TRACK 2

HITB SECCONF
AMSTERDAM - 2021

How I Found 16 Microsoft Office Excel Vulnerabilities in 6 Months

Quan Jin(@jq0904) of DBAPPSecurity Lieying Lab

About me

- Security Research Expert from DBAPPSecurity Lieying Lab
  - Previous: 360 ATA Team
  - BlueHat Shanghai 2019 Speaker

- Focus on vulnerability research and ITW 0day hunting
  - 25+ CVE acknowledgments from Microsoft and Adobe
  - TOP39 in 2020 MSRC Most Valuable Security Researchers
  - Captured several in-the-wild 0days on Windows platform
    - Previous: CVE-2018-0802, CVE-2018-8174, CVE-2018-5002, CVE-2018-15982
    - Recent: CVE-2021-1732

Agenda

- Introduction
- Methodology & Implement
- Equipment
- Problems
- Results
- Limitations
- Acknowledgements

Introduction

- At the beginning of 2020, I decided to learn something about fuzzing. I first read some papers about fuzzing, such as:
  - 《Finding security vulnerabilities with modern fuzzing techniques》

- After learning the basic concepts about fuzzing, I decide to do some fuzzing job on Windows platform

- My goal was to get a CVE number from Microsoft through fuzzing

Fuzzers

- Linux platform
  - ~~AFL~~
  - ~~LibFuzzer~~
  - ~~Honggfuzz~~

- Windows platform
  - ~~WinAFL~~
    - It is a great tool
    - It can not handle large & complex software such as Microsoft Office
    - I want to choose a target which is less targeted by WinAFL
  - Other tools
    - Looking for ...

Choose a target

- Adobe Reader was heavily fuzzed by WinAFL at the year of 2018
- Internet Explorer was heavily fuzzed by Domato during 2017, 2018 and 2019
- Few people have done effective Office fuzzing work, but there do have some, such as Jaanus Kaap's presentation at POC2018
  - Microsoft Office is a good target for me

Two questions

1. Is it possible to find a bug on Microsoft Office on several months for a newcomer in fuzzing?
   - I'm a newcomer in fuzzing
   - But I have extensive experience in office vulnerability analysis

2. Microsoft Office consists of multiple components, should I choose Word, PowerPoint, Excel or other component to fuzz?
   - Let's do some statistics to answer it

Statistics

	Word	PowerPoint	Excel	Outlook	Office
2017	0	3	5	8	43
2018	4	3	23	9	33
2019	8	1	12	1	2
2020(up to June)	7	0	9	0	3
Summary	19	7	**49**	18	81

Note: The column "office" represents office vulnerabilities that do not specify specific components. Which means that they may be Word, PowerPoint, Excel, Outlook or other vulnerabilities

The initial statistical time is up to April 2020, I updated the statistical data in June 2020.

Learn from statistics

- Around 2018, Microsoft made a change to the disclosure name of office vulnerabilities to make the classification more detailed
- From 2017 to 2020, the Excel component has the most vulnerabilities almost every year
- From 2017 to 2020, the PowerPoint component has the least vulnerabilities almost every year

Let's fuzz Excel

Methodology & Implement

- Seeds - How to collect seeds
- Mutator - How to mutate
- Detection - How to catch exceptions
- Triage - How to classify and de-duplicate crash files
- Reproducer - How to reproduce the crash
- Report - How to report the vulnerability to the vendor

Seeds

- Contextures (https://www.contextures.com)
- Vertex42 (https://www.vertex42.com)
- ~~Excel files provided by Jaanus Kaap (https://foxhex0ne.com)~~
  - No longer accessible now

Corpus distillation

- While trying to solve the above problem, I saw two blogs by Jaanus Kaap(no longer accessible now):
  - ~~Let's get things going with basics of file parsers fuzzing~~
  - ~~Let's continue with corpus distillation~~

- You can still read his presentation on POC2018 Conference
  - Document parsers "research" as passive income

Corpus distillation

- Basic ideas of corpus distillation on Excel files:
  1. Select a module
  2. Use tools(IDA+Python) and initial seeds to make statistics on the module coverage
     - Count all code blocks of the module
     - Replace the origin byte with 0xCC(int 3) at the start of every code block
     - Execute files and restore the origin byte, recode the block that is hit
  3. Select the smallest number of files with the highest module coverage
  4. If possible, the smaller the file size, the better

