CONCLUSION

We've identified both software and hardware vulnerabilities, affecting Qualcomm's TEE named QSEE, as implemented on Qualcomm IPQ40xx-based devices. These vulnerabilities enable an attacker to execute arbitrary code at the highest privilege level. We reported all vulnerabilities to Qualcomm using a responsible disclosure process.

We've identified the software vulnerabilities by reverse engineering the QSEE binary that we've extracted from multiple devices. Even though these vulnerabilities were critical, they can be easily fixed using a software update, which can be distributed to devices already in the field. Therefore, we anticipate these vulnerabilities to be fixed in the future.

However, the hardware vulnerability, which can be exploited using EM glitches, cannot be easily mitigated, especially not for devices already in the field. Qualcomm indicated to us that these types of attacks are outside of the IPQ40xx's threat model. Therefore, an attacker capable of injecting EM glitches, will always be able to break into QSEE, without relying on any software vulnerability.

The impact of software vulnerabilities is typically much larger than hardware attacks that require physical access to a device. Mass exploitation is for example typically not possible.

Nonetheless, we like to stress that hardware attacks should not be immediately omitted from the threat model of a device. They are often used by attackers to get access to secured code or data in order to perform subsequent research during which easier to exploit (software) vulnerabilities are identified. □

HOW I FOUND 16 MICROSOFT OFFICE EXCEL VULNERABILITIES IN 6 MONTHS

Quan Jin

INTRODUCTION

At the beginning of 2020, I decided to learn something about fuzzing. I first read some papers about fuzzing, include "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

There are many fuzz tools for linux platform, such as AFL, LibFuzzer and Honggfuzz, but there are less fuzz tools on Windows. WinAFL is a great tool, however it cannot handle large and complex software such as Microsoft Office.

Over the past three years, hundreds of bugs on Windows were found by WinAFL, which means there are basically no chance to find more bugs through it. Some researchers make some improvements on WinAFL, and find more bugs based on their custom WinAFL.

From my perspective, I want to choose a target which is less targeted by WinAFL and I'm familiar with this target.

Choose a Target

There are several candidates: Adobe Reader, Internet Explorer and Microsoft Office. Let's review them one by one.

- 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

It seems that Microsoft Office is a good candidate. But here comes two questions:

1. Is it possible to find a bug in Microsoft Office on several months for a newcomer in fuzzing?
2. Microsoft Office consists of multiple components, should I choose Word, PowerPoint, Excel or another component to fuzz?

Let me first answer the first question. I'm a newcomer in fuzzing, but I have extensive experience in office vulnerability analysis. So, it's possible for me to find a bug in Microsoft Office.

To answer the second question, I counted the Microsoft Office CVE numbers and their distribution from 2017 to 2020. The initial statistical time is up to April 2020, I updated the statistical data in June 2020.

Here is the up to June 2020 statistical results:

	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.

We can learn something from the table:

1. Around 2018, Microsoft made a change to the disclosure name of Office vulnerabilities to make the classification more detailed;
2. From 2017 to 2020, the Excel component has the most vulnerabilities almost every year;
3. From 2017 to 2020, the PowerPoint component has the least vulnerabilities almost every year

If a security researcher invests the same amount of time in security testing for each Office component, Excel is obviously the most hopeful one, and PowerPoint is the least. Word and Outlook are in the middle. If I can choose only one target, it will be Excel.

METHODOLOGY AND IMPLEMENTATION

Now, I have selected Excel as my target. Before starting fuzzing, I need to evaluate the feasibility of the basic steps involved in Excel fuzz. A common fuzzing process usually includes the following stages:

1. Seeds - How to collect seeds?
2. Mutator - How to mutate?
3. Detection - How to catch exceptions?
4. Triage - How to classify and de-duplicate crash files?
5. Reproducer - How to reproduce the crash?
6. Report - How to report the vulnerability to the vendor?

Let's examine them one by one.

Seeds

Before fuzzing Excel, I need to collect some Excel files as seeds. After counting the file types involved in the Excel vulnerabilities announced by ZDI in the last 3 years, I realized that the proportion of vulnerabilities in the OpenXML format is far less than that of the OLE2 format, So I began to focus on xls files. After some exploration, the source of my seeds is as follows:

1. Contextures (https://www.contextures.com)
2. Vertex42 (https://www.vertex42.com)
3. Excel files provided by Jaanus Kaap (https://foxhex0ne.com)

Many fuzz tutorials tell us that the more files are not the better, nor the bigger the better. So I need to minimize the collected Excel files. If the fuzz tool is WinAFL, you can use the built-in components to distill the seed files. I don't want to use WinAFL, so I need to implement this function by myself.

