Fuzzing everything in 2014: a (mostly) director's perspective

Alisa Esage

About me

- Hacking binary since 15
- Left my first=last employer in 2004, independent ever since
- Done: binary reversing to malware analysis to cyber investigation, pentesting to blackbox auditing to vulnerability discovery to exploitation…
- Founded: Esage Lab => Neuronspace, Malwas, TZOR

Section 1: hacker's

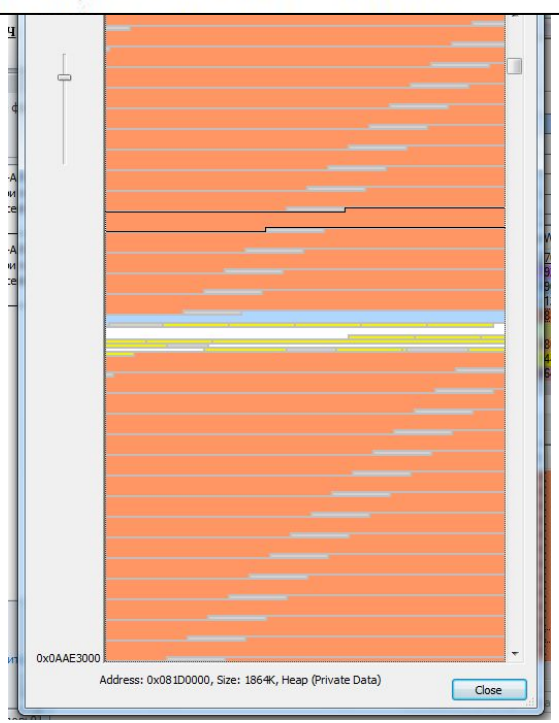
A BIT AWAY FROM A 0-DAY…

Microsoft Word 2007/2010 E

```
!exploitable 1.6.0.0
Exploitability Classification: EXPLOITABLE
Recommended Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symbol
called from user32!gapfnScSendMessage+0x00000000

User mode DEP access violations are exploitable.
ANALYSIS END
quit:
```

```
0:000> kb
ChildEBP RetAddr  Args to Child
WARNING: Frame IP not in any known module. Following frames may be wrong.
0023d680 7606c4e7 0004016a 0000031f 00000001 0x5701001
0023d6ac 76085b7c 05700ff0 0004016a 0000031f USER32!InternalCallWinProc+0x23
0023d728 760859f3 00000000 05700ff0 0004016a USER32!UserCallDlgProcCheckWow+0x132
                                    0001 USER32!DefDlgProcWorker+0xa8
                                    0001 USER32!DefDlgProcW+0x22
                                    031f USER32!InternalCallWinProc+0x23
                                    016a USER32!UserCallWinProcCheckWow+0xe0
                                    d8b0 USER32!DispatchMessageWorker+0x35e
                                    d8d8 USER32!DispatchMessageW+0xf
                                    0000 wwlib!GetAllocCounters+0x4d646
                                    5175 wwlib!GetAllocCounters+0x4bfdb
                                    d.  Defaulted to export symbols for winword.exe -
0023fa6c 2f091ec2 2f090000 00000000 01382211 winword!wdGetApplicationObject+0x63a
0023fafc 7692ed5c 7ffd5000 0023fb48 77c637eb winword!wdGetApplicationObject+0x894
0023fb08 77c637eb 7ffd5000 77e3daa5 00000000 kernel32!BaseThreadInitThunk+0xe
0023fb48 77c637be 2f092045 7ffd5000 ffffffff ntdll!__RtlUserThreadStart+0x70
0023fb60 00000000 2f092045 7ffd5000 00000000 ntdll!_RtlUserThreadStart+0x1b
```

```
        Откуда взялся адрес 05700ff0?

0x56f0000 - адрес маппинга C:\Windows\[redacted].dll.mui (размер 0x1000)
0x5700ff0 = 0x56f0000+0x1000+0xfff0

Иногда смещение другое:
0:000> ? 0x7370ff0-0x7350000
Evaluate expression: 135152 = 00020ff0
```




```
InternalCallWndProc вызывает адрес (база mui)+0x1000+0xfff0:
0d4e0ff0 e00f            loopne  0d4e1001
0d4e0ff2 4e              dec     esi
0d4e0ff3 0d70d71300      or      eax,13D770h
0d4e0ff8 e9a6367d5e      jmp     [redacted]!DllUnregisterServer+0x18b
0d4e0ffd 0000            add     byte ptr [eax],al
0d4e0fff 00              ???
0d4e1001 ??              ???

"loopne 0d411001" = (0d4e)0fe0 = указатель на предыдущий heap chunk
```

```
03360ff0 c7442404b8977200 mov     dword ptr [esp+4],7297B8h
03360ff8 e9a636ac64       jmp     [redacted]!DllUnregisterServer+0x18b34 (67e246a3)
```

