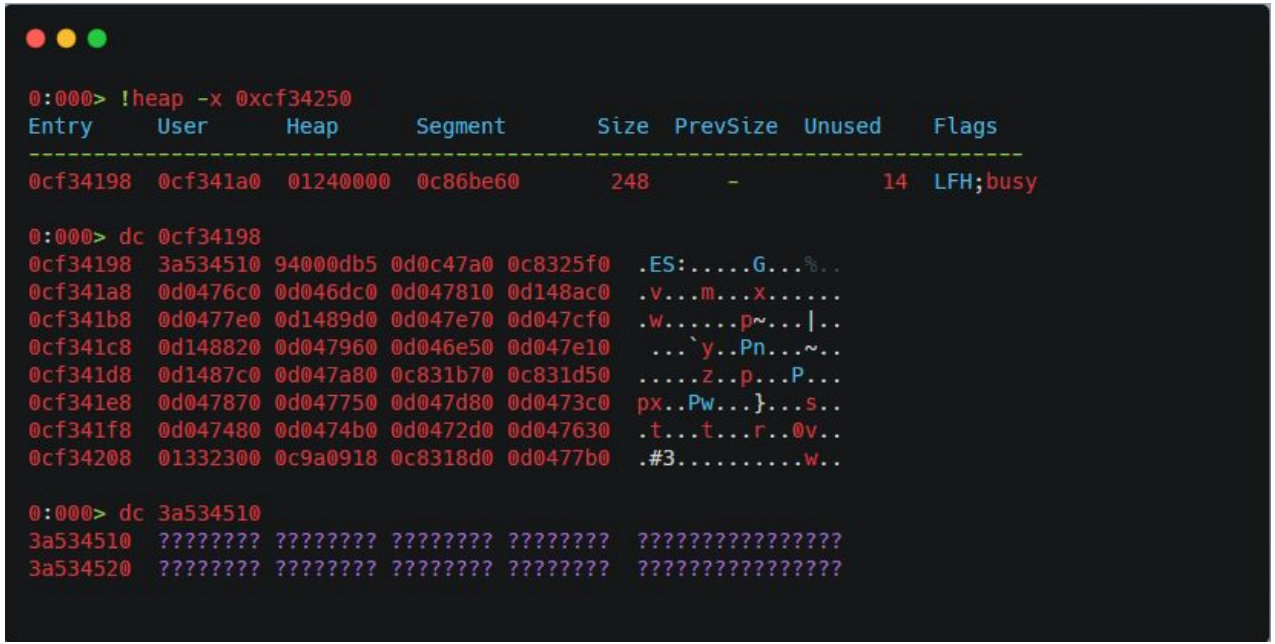
Analysis (1/3)

```
(39c.d14): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=5c002000 ebx=0d046af0 ecx=5c002000 edx=577d8b18 esi=0cf34250 edi=00000000
eip=6aa18f9a esp=00f7e20c ebp=00f7e20c iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00210206
HwordApp!HwordDeletePropertyArray+0xa5ee1a:
6aa18f9a 8b480c          mov     ecx,dword ptr [eax+0Ch] ds:002b:5c00200c=????????
```

Analysis (2/3)

```
STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
00f7e20c 6aaca178 0ceee2f8 6aacb2eb 0ce7e308 HwordApp!HwordDeletePropertyArray+0xa5ee1a
00f7e234 6a74a747 00000043 00000043 41e32dfb HwordApp!HwordDeletePropertyArray+0xb0fff8
00f7e2b0 6aa2c2f0 0cc85428 00000001 00000000 HwordApp!HwordDeletePropertyArray+0x7905c7
```

Analysis (3/3)


0:000> !heap -x 0xcf34250
Entry User Heap Segment Size PrevSize Unused Flags
0cf34198 0cf341a0 01240000 0c86be60 248 - 14 LFH;busy
0:000> dc 0cf34198
0cf34198 3a534510 94000db5 0d0c47a0 0c8325f0 .ES:.....G...%..
0cf341a8 0d0476c0 0d046dc0 0d047810 0d148ac0 .v...m...x......
0cf341b8 0d0477e0 0d1489d0 0d047e70 0d047cf0 .w.....p~...|..
0cf341c8 0d148820 0d047960 0d046e50 0d047e10 ...`y..Pn...~..
0cf341d8 0d1487c0 0d047a80 0c831b70 0c831d50 .....z..p...P...
0cf341e8 0d047870 0d047750 0d047d80 0d0473c0 px..Pw...}...s..
0cf341f8 0d047480 0d0474b0 0d0472d0 0d047630 .t...t...r..0v..
0cf34208 01332300 0c9a0918 0c8318d0 0d0477b0 .#3.........w..
0:000> dc 3a534510
3a534510 ???????? ???????? ???????? ???????? ????????????????
3a534520 ???????? ???????? ???????? ???????? ????????????????

Vulnerability Disclosure...again...

Disclosure

- Were still seeing debates on vulnerability disclosure
- Painful processes, both party researchers and vendors
- We do see most vendors have vulnerability disclosure process
- Some offered bounty and some don’t, there’s debate on this too

Do’s & Don’ts!

- Provide as much information to ease the vendors task
- If necessary, use all the mediums to inform vendors
- Get some feedback from other researchers on disclosing vulnerability
- Follow the standard vulnerability disclosure (90 days perhaps?)
- Get CERTs involved

- Avoid public disclosure without notifying vendors
- Do not talk publicly on what you found not until it gets fix

Conclusion

- Best defense is offense
- Finding bugs = needle in a haystack
- Proper disclosure with vendors for bugs fixes
- Long live file format fuzzing :)