While trying to solve the above problem, I saw two blogs by Jaanus Kaap:

- Let's get things going with basics of file parsers fuzzing
- Let's continue with corpus distillation

Unfortunately, at the time of writing this presentation, these blogs are no longer accessible, but I read these two articles in detail at that time.

Although it is no longer possible to obtain relevant knowledge from the author's blog, Jaanus Kaap once shared his experience at the POC2018 Conference entitled □

"Document parsers 'research' as passive income."

However, the ideas of corpus distillation are similar between different tools: for the software you want to fuzz, first select a module, then use the tools and initial seeds to make statistics on the module coverage. The goal of this is to select the smallest number of files with the highest module coverage, and hope that these files are as small as possible.

With the help of static count and dynamic execution, I distilled a set of Excel seeds in an acceptable time as the initial seeds for my fuzzing.

Mutation

Mutation algorithm is an important part of fuzz, and its quality directly affects the result of fuzz.

I transplant the following mutation algorithms in Honggfuzz:

- mangle_Bit
- mangle_IncByte
- mangle_DecByte
- mangle_NegByte
- mangle_Bytes
- mangle_ASCIINum
- mangle_CloneByte
- mangle_AddSub

For the remaining mutation methods in Honggfuzz, after careful evaluation, I chose not to transplant.

I also integrate all the values of the byte's replacement part of AFL, LibFuzzer and Honggfuzz, and construct a mutation value replacement table covering these three fuzzers.

Detection

The detection part can be simply abstracted into automatic start of the program, open the file, monitor process and catch the exception. There are many good solutions on Github, which are generally implemented by winappdbg or pydbg.

Vanapagan by Jaanus Kaap is a good example.

In order to improve the catch rate of heap memory access exceptions, I use Global Flags to enable Page Heap for Excel process.

Triage

During the fuzzing process, hundreds of crash files will be collected, how to effectively classify them is a science. Based on existing experience, I mainly pay attention to the following conditions:

1. Access violation: the exception code is 0xC0000005. Microsoft does not accept stack exhaustion vulnerabilities such 0xC00000FD;
2. Non-null pointer reference exception: Microsoft does not accept null pointer reference vulnerabilities

Based on the above considerations, my classification rule is to distinguish a null pointer reference from a non-null pointer reference, distinguish access violation from other exception types. Under this rule, those non-null pointers with an exception code of 0xC0000005 are the crashes that I need to focus on.

My classification rule for Microsoft Office Excel exceptions for Non-null pointer reference is as follow:

- Read access violation
  - Out-of-bound read
  - Use-after-free read
- Write access violation
  - Out-of-bound write
  - Use-after-free write

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 can be reproduced. So I write a reproducer based on my fuzzer. This reproducer is used to reproduce the crash results in various full patch Office environments and record the reproduced results. I make multiple Office environments to reproduce the crash files. Including but not limited to these:

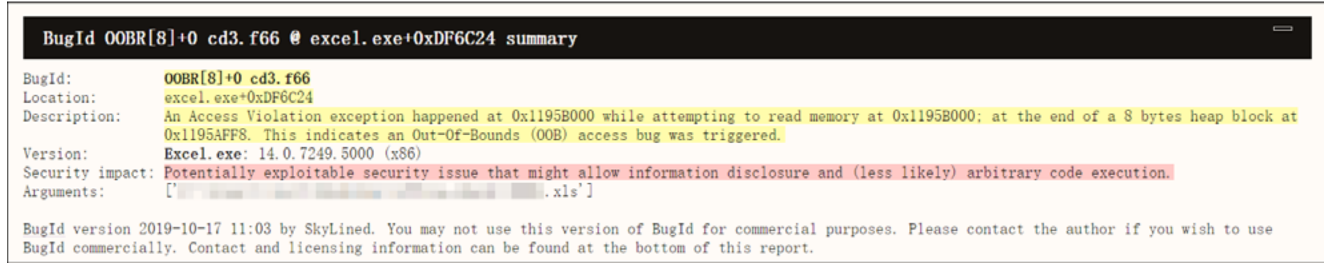
- Office 2007 - no patch & full patch
- Office 2010 - no patch & full patch
- Office 2013 - no patch & full patch
- Office 2016 - no patch & full patch
- Office 2019 - no patch & full patch

For those reproduced by the reproducer, I will perform some manual check. If both pass, these files are regarded as valid vulnerability files.