Mutation

- I transplant the following mutation algorithms in Honggfuzz
  - mangle_Bit
  - mangle_IncByte
  - mangle_DecByte
  - mangle_NegByte
  - mangle_Bytes
  - mangle_ASCIINum
  - mangle_CloneByte
  - mangle_AddSub
- I also integrate all the values of the bytes replacement part of AFL, LibFuzzer and Honggfuzz, and construct a mutation value replacement table covering these three fuzzers

Detection

- winappdbg or pydbg
- Vanapagan is a good example
  - https://github.com/JaanusKaapPublic/Vanapagan
- Turn on Page Heap for Excel.exe
  - Improve the catch rate of heap memory access exceptions

Triage

- Only need to pay attention to the exception with the exception code of 0xC0000005
  - 0xC0000005: Access violation
  - ~~0xC00000FD: Stack exhaustion, MSRC doesn't accept this type~~

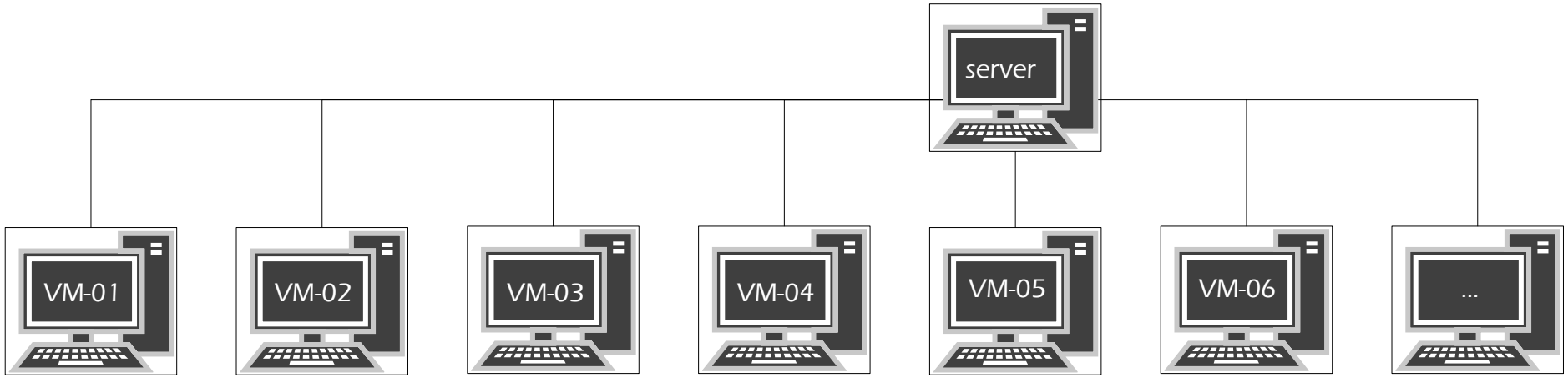
- Only need to pay attention to the non-null pointer reference case
  - Microsoft does not accept null pointer reference vulnerabilities
    - Non-null pointer reference: MSRC loves it
    - ~~Null pointer reference for MSRC: MSRC doesn't like it~~

Triage

- My classification rule for Microsoft Office Excel exceptions
  - Non-null pointer reference
    - Read access violation
      - Out-of-bound read
      - Use-after-free read
      - ...
    - Write access violation
      - Out-of-bound write
      - Use-after-free write
      - ...

Triage

- In terms of real-time synchronization of the fuzz results across multiple virtual machines, I use a FTP server which serves in a virtual machine as the result server, and install the pyftpdlib module in server and clients



Reproducer

- Not all crash files captured during fuzzing can be reproduced
- I make multiple Office environments to reproduce the crash files →
- I write a reproducer based on my fuzzer
  - For those reproduced by the reproducer, I will perform some manual check
  - If both pass, the file will be regarded as a valid vulnerability file

- Office2007
  - no patch
  - full patch
- Office2010
  - no patch
  - full patch
- Office2013
  - no patch
  - full patch
- Office2016
  - no patch
  - full patch
- Office2019
  - no patch
  - full patch

Report

- When a crash file is successfully reproduced, it can be generated a professional report with the help of BugId
  - Thanks to SkyLined(@berendjanwever) for this great tool
- BugId can only run on Windows 10 environment
- Need to make a "Windows 10+Office" with the latest Office and full patch version environment