Section 2: engineer's

THE IDEAL FUZZER

Problems with fuzzers

1. Too specialized.

E.g. fuzz only browsers, or only files

Not suitable for fuzzing everything by design

2. Enforce unnecessary constraints.

E.g. glue mutation with automation with crash monitoring

Kills flexibility => not suitable for fuzzing everything

3. Steep learning curve.

E.g. templates & configs

Is it worthy to learn a system which is constrained anyway?

What I want (from a fuzzer)

Omnivore.

Target invariant: software type, data type, platform, architecture

Omnipresent.

Hosting platform invariant: VM/hardware/laptop/localnet/clouds...

“LEGO”

Mix & match components

Rapid support for new targets

Hot patching for tweaking

What I want, cont'd

Autonomous.

Can leave it for a week?

Just runs

Unlimited, native scaling.

Any number of fuzzers running at the same time

0 time to set up new targets

Right now.

No time for development

Key design decisions

Network client-server architecture
Build upon isolated, generic tools

Native automation

bash, cmd/PowerShell, cscript/wscript, AppleScript…

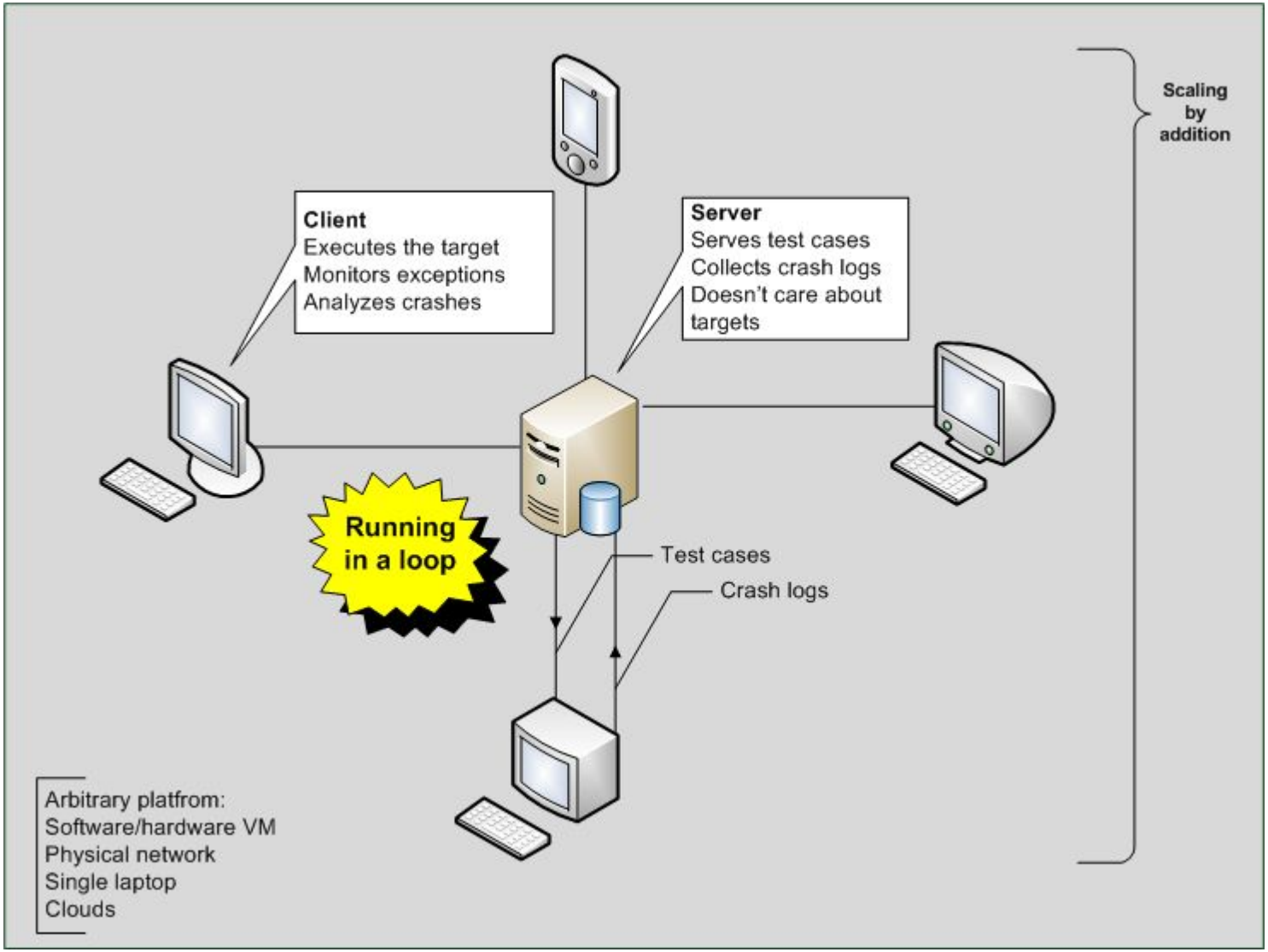
Native instrumentation

DebugAPI, CrashWrangler, cdb postmortem scripts…

Highly generic mutators

Home-made bitflipping tools, grep/sed/urandom, radamsa…

Done


Scaling by addition
Client
Executes the target
Monitors exceptions
Analyzes crashes
Server
Serves test cases
Collects crash logs
Doesn't care about targets
Running in a loop
Test cases
Crash logs
Arbitrary platfrom:
Software/hardware VM
Physical network
Single laptop
Clouds

Results

```
aware@Gromozeka:~/fuzzing-server$ ./estats ./results
      9 APP:  cscript.exe
      7 APP:  excel.exe
    107 APP:  viewer_.exe
      2 APP:  viewer.exe
    256 APP:  winword.exe

    539 //
      2 cscript_exe/5_8_7600_16385/
      7 cscript_exe/5_8_7601_18283/
     14 EXCEL_EXE/12_0_6683_5002/
      5 ISSymbol_ocx/1201_1404_202_0/
      2 kernel32_dll/6_1_7601_18409/
      5 mfc90u_dll/9_0_30729_6161/
     63 MSO_DLL/12_0_6662_5000/
     92 MSO_DLL/12_0_6683_5000/
      3 MSPTLS_DLL/12_0_6682_5000/
      2 msvcr90_dll/9_0_30729_6161/
     42 msvcrt_dll/7_0_7601_17744/
      1 NPSVG3_dll/3_0_0_94/
      2 ntdll_dll/6_1_7600_16915/
      8 ntdll_dll/6_1_7601_18247/
      4 ole32_dll/6_1_7601_17514/
     52 unknown/0_0_0_0/
    107 Viewer__exe/1201_1404_202_0/
      2 Viewer_exe/1201_1404_202_0/
     69 WINWORD_EXE/12_0_6668_5000/
    115 WINWORD_EXE/12_0_6690_5000/
     72 WINWORD_EXE/12_0_6695_5000/
      4 WINWORD_EXE/14_0_7113_5001/
      6 WWLIB_DLL/12_0_6668_5000/
      1 wwlib_dll/12_0_6690_5000/
     22 WWLIB_DLL/12_0_6690_5000/
     68 WWLIB_DLL/12_0_6695_5000/

      3 Exploitability Classification: EXPLOITABLE
     97 Exploitability Classification: EXPLOITABLE
     34 Exploitability Classification: PROBABLY_EXPLOITABLE
     35 Exploitability Classification: PROBABLY_EXPLOITABLE
     43 Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
     80 Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
      3 Exploitability Classification: UNKNOWN
    162 Exploitability Classification: UNKNOWN

      5 Bug Title: Data from Faulting Address controls Branch Selection starting at mfc90u!ATL::CSimpleStringT<wchar_t,1>::operator=+0x000000
      2 Bug Title: Data from Faulting Address controls Branch Selection starting at MSO!MsoDwRegGetDw+0x000000000000003c (Hash=
     16 Bug Title: Data from Faulting Address controls Branch Selection starting at WWLIB!FMain+0x0000000000003bda2 (Hash=
      1 Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symbol @ 0x000000000019a1001 called from user32!gapfn
      1 Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symbol @ 0x0000000004651001 called from user32!gapfn
      1 Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symbol @ 0x0000000004861001 called from user32!gapfn
      2 Bug Title: Exploitable - Heap Corruption starting at ntdll!RtlReportCriticalFailure+0x0000000000000057 called from ole32!CRetailMallo
      3 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000020534f44 called from
      1 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x000000000240a0d0d called from
     25 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000044206e69 called from
      6 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000065622074 called from
     13 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000069206e75 called from
      1 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000075722065 called from
```

Section 3: director’s

THE MAGIC

Fuzzing in 2014

“Shellcoder’s Handbook”: 10 years ago

“Fuzzing: Brute Force Vulnerability Discovery”: 7 years ago

Dozens of publications, hundreds of tools, thousands of vulns found (=> code audited)

Driven by market and the competition

The beginner's delusion

"Success in fuzzing is defined by speed & scale"

Not exactly

ClusterFuzz is still beaten by standalone researchers

My results: ~1 night per fuzzer

Thinking

One only needs millions of test cases, if majority of those test cases are bad

Rejected by the validator or not reaching or not triggering vulnerable code paths

Cornerstone: bug-rich branches of code

Problem

No algorithm to discover “fresh” code paths

Code coverage can only measure the already reached paths

Evolutionary input generation is tiny (think Word with embeddings)

Where is the “new” code?

Code unobviously triggered or reached

Presumably effortful input generation

Presumably constrained exploit

Unobvious Examples

CVE-2013-3906: TIFF 0’day

Ogl.dll=gdiplus.dll alternative only in Office 2007

CVE-2014-0315: **Insecure Library Loading** with .cmd and .bat

CVE-2013-1324: Microsoft Word WPD stack based **buffer overflow**

Presumably Effortful Examples

CVE-2013-1296: MS RDP ActiveX Use-after-Free

No public ActiveX tools can target UaFs

Strict syntax-based and/or layered formats

My experience: the better the generator, and the deeper the targeted data layer, the more bugs found

Microsoft DKOM/RPC

Did you know one can send a DKOM/RPC request to the port mapper (135) to enable RDP?

Presumably Constrained Example

Standard ActiveX in Windows

Requires user interaction in IE

But IE is not the only wide-spread software capable of loading ActiveX…

Section 4: sponsor’s ☺️

RESULTS

Microsoft Word

```
aware@Gromozeka:~/fuzzing-server$ ./estats ./results/winword/
    183 APP:  winword.exe

    232 //
     63 MSO_DLL/12_0_6662_5000/
     91 MSO_DLL/12_0_6683_5000/
      4 ole32_dll/6_1_7601_17514/
     69 WINWORD_EXE/12_0_6668_5000/
    114 WINWORD_EXE/12_0_6690_5000/
      4 WINWORD_EXE/14_0_7113_5001/
      6 WWLIB_DLL/12_0_6668_5000/
      1 wwlib_dll/12_0_6690_5000/
     22 WWLIB_DLL/12_0_6690_5000/

      3 Exploitability Classification: EXPLOITABLE
     34 Exploitability Classification: PROBABLY_EXPLOITABLE
     31 Exploitability Classification: PROBABLY_EXPLOITABLE
     43 Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
     15 Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
      3 Exploitability Classification: UNKNOWN
    137 Exploitability Classification: UNKNOWN

      2 Bug Title: Data from Faulting Address controls Branch Selection starting at MSO!MsoDwRegGetDw+0x000000000000003c (Hash=
      1 Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symbol @ 0x00000000019a1001 called from user32!gapfnScSendMessage+0x00
      1 Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symbol @ 0x0000000004651001 called from user32!gapfnScSendMessage+0x00
      1 Bug Title: Exploitable - Data Execution Prevention Violation starting at Unknown Symbol @ 0x0000000004861001 called from user32!gapfnScSendMessage+0x00
     26 Bug Title: Probably Exploitable - Data Execution Prevention Violation near NULL starting at Unknown Symbol @ 0x0000000000000000 called from MSPTLS!LssbI
     11 Bug Title: Probably Exploitable - Data from Faulting Address controls Code Flow starting at MSO!Ordinal4480+0x00000000000002a8 (Hash=
      3 Bug Title: Probably Exploitable - Data from Faulting Address controls Code Flow starting at MSO!Ordinal4480+0x00000000000002b4 (Hash=
      6 Bug Title: Probably Exploitable - Data from Faulting Address controls Code Flow starting at WWLIB!DllCanUnloadNow+0x00000000002da852 (Hash=
      1 Bug Title: Probably Exploitable - Data from Faulting Address controls Code Flow starting at wwlib!DllCanUnloadNow+0x00000000002dc782 (Hash=
     10 Bug Title: Probably Exploitable - Data from Faulting Address controls Code Flow starting at WWLIB!DllCanUnloadNow+0x00000000002dc782 (Hash=
      8 Bug Title: Probably Exploitable - Read Access Violation on Control Flow starting at MSO!Ordinal4178+0x000000000000009f7 (Hash=
      1 Bug Title: Read Access Violation near NULL starting at cryptsp!CryptReleaseContext+0x000000000000001a (Hash=
     38 Bug Title: Read Access Violation near NULL starting at MSPTLS!LssbFIsSublineEmpty+0x0000000000004ffa (Hash=
      2 Bug Title: Read Access Violation near NULL starting at MSPTLS!LssbFIsSublineEmpty+0x0000000000045465 (Hash=
      4 Bug Title: Read Access Violation near NULL starting at ole32!CExposedDocFile::CopySStreamToIStream+0x000000000000018b (Hash=
      2 Bug Title: Read Access Violation near NULL starting at WWLIB!DllGetLCID+0x00000000001b5009 (Hash=
      6 Bug Title: Read Access Violation near NULL starting at WWLIB!FMain+0x00000000000af7e2 (Hash=
      1 Bug Title: Read Access Violation near NULL starting at WWLIB!wdCommandDispatch+0x00000000002fb222 (Hash=
      4 Bug Title: Read Access Violation near NULL starting at WWLIB!wdCommandDispatch+0x00000000002fb74f (Hash=
    137 Bug Title: Read Access Violation starting at MSO!Ordinal2669+0x0000000000000078 (Hash=
      1 Bug Title: User Mode Write AV near NULL starting at ntdll!EtwEventEnabled+0x00000000000001ca (Hash=
```

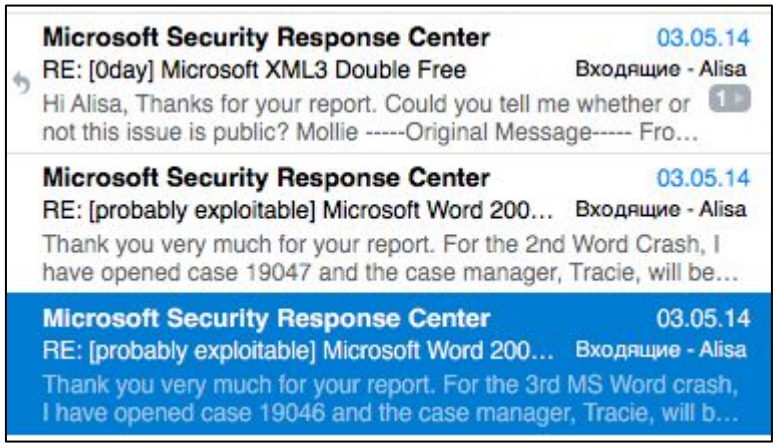
Microsoft XML

```
aware@Gromozeka:~/fuzzing-server$ ./estats ./results/xsl-test2/
      9 APP:  cscript.exe

      9 //
      2 cscript_exe/5_8_7600_16385/
      7 cscript_exe/5_8_7601_18283/
      2 ntdll_dll/6_1_7600_16915/
      7 ntdll_dll/6_1_7601_18247/

      2 Exploitability Classification: EXPLOITABLE

      2 Bug Title: Exploitable - Heap Corruption starting at ntdll!RtlReportCriticalFailure+0x0000000000000057

      2 ExceptionAddress: 77ae33bb (ntdll!RtlReportCriticalFailure+0x00000057)
      7 ExceptionAddress: 77c03873 (ntdll!RtlReportCriticalFailure+0x00000057)


      2 LAST_CONTROL_TRANSFER:  from 77ae42eb to 77ae33bb
      7 LAST_CONTROL_TRANSFER:  from 77c047a3 to 77c03873
```

Reporting & Bounty

Tag	Timestamp
Case Opened	2014-02-17 14:10 GMT-6
A case has been opened and added to the queue for review.	
Case Reviewed	2014-04-07 21:47 GMT-6
This case has been reviewed.	
Offer Made	2014-04-09 14:42 GMT-6
An offer has been made for this case. Please see our email for further details.	
Case Contracted	2014-04-10 15:13 GMT-6
This case has been officially contracted to the ZDI.	
Vendor Disclosure	2014-04-24 14:39 GMT-6
The details of this case have been submitted to the vendor as ZDI-CAN-2277.	

+ Money arrived: 2014-05-07 ($2000)



Today: 22.05.2014

“Critical infrastracture attack” contest @ PHDays: my 5 cents ☺

```
219 //
  5 ISSymbol_ocx/1201_1404_202_0/
  2 kernel32_dll/6_1_7601_18409/
  5 mfc90u_dll/9_0_30729_6161/
  2 msvcr90_dll/9_0_30729_6161/
 42 msvcrt_dll/7_0_7601_17744/
  1 NPSVG3_dll/3_0_0_94/
  1 ntdll_dll/6_1_7601_18247/
 52 unknown/0_0_0_0/
107 Viewer__exe/1201_1404_202_0/
  2 Viewer_exe/1201_1404_202_0/
  1 WINWORD_EXE/12_0_6690_5000/

 95 Exploitability Classification: EXPLOITABLE
  4 Exploitability Classification: PROBABLY_EXPLOITABLE
  3 Exploitability Classification: PROBABLY_NOT_EXPLOITABLE
  8 Exploitability Classification: UNKNOWN

  5 Bug Title: Data from Faulting Address controls Branch Selection starting at mfc90u!ATL::CSimpleStringT<wchar_t,1>::operat
  3 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000020534f44
  1 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x000000000240a0d0d
 25 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000044206e69
  6 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000065622074
 13 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000069206e75
  1 Bug Title: Exploitable - Read Access Violation at the Instruction Pointer starting at Unknown Symbol @ 0x0000000075722065
  2 Bug Title: Exploitable - User Mode Write AV starting at ISSymbol!DllCanUnloadNow+0x0000000000078f4d (Hash=
  1 Bug Title: Exploitable - User Mode Write AV starting at msvcr90!_wcwild+0x000000000000011c (Hash=
 42 Bug Title: Exploitable - User Mode Write AV starting at msvcrt!_wfindfirst64+0x0000000000000084 (Hash=
  1 Bug Title: Exploitable - User Mode Write AV starting at ntdll!LdrpResCompareResourceNames+0x000000000000001e (Hash=
  1 Bug Title: Possible Stack Corruption starting at kernel32!InterlockedDecrement+0x0000000000000009 (Hash=
  1 Bug Title: Possible Stack Corruption starting at Unknown Symbol @ 0x00000000017018e3 called from user32!DispatchHookW+0x0
  1 Bug Title: Probably Exploitable - Data from Faulting Address controls Code Flow starting at NPSVG3!DllUnregisterServer+0x
  2 Bug Title: Probably Exploitable - Read Access Violation Near Null at the Instruction Pointer starting at Unknown Symbol @
  1 Bug Title: Probably Exploitable - Read Access Violation on Block Data Move starting at msvcr90!memcpy+0x000000000000005a
  2 Bug Title: Read Access Violation near NULL starting at ISSymbol!DllUnregisterServer+0x000000000017426f (Hash=
  1 Bug Title: Read Access Violation near NULL starting at ISSymbol!DllUnregisterServer+0x000000000022f32c (Hash=
  1 Bug Title: User Mode Write AV near NULL starting at kernel32!InterlockedDecrement+0x0000000000000009 (Hash=
```

Lessons Learnt

Research! Primary target: code bases
Not data formats or data input interfaces or fuzzing automation technology
Yes: Ancient code, hidden/unobvious functionality etc.

Bet on complex data formats
For complex data, code paths exist which are **not reachable** automatically, which means probably never audited code base and zero competition.

Craft complex fuzzing seeds manually
The rule of “minimal size sample”, as stated in the book “Fuzzing: Brute Force Vulnerability Discovery” is obsolete 2014.

Remove 1-2 data format layers before injecting malformed data
Deep parsers are less audited (researchers are lazy?)
Deep parsers tend to contain more bugs (programmers are lazy?)

Estimate potency of a new vector by dumbest fuzzing prior to investing in smart fuzzing
Criteria: Bugs crowd
Bugs crowd in direction of “less audited” code base

Tweak a lot to get a “feeling” for the particular target

Keep the fuzzing setting dirty
Fuzzing is dirty by design
Pretty lying it into a well-designed system kills flexibility necessary for tweaking & rapid prototyping.

Research, again

Thank you!