Make static instrumentation great again
High performance fuzzing for Windows system

Lucas Leong

#whoami

- Security researcher from Trend Micro
- Interested in
  - vulnerability discovery
  - binary exploitation
  - reverse engineering
  - symbolic execution
- MSRC TOP 100
- HITCON CTF team

Agenda

- Motivation
- Related works
- AFL 101
- Implementation
- Benchmark
- Demo
- Case study
  CLFS, CNG, Registry
- Conclusion

Motivation

- 2014 Nov, AFL is released
- I want to fuzz windows target

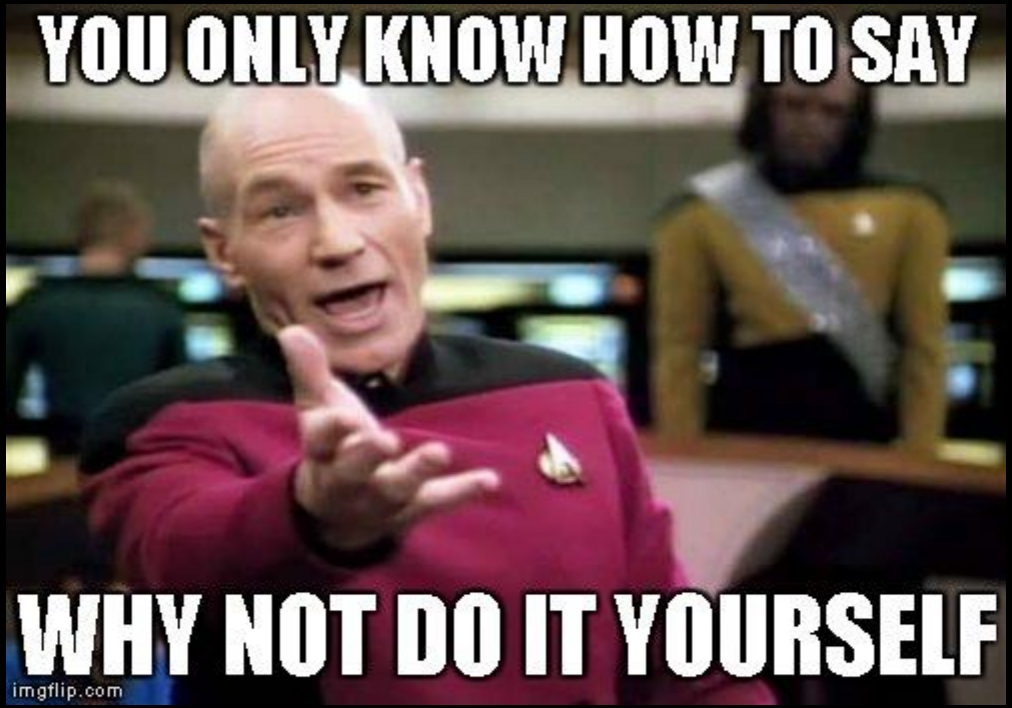
Motivation

- 2014 Nov, AFL is released
- I want to fuzz windows target
  - 2016 Jul, WinAFL is committed
- I want a better performance, support kernel

Motivation

- 2014 Nov, AFL is released
- I want to fuzz windows target
  - 2016 Jul, WinAFL is committed
- I want a better performance, support kernel
  - 2017 Jul, Static binary instrumentation via syzygy is merged
- I don’t have full PDB
- And I want more, scale up, etc

Motivation



Related works – static

- WinAFL
  - Use dynamic binary instrumentation via DynamoRIO
  - Support static binary instrumentation via syzygy
  - Require full PDB

Related works – dynamic

- DARKO
  - Static analysis via Capstone
  - Dynamic binary rewriting via Keystone
  - Cross platforms and architectures
- KFUZZ
  - Focus on windows kernel driver
  - Dynamic binary rewriting
  - Use interrupt instead of hook to solve the tiny basic block problem