Report

- Below is the BugId report I generate for one of my Excel vulnerabilities

BugId OOBR[8]+0 cd3.f66 @ excel.exe+0xDF6C24 summary

BugId:	OOBR[8]+0 cd3.f66
Location:	excel.exe+0xDF6C24
Description:	An Access Violation exception happened at 0x1195B000 while attempting to read memory at 0x1195B000; at the end of a 8 bytes heap block at 0x1195AFF8. This indicates an Out-Of-Bounds (OOB) access bug was triggered.
Version:	**Excel.exe**: 14.0.7249.5000 (x86)
Security impact:	Potentially exploitable security issue that might allow information disclosure and (less likely) arbitrary code execution.
Arguments:	['[redacted].xls']

BugId version 2019-10-17 11:03 by SkyLined. You may not use this version of BugId for commercial purposes. Please contact the author if you wish to use BugId commercially. Contact and licensing information can be found at the bottom of this report.

Report

- Once you have the BugId report, you can submit the vulnerability to MSRC
  - MSRC Researcher Portal
    - https://msrc.microsoft.com
  - The specific format of the vulnerability report
    - https://www.microsoft.com/en-us/msrc/bounty-example-report-submission
  - The poc file and BugId report can be uploaded as attachments

Equipment

- My entire fuzz machine is only one computer with the following configuration:
  - i7-8700 (12 Cores)
  - 16G DDR3 RAM
  - 3.2GHz Primary Frequency
  - 1T HDD
- I also have a laptop for reproduction and report generation
- These are all my fuzzing equipment

Problems

- Dialog click
- Virtual machine size
- Speed of execution
- Version switching
- Fuzz strategy
- Crash management

Dialog click

- My way of solving Excel dialog boxes are as follows
  1. Before each start of the file(or the end of the file), clean up the relevant registry item
     - HKCU\Software\Microsoft\Office\Version\Excel\Resiliency
  2. Add a simple simulation click tool during the fuzzing, such as starting a separate thread for window enumeration and dialog click. A good example is cuckoo sandbox human plugin
     - https://github.com/cuckoosandbox/cuckoo/blob/master/cuckoo/data/analyzer/windows/modules/auxiliary/human.py

Virtual machine size

- During the fuzzing process, a large number of files are generated in each virtual machine
  - These files will gradually increase the size of each virtual machine
  - Usually several to dozens of GBs per virtual machine

- This will lead the disk overhead of the host
  - The fuzzer will stop

Virtual machine size

- Ensure the fuzzer has effectively cleaned up temp files which generated by the previous file before starting the next file
  - %AppData%\Local\Temp
  - %AppData%\Roaming\Microsoft\Office\Recent
  - %AppData%\Roaming\Microsoft\Windows\Recent
- Use Dism++ tool to regularly clean up the temp files inside each virtual machine
- Configure the virtual machine to automatically clean up the disk after shutting down
  - VMware provides this option for Windows Virtual Machine

Speed of execution

- File size
- Office version
- The stability of fuzzer
- Disk IO
- CPU Cores, RAM and Primary Frequency

File size

- In the corpus distillation stage, I have selected as small a seed as possible while ensuring coverage

- From a statistical point of view, for Excel, files smaller than 400KB are more likely to produce vulnerabilities

Office version

- Conflict
  - The higher the version, the slower the opening speed
  - The higher the version, the larger the amount of code and the number of potential vulnerabilities

- I decide to focus on vulnerabilities which exists from Office 2007 to Office 2019
  - The main fuzz environment I finally choose is Office 2010
  - Speed up fuzzing by choosing a lower Office version
  - 1.5w per day each VM * 10 (not very fast)

The stability of fuzzer

- If a fuzzer is unstable and crashes itself when executing, that is sad
  - Some fuzzers that uses winappdbg may have this problem on x64 environment

- I mainly run my fuzzer on x86 environment
  - After observing and improving for a long period of time, my fuzzer has achieved relatively good stability, it can run for weeks without problems

Disk IO

- I use HDD for fuzzing
  - Due to the limitation of disk IO, sometimes the fuzzers in virtual machines will cause VMware itself to hang on for a long time
  - It is necessary to clean up the environment in the virtual machine and restart the fuzzing every once in a while