Report

When a crash file is successfully reproduced, it can be automatically generated a professional report with the help of BugId. It should be noted that BugId can only run on Windows 10, so a "Windows 10+Office environment" with the latest Office and full patch version need to be made.

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.
```

Once you have the BugId report, you can submit the vulnerability to MSRC:

- MSRC Researcher Portal
- The specific format of the vulnerability report can be referred to here
- The poc and BugId reports can be uploaded as attachments.

EQUIPMENT

I have a laptop for reproduction and report generation. These are all my fuzzing 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

PROBLEMS

Throughout the process, I encountered at least the following problems:

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

Dialog Click

The Excel software has various dialog boxes during the excuting process. Some dialog boxes such as “Safe Mode” can be resolved by cleaning the registry, while others need to be manually clicked.

My way of solving these dialog boxes are as follows:

- 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
- 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:

These methods can only handle most of the dialog box click problems, there are still some dialog boxes that I cannot solve, but there is no need to be perfect, it is enough to do these.

Virtual Machine Size

I use VMware to fuzz. 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.

Over time, the disk overhead of the host will increase significantly(usually several to dozens of GBs per virtual machine).

In order to solve this problem, you must ensure that the current fuzzer has effectively cleaned up the files generated by the previous fuzz before starting the next file, mainly the following folders:

- %AppData%\Local\Temp
- %AppData%\Roaming\Microsoft\Office\Recent
- %AppData%\Roaming\Microsoft\Windows\Recent

Otherwise, once the number of fuzz executions increases, the size of the virtual machine will explode. As a result, the fuzzer will stop.

In addition to above operations, I also use Dism++ tool to regularly clean up the temp files inside each virtual machine, and configure the virtual machine to automatically clean up the disk after shutting down.

In this way, the size of each virtual machine will be automatically reduced after shutdown, and the size of each virtual machine can be restored to the original size after a fixed interval (such as a few weeks), thus creating a basis for continuous fuzzing.

Speed of execution

When other conditions remain unchanged, the speed of fuzzing directly affects the output efficiency.

After some testing and evaluation, I think the main factors affecting Excel fuzz are as follows:

- 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
  - » There are many versions of Office. The higher the version, the slower the opening speed. From another perspective, the higher the version, the larger the amount of code and the number of potential vulnerabilities. I need to make some trade-offs. After a period of evaluation, I decide to focus on vulnerabilities which exists from Office 2007 to Office 2019.
  - » After making this choice, I can speed up fuzzing by choosing to execute the file in a lower Office version. Although Office 2007 / Office 2010 have successively withdrawn from the support list, they are useful if the crash file which collected in these environments can affect the latest version of Office software.
  - » The main fuzz environment I finally chose is Office 2010. After many fine-tuning, my fuzzer can be stabilized on 10 virtual machines, and each virtual machine executes an average of 15w files per day, that is, runs about 15w files per day.
- The stability of fuzzer
  - » If a fuzzer is unstable and crashes itself when executing, that is sad. Some fuzzers that use winappdbg may have this problem on x64 environment, so 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
  - » This problem was discovered through observation. My fuzzing environment uses HDD. When using VMware to open multiple virtual machines (I open up to 11 virtual machines on a single computer), disk IO will become very stuck.
  - » Due to the limitation of disk IO, the fuzzing of inner virtual machines will cause VMware itself to hang on for a long time, which significantly affect the fuzz speed. Sometimes the fuzzing in a single virtual machine ends abnormally.
  - » It is necessary to clean up the environment in the virtual machine and restart the fuzzing, or restart the related virtual machine to resume the fuzzing. This process is a waste of time. I think SSD will improve a lot.
- CPU Cores, RAM and Primary Frequency
  - » CPU Cores, RAM and Primary Frequency: The number of CPU cores and the RAM capacity directly determine the maximum number of virtual machines that can be opened at the same time. The bigger the two indicators, the better. The primary frequency directly affects the opening speed of the program. The bigger the primary frequency, the better.

Version Switching

During the fuzzing, it is necessary to consider the inconsistency of processing the same file by different architectures (x86 and x64), different patches(no patch and full patch), and different language versions (Chinese and English). I mainly consider the following scenes:

- 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

Therefore, I test the above scenes with each set of seed files, and gain some extra crashes.

Fuzz Strategy

I think fuzz strategy is the most important part of my Excel fuzzing. 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:

1. Obtain as much vulnerabilities as possible in the shortest time
2. Find vulnerabilities that exist in all versions of Office

This forces me to do many thoughts and explorations on how to configure fuzz strategies, my experience on fuzz strategies including but not limited to the following:

- Skip the first 512 bytes of the header of the OLE2 file during mutation to improve the effectiveness of the mutation;
- Use an older version of Office for fuzzing to improve the speed of fuzzing;
- Use smaller Excel files for fuzzing to increase the speed of fuzzing;
- Use Google to collect xls files which were made with old versions of Excel in the 1990s and 2000s, and add them to the initial seed collection;
- Select Office attack surface that may cause problems based on my experience (e.g. pivot table), then select related files for fuzzing;
- 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, because the file that causes a problem often causes other similar problems;
- For the same files, only use one mutation algorithm for fuzzing within a period of time, and continue to observe the effectiveness of the current mutation algorithm. 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, and 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

As more and more results are obtained from fuzzing, how to manage these crash files has become a very important thing. I mainly consider 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

Regarding how to merge the same cases generated in different fuzz machines, I have explained in the section “Methodology & Implementation - Triage” above.

I use a FTP server to receive crash files across virtual machines, if a crash file has the same module and the same crash address with a previous file, the server will reject it.

Every once in a while, I will drag out all the crash files in the FTP server and reproduce them in a full patch environment with the help of my reproducer (I make several full patch environments, only one is frequently used).

Only those newly appeared crash files need to be examined. Therefore, I use a python script to save all crash files processed by the reproducer to a local “database”(this database is just a simple folder list, but it is very effective).

When the number of crash case in the database becomes more and more, the newly appeared crash files will be fewer and fewer, at the same time, the vulnerability rate of these new files will be higher and higher.

RESULTS

After half a year of fuzzing (from 2020.05 to 2020.10), I reported a total of 20 Excel vulnerabilities to Microsoft.

Two of them were marked as "Valid" but will not be fixed immediately, one was marked as "Won't fix", and the remaining 17 vulnerabilities are all fixed, and helped me receive 16 CVE acknowledgements from Microsoft (one of them is duplicate).

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

Note: "ALL" refers to Office2010, Office2013, Office2016, Office2019

Note: Case 61461 has been fixed in the January 2021 patch but it is duplicate, I have not tracked down its corresponding CVE number.

Below I share some cases found by my fuzzer.

- **CVE-2020-1494** is an unallocated memory write issue in excel.exe.
- **CVE-2020-17126** is an out of bound read issue in excel.exe.
- **CVE-2020-17127** is an use after free read issue in excel.exe, it is a nice UAF.

CVE-2020-1494