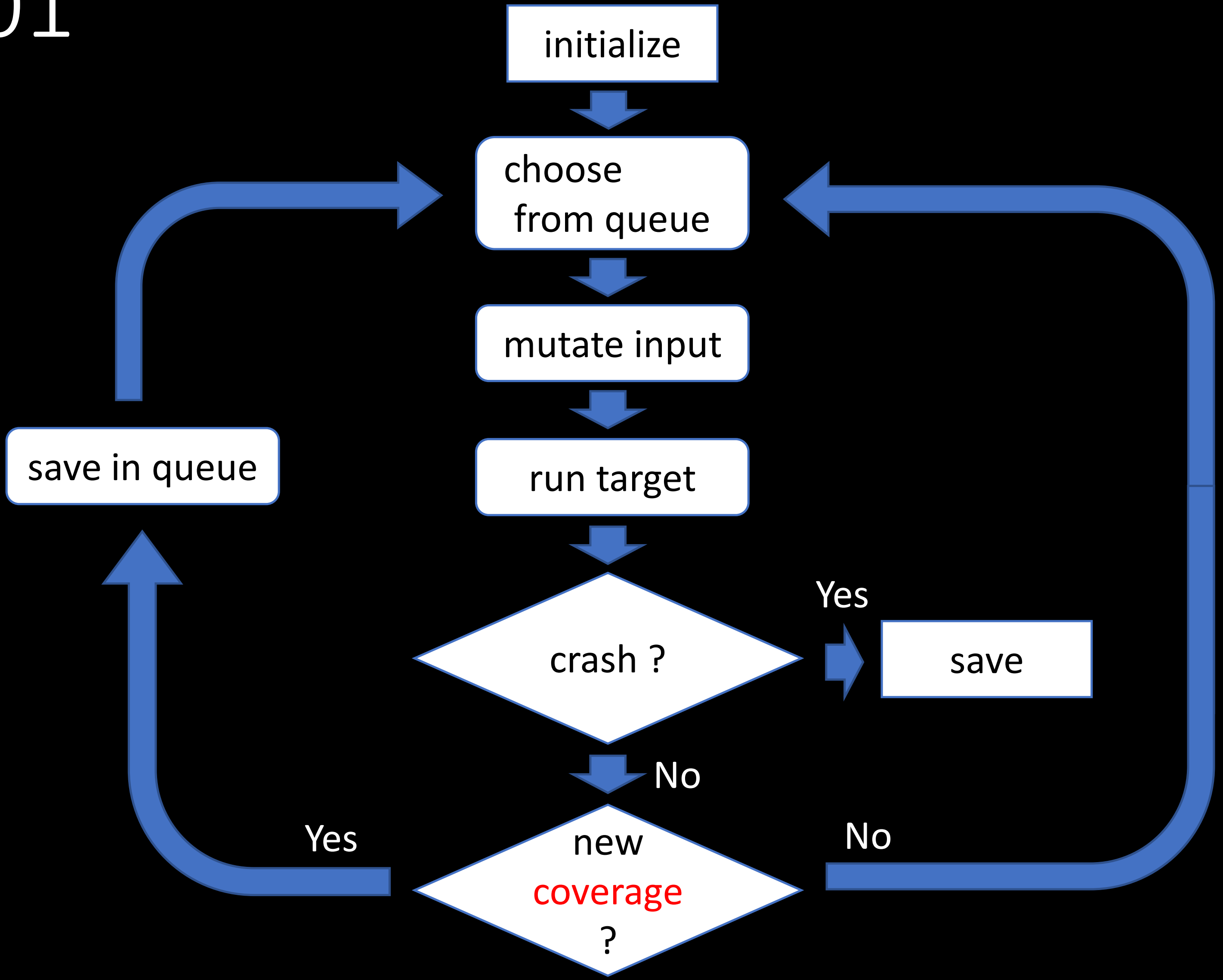
Related works – hardware

- winafl-intelpt
  - Use the built-in Intel PT driver (ipt.sys) in RS5
- kAFL
  - Combine QEMU/KVM and Intel PT
  - Scale-up and cross platform fuzzing
  - Filter with vCPU/Supervisor/CR3/IP-Range

Related works – virtualization

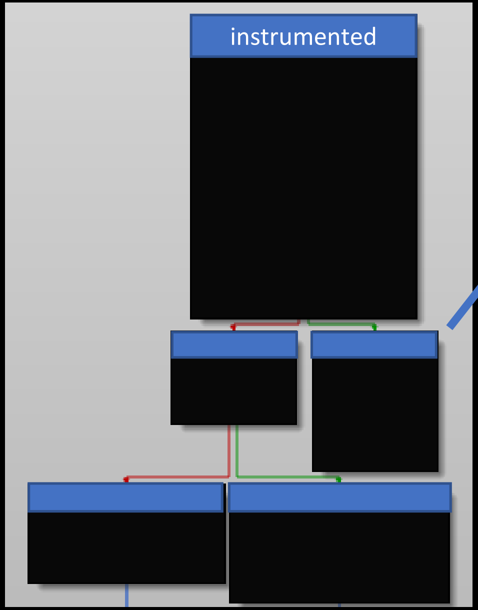
- applepie
  - Combine Bochs and WHVP API
  - Get code coverage at the hypervisor level
  - Restore snapshot with the modified pages only

AFL 101


initialize
choose from queue
mutate input
run target
crash ?
Yes
save
No
new coverage ?
Yes
save in queue
No

AFL 101

- Instrument each basic block on compile-time (afl-gcc)