- I think SSD will improve a lot

CPU Cores, RAM and Primary Frequency

- The number of CPU cores and the RAM capacity
  - Determine the maximum number of virtual machines that can be opened at the same time
  - The bigger the two indicators, the better

- Primary Frequency
  - Affects the opening speed of the Excel process
  - The bigger the primary frequency, the better

Version switching

- Files that cannot be triggered on x86 can be triggered under x64
- Files that cannot be triggered in a lower patch environment can be triggered in a higher patch environment
- Files that cannot be triggered in the English environment can be triggered in the Chinese environment

Fuzz strategy

- What I have is a machine consisted of these
  - i7-8700 (12 Cores) +16G DDR3 RAM + 3.2GHz Primary Frequency + 1T HDD

- What I want are
  - Obtain as much vulnerabilities as possible in the shortest time
  - Find vulnerabilities that exist in all versions of Office

- This force me to do many thoughts and explorations on how to configure fuzz strategies

Fuzz strategy

- Skip the first 512 bytes of the header of the OLE2 file during mutation
- Collect xls files which were made with old versions of Excel in the 1990s and 2000s
- Select attack surface that may cause problems based on my experience
  - E.g. pivot table
- For a period of time, select the Excel files that is most likely to cause problems in the current results, and increase the proportion of them
  - The file that causes a problem often causes other similar problems

Fuzz strategy

- For the same files, only use one mutation algorithm for fuzzing within a period of time
  - If there are still more new outputs after a week, continue to fuzz
  - If there are almost no new outputs after a week, switch to another mutation algorithm

- Categorize the size of seed files
  - Such as 0-100KB, 101-400KB, 401-1024KB, >1MB
  - Test each seed set of a specific size in a specific period of time

- The same files will be tested in full patch and no patch environments, in Chinese and English environments and in x86 and x64 environments

Crash management

- How to manage crash files is very important, I mainly considered the following conditions
  - How to merge the same cases generated in different fuzz machines
  - How to exclude crash cases that have appeared before from the newly added crash files

- Use a FTP server to receive crash files across virtual machines
  - If a crash file has the same crash address with a previous file, the server will reject it
  - Save all crash files processed by the reproducer to a local "database", only those newly appeared crash files need to be examined

Results

- After half a year of fuzzing(from May 2020 to October 2020), I reported a total of 20 Excel vulnerabilities to Microsoft
  - 2 of them were marked as "Valid" but will not be fixed immediately
  - 1 was marked as "Won't fix“
  - The remaining 17 vulnerabilities are all fixed(1 of them is duplicate)

- These vulnerabilities help me receive 16 CVE acknowledgements from Microsoft

Results

MSRC Case ID	Vulnerability Type	Effect Version	Impact	CVE
58769	OOB Read	All	RCE	CVE-2020-1495
58805	OOB Read	All	RCE	CVE-2020-1496
58974	OOB Read	All	Info Leak	CVE-2020-1497
59124	OOB Read	All	RCE	CVE-2020-1498
59203	OOB Read	Office2010	RCE	CVE-2020-1504
59378	OOB Read	All	Info Leak	CVE-2020-1224
59494	Unallocated Memory Write	All	RCE	CVE-2020-1494
59482	OOB Read	All	RCE	Won't Fix
59646	OOB Read	All	RCE	Valid
59663	OOB Read	All	RCE	CVE-2020-1335
60883	OOB Read	All	RCE	Valid
60594	UAF Read	All	RCE	CVE-2020-17064
61205	OOB Read	All	Info Leak	CVE-2020-17126
60654	UAF Read	All	RCE	CVE-2020-17065
60979	UAF Read	Office2010	RCE	CVE-2020-17066
61030	UAF Read	Office2010	RCE	CVE-2020-17122
61223	UAF Read	Office2010	RCE	CVE-2020-17127
61460	OOB Write	All	RCE	CVE-2020-17129
61461	UAF Read	Office2010	RCE	Duplicate
61646	Unallocated Memory Free	All	RCE	CVE-2021-1714

CVE-2020-1494

- CVE-2020-1494 is an unallocated memory write issue in Excel.exe

```
(12a8.da0): Access violation - code c0000005 (first/second chance not available)
For analysis of this file, run !analyze -v
eax=02f842ec ebx=53348fc8 ecx=00004f00 edx=00004f00 esi=02f7f3ec edi=41004f00
eip=6a7b2dae esp=02f7f36c ebp=02f7f38c iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210203
VCRUNTIME140!memmove+0x4e:
6a7b2dae f3a4            rep movs byte ptr es:[edi],byte ptr [esi]

0:000> dc edi
41004f00  ???????? ???????? ???????? ????????  ????????????????
41004f10  ???????? ???????? ???????? ????????  ????????????????
41004f20  ???????? ???????? ???????? ????????  ????????????????
41004f30  ???????? ???????? ???????? ????????  ????????????????
41004f40  ???????? ???????? ???????? ????????  ????????????????
41004f50  ???????? ???????? ???????? ????????  ????????????????
41004f60  ???????? ???????? ???????? ????????  ????????????????
41004f70  ???????? ???????? ???????? ????????  ????????????????
```

CVE-2020-17126

- CVE-2020-17126 is an out of bound read issue in Excel.exe

```
(ddc.1678): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=5d1a10b8 ebx=00ce8354 ecx=000000b8 edx=00000150 esi=5d1a1000 edi=4e19cf48
eip=657f36fe esp=00ce6794 ebp=00ce67ac iopl=0         nv up ei pl nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010203
VCRUNTIME140!memmove+0x4e:
657f36fe f3a4            rep movs byte ptr es:[edi],byte ptr [esi]

0:000> !heap -p -a edi
    address 4e19cf48 found in
    _DPH_HEAP_ROOT @ d01000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -         VirtAddr         VirtSize)
                                5bba3b94:         4e19cea8              158 -         4e19c000             2000
    5873ab70 verifier!AVrfDebugPageHeapAllocate+0x00000240
    770090bb ntdll!RtlDebugAllocateHeap+0x00000039
    76f5349d ntdll!RtlpAllocateHeap+0x000000ed
    76f5214b ntdll!RtlpAllocateHeapInternal+0x000006db
    76f51a46 ntdll!RtlAllocateHeap+0x00000036
    5467cadf mso20win32client!Ordinal951+0x00000034
    ...cut...
```

CVE-2020-17127

- CVE-2020-17127 is an use after free read issue in Excel.exe, it is a nice UAF

```
(518.1010): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=049c6e94 ebx=0429cd90 ecx=04a20e28 edx=01700000 esi=049c6dc8 edi=11bea880
eip=2fadc12e esp=006f1fbc ebp=006f24ac iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00210246
Excel!Ordinal40+0x19c12e:
2fadc12e 8b01            mov     eax,dword ptr [ecx]  ds:0023:04a20e28=????????

1:014> u eip
Excel!Ordinal40+0x19c12e:
2fadc12e 8b01            mov     eax,dword ptr [ecx]
2fadc130 51              push    ecx
2fadc131 ff5008          call    dword ptr [eax+8]
2fadc134 c3              ret
2fadc135 a130039c30      mov     eax,dword ptr [Excel!DllGetLCID+0xd1ef7 (309c0330)]
2fadc13a 050c030000      add     eax,30Ch
2fadc13f 833800          cmp     dword ptr [eax],0
2fadc142 7468            je      Excel!Ordinal40+0x19c1ac (2fadc1ac)
```

Limitations

- My fuzz method is aimed at the vulnerabilities that affect all office versions. Due to the limitations of my testing methodology, those vulnerabilities that only exist in the latest version of Office but not in the lower version of Office cannot be found through my fuzzing method

- If the current disk can be replaced with SSD, the file read/write speed will be significant increase

- The mutation algorithm can still be improved. If I continue adding better mutation algorithms to the current fuzz framework, it can further improve the results

Limitations

- The start and stop time of Excel process is too expensive. If there is a better way for simulating Excel execute process, it will significantly reduce the opening and closing time of the Excel process, and the fuzz speed can be greatly improved

- The corpus distillation method in this presentation uses static code coverage statistics. Compared with dynamic coverage statistics, this statistical method has lower coverage accuracy. Only a rough coverage assessment can be done, so there is room for improvement

- The initial seed set used by me is limited. If all non-malware xls files on VirusTotal can be used for corpus distillation, the coverage result will be better, and there will be more results

Acknowledgements

- Special thanks to Jaanus Kaap's blogs, the presentation he shared at the POC2018 Conference, his Vanapagan project and the XLS seed files he shared, all of which helped me a lot

- Special thanks to @hackyzh, some of his fuzz ideas have inspired me a lot

Thank You

For your attention