```
(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

```
(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)
4e19c000             2000          5bba3b94:         4e19cea8              158 -
    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...

0:000> !heap -p -a esi
    address 5d1a1000 found in
    _DPH_HEAP_ROOT @ d01000
    in busy allocation (  DPH_HEAP_BLOCK:         UserAddr         UserSize -
VirtAddr         VirtSize)
5d1a0000             2000          24d62270:         5d1a0f58               a8 -
    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

```
(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> !heap -p -a ecx
    address 04a20e28 found in
    _DPH_HEAP_ROOT @ 1701000
    in free-ed allocation (  DPH_HEAP_BLOCK:         VirtAddr         VirtSize)
                                 4952d00:          4a20000             2000
    61e0adc2 verifier!AVrfDebugPageHeapFree+0x000000c2
    77d99913 ntdll!RtlDebugFreeHeap+0x0000003e
    77cdfb7e ntdll!RtlpFreeHeap+0x000000ce
    77cdfa46 ntdll!RtlpFreeHeapInternal+0x00000146
    77cdf49e ntdll!RtlFreeHeap+0x0000003e
79645cc3 mso!Ordinal149+0x000078ef
...cut...

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

For now, my fuzz method has the following shortcomings:

1. As mentioned earlier, in order to improve the speed and efficiency of fuzzing, I selectively ignored some potential vulnerabilities in terms of strategy (such as vulnerabilities only in the newer Office version). The fuzz method in this presentation 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;
2. If the current disk can be replaced with SSD, the file read/write speed will be significant increase, which can improve the fuzzing speed;
3. The mutation algorithm can still be improved. According to observations, after transplanting the Honggfuzz mutation algorithm to my custom fuzzer, the fuzz output has increased significantly, which shows that an effective mutation algorithm can greatly improve the fuzz output. If I continue adding better mutation algorithms to the current fuzz framework, it can further improve the results;
4. 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;
5. 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;
6. The initial seed set used by my fuzzer 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 output

ACKNOWLEDGEMENTS

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

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