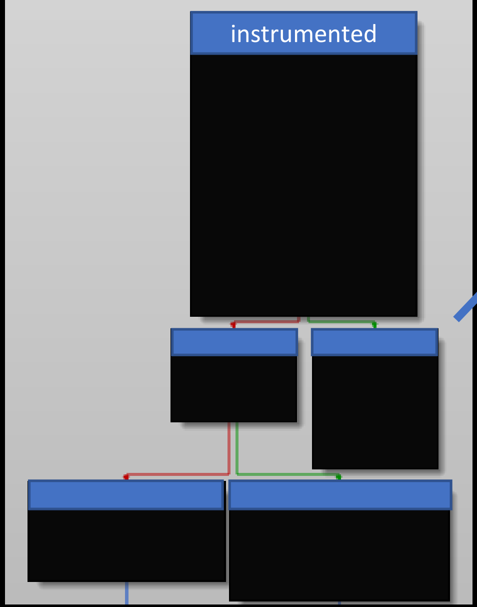


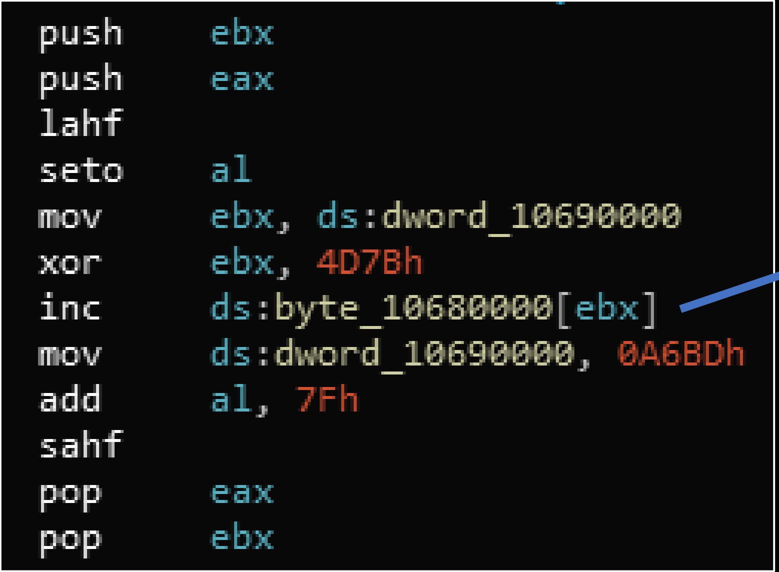
```
lea   rsp,[rsp-0x98]
mov   QWORD PTR [rsp],rdx
mov   QWORD PTR [rsp+0x8],rcx
mov   QWORD PTR [rsp+0x10],rax
mov   rcx,0x5c80
call  4009a8 <__afl_maybe_log>
mov   rax,QWORD PTR [rsp+0x10]
mov   rcx,QWORD PTR [rsp+0x8]
mov   rdx,QWORD PTR [rsp]
lea   rsp,[rsp+0x98]
```

- Record code coverage on execution-time (afl-fuzz)

Implementation – pe-afl

- Do the similar thing statically



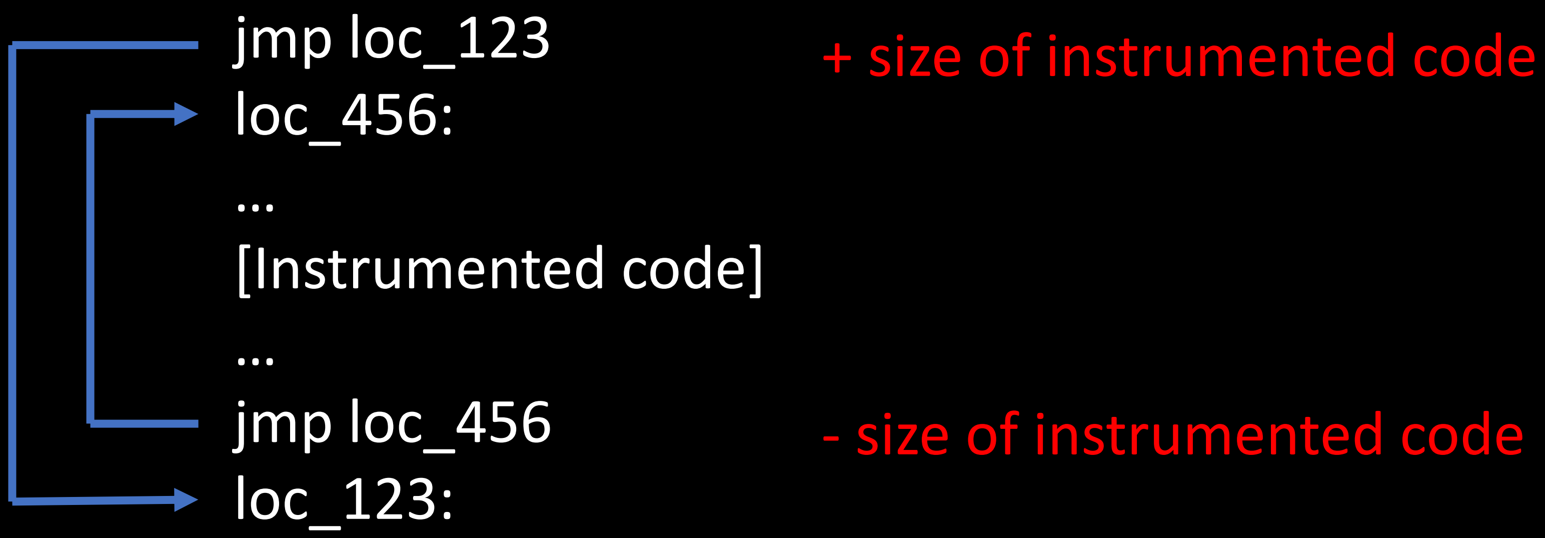


```
push    ebx
push    eax
lahf
seto    al
mov     ebx, ds:dword_10690000
xor     ebx, 4D7Bh
inc     ds:byte_10680000[ebx]
mov     ds:dword_10690000, 0A6BDh
add     al, 7Fh
sahf
pop     eax
pop     ebx
```

coverage bitmap

Implementation – pe-afl

- Expand code and update jump
    - short jump to long jump



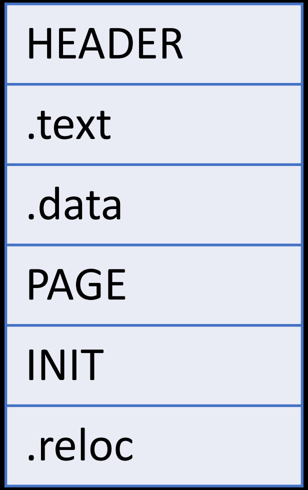
```
jmp loc_123
loc_456:
…
[Instrumented code]
…
jmp loc_456
loc_123:
```

+ size of instrumented code

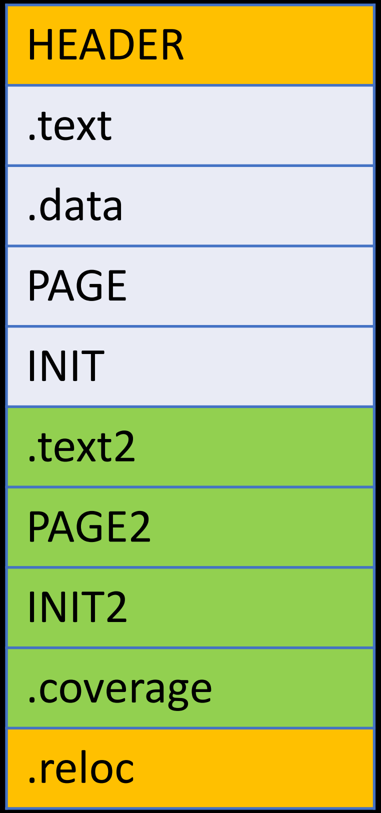
- size of instrumented code

Implementation – pe-afl

- Duplicate executable section
  - Some DATA still remains on the original section
- Append .coverage for coverage bitmap
- Update
  - PE header
  - section table
  - export table
  - SEH handle table
  - relocation table



Before instrument



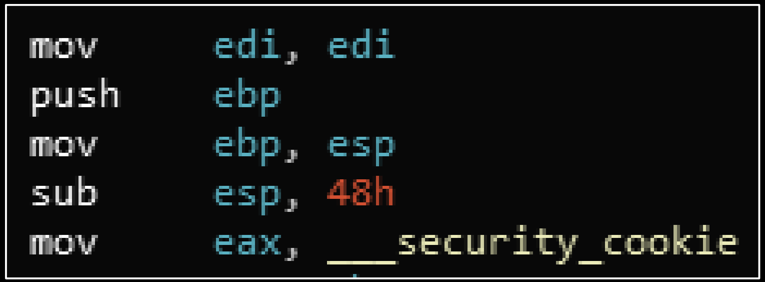
After instrument

Implementation – pe-afl

- All the static information is from IDA pro
  - basic block
  - branch
    - target address
    - op code
    - operand
  - stack frame
  - …

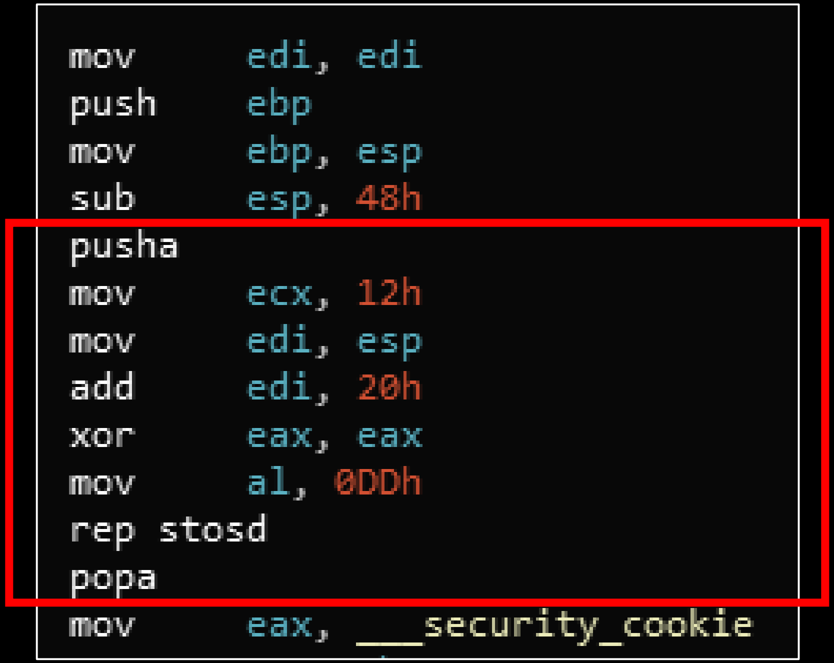
Implementation – pe-afl

- Reason to collect stack frame information



```
mov     edi, edi
push    ebp
mov     ebp, esp
sub     esp, 48h
mov     eax, ___security_cookie
```

Before stack frame poisoning

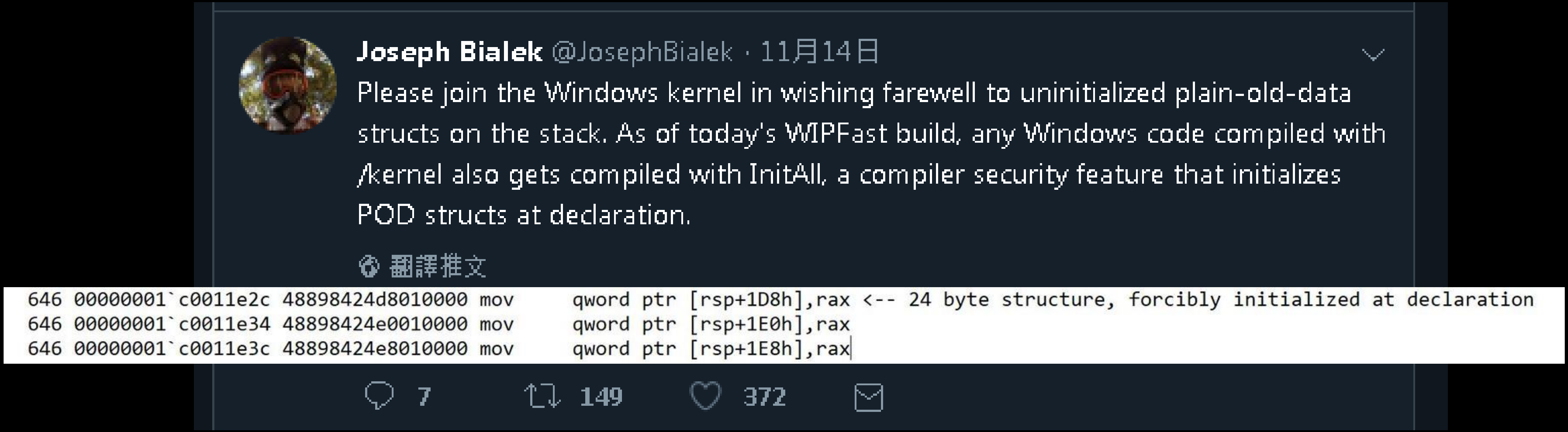


```
mov     edi, edi
push    ebp
mov     ebp, esp
sub     esp, 48h
pusha
mov     ecx, 12h
mov     edi, esp
add     edi, 20h
xor     eax, eax
mov     al, 0DDh
rep stosd
popa
mov     eax, ___security_cookie
```

After stack frame poisoning

Implementation – pe-afl

- Oops



Joseph Bialek @JosephBialek · 11月14日

Please join the Windows kernel in wishing farewell to uninitialized plain-old-data structs on the stack. As of today's WIPFast build, any Windows code compiled with /kernel also gets compiled with InitAll, a compiler security feature that initializes POD structs at declaration.

```
646 00000001`c0011e2c 48898424d8010000 mov     qword ptr [rsp+1D8h],rax <-- 24 byte structure, forcibly initialized at declaration
646 00000001`c0011e34 48898424e0010000 mov     qword ptr [rsp+1E0h],rax
646 00000001`c0011e3c 48898424e8010000 mov     qword ptr [rsp+1E8h],rax
```

Challenge for SBI

- The mix of DATA and CODE in executable section is the source of problems
    - Take care of DATA in executable section
        - 2-byte alignment for unicode string argument in WIN32 API
        - 4-byte alignment for SEHandlerTable

Challenge for SBI

- The mix of DATA and CODE in executable section is the source of problems
    - Confuse between DATA and CODE
        - Assume DATA as CODE, DATA may be corrupted

          eg. CreateFile(“ABC”) -> CreateFile(“[instrumented code]ABC”)
        - Assume CODE as DATA, coverage is missed or the execution may fail

          eg. jmp [old loc] -> jmp [old loc]

Challenge for SBI

- The mix of DATA and CODE in executable section is the source of problems
  - Confuse between DATA and CODE
    - Assume DATA as CODE, DATA may be corrupted
    - Assume CODE as DATA, the execution may fail

Challenge for SBI

- The mix of DATA and CODE in executable section is the source of problems
  - Confuse between DATA and CODE
    - Public symbol can solve

Challenge for SBI

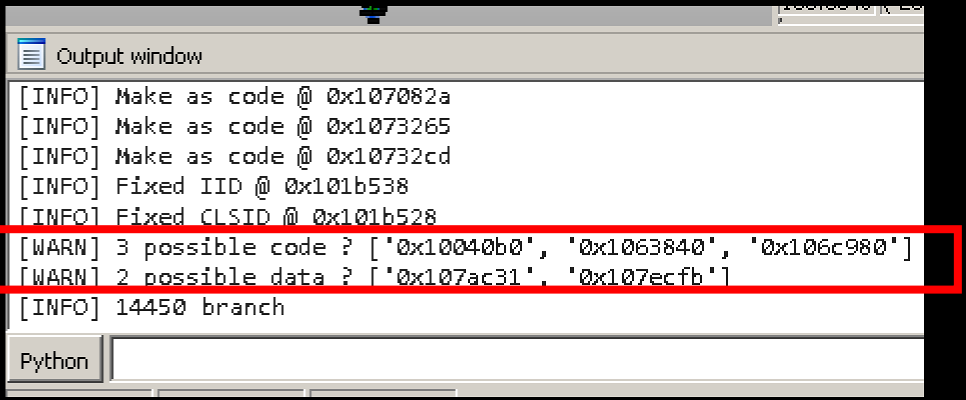
- The mix of DATA and CODE in executable section is the source of problems
  - Confuse between DATA and CODE
    - Public symbol can solve, otherwise …
    - IDA pro is improving

Challenge for SBI

- The mix of DATA and CODE in executable section is the source of problems
  - Confuse between DATA and CODE
    - Public symbol can solve, otherwise …
    - IDA pro is improving, otherwise …
    - Assume DATA as CODE, DATA may be corrupted
      - Instrument before branch instead of basic block
      - Validate the branch, otherwise alert it
    - Assume CODE as DATA, the execution may fail
      - Look for valid branch in suspicious data
      - Filter with known data type and alert it

Challenge for SBI

- The mix of DATA and CODE in executable section is the source of problems
  - Confuse between DATA and CODE



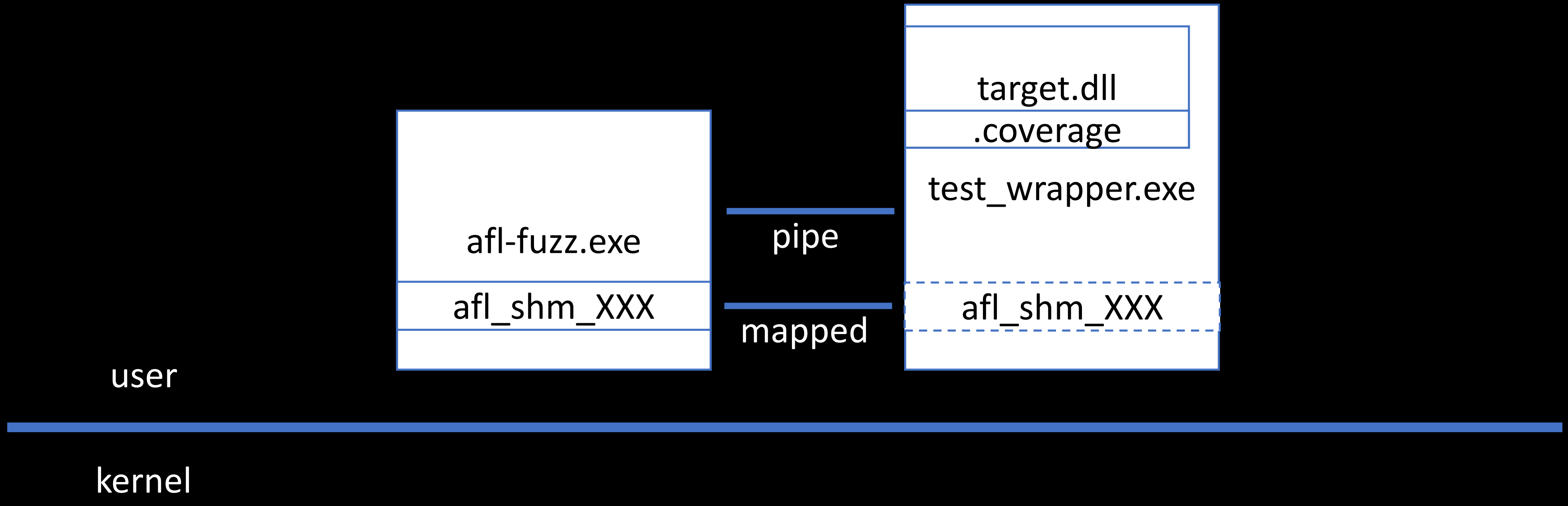
```
[INFO] Make as code @ 0x107082a
[INFO] Make as code @ 0x1073265
[INFO] Make as code @ 0x10732cd
[INFO] Fixed IID @ 0x101b538
[INFO] Fixed CLSID @ 0x101b528
[WARN] 3 possible code ? ['0x10040b0', '0x1063840', '0x106c980']
[WARN] 2 possible data ? ['0x107ac31', '0x107ecfb']
[INFO] 14450 branch
```

Instrumenting mspaint.exe without PDB

  - Workaround

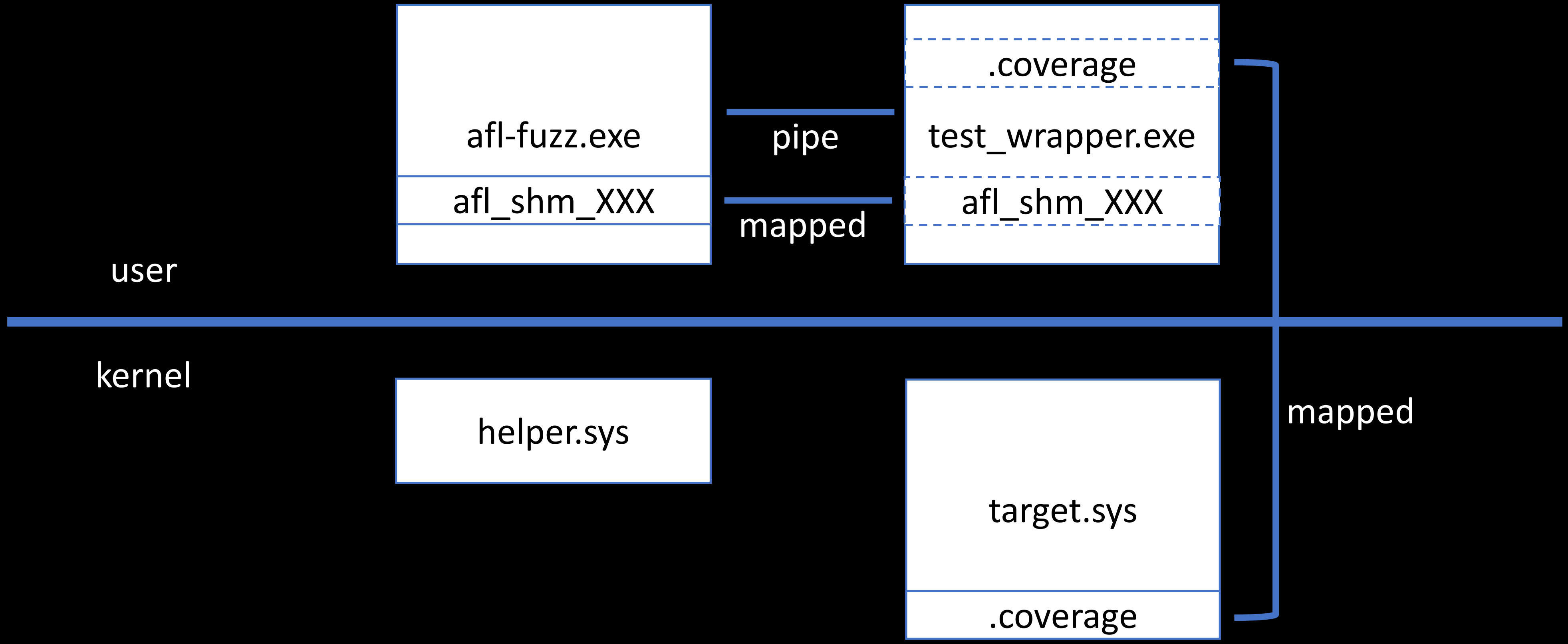
Implementation – pe-afl

- Fuzz on user-mode



Implementation – pe-afl

- Fuzz on kernel-mode



Implementation – pe-afl

- Type of instrument on fuzzing
    - PID filtering
    - multi-thread

      different afl_prev_loc for each thread
    - inline-mode in assembly vs. callback-mode in C

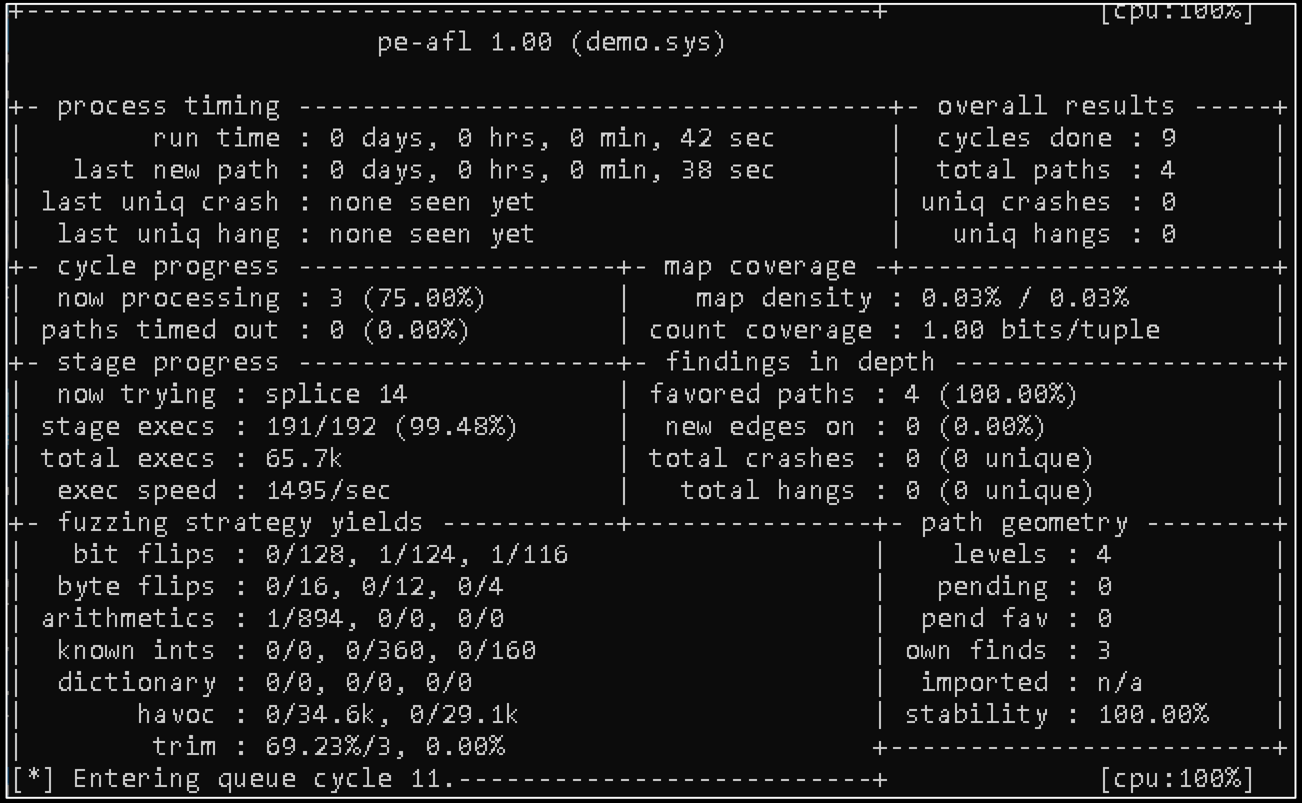
Benchmark

- Test on gdiplus.dll
- Win10, 1 vm, 4GB ram, i7-7600, 1 core

pe-afl (w/o instrument)	522 exec/s
pe-afl	508 exec/s
winafl (edge mode)	236 exec/s

- WINAFL states that *“This approach has been found to introduce an overhead about 2x compared to the native execution speed”*

Demo



```
+------------------------------------------------------+          [cpu:100%]
                      pe-afl 1.00 (demo.sys)

+- process timing -------------------------------------+- overall results -----+
|        run time : 0 days, 0 hrs, 0 min, 42 sec      |  cycles done : 9      |
|   last new path : 0 days, 0 hrs, 0 min, 38 sec      |  total paths : 4      |
| last uniq crash : none seen yet                      | uniq crashes : 0      |
|  last uniq hang : none seen yet                      |   uniq hangs : 0      |
+- cycle progress --------------------+- map coverage -+-----------------------+
|  now processing : 3 (75.00%)        |    map density : 0.03% / 0.03%         |
| paths timed out : 0 (0.00%)         | count coverage : 1.00 bits/tuple       |
+- stage progress --------------------+- findings in depth --------------------+
|  now trying : splice 14             | favored paths : 4 (100.00%)            |
| stage execs : 191/192 (99.48%)      |  new edges on : 0 (0.00%)              |
| total execs : 65.7k                 | total crashes : 0 (0 unique)           |
|  exec speed : 1495/sec              |   total hangs : 0 (0 unique)           |
+- fuzzing strategy yields -----------+----------------+- path geometry --------+
|   bit flips : 0/128, 1/124, 1/116                    |    levels : 4          |
|  byte flips : 0/16, 0/12, 0/4                        |   pending : 0          |
| arithmetics : 1/894, 0/0, 0/0                        |  pend fav : 0          |
|  known ints : 0/0, 0/360, 0/160                      | own finds : 3          |
|  dictionary : 0/0, 0/0, 0/0                          |  imported : n/a        |
|       havoc : 0/34.6k, 0/29.1k                       | stability : 100.00%    |
|        trim : 69.23%/3, 0.00%                        +------------------------+
[*] Entering queue cycle 11.---------------------------+          [cpu:100%]
```

Case study (1)

- CLFS
  - First try on kernel driver
  - Well-known attack vector
    - Btw, it was sandboxed
  - Parsing un-document BLF binary format in kernel
  - Entry point

    CreateTransactionManager(“input.blf”)
  - Patch checksum
  - 2 weeks, 8 vms
  - 2 CVE + won’t fix case
    - CVE-2018-0844, pool overflow
    - CVE-2018-0846, UAF

Case study (2)

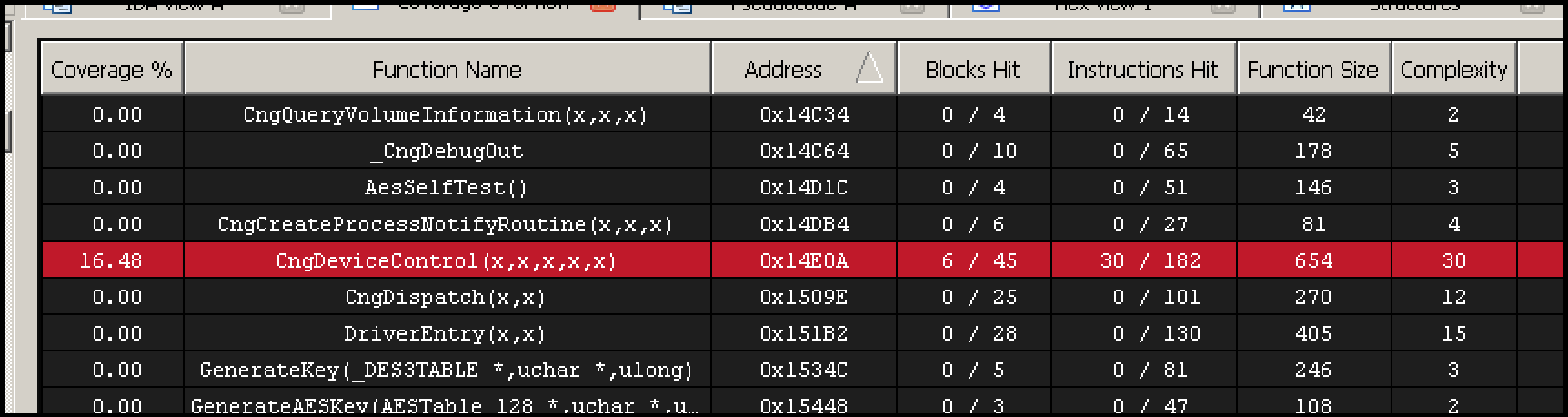
- CNG
  - Entry point

    IOCTL
  - Applicable on any kind of IOCTL fuzzing
  - Coverage is stuck at the beginning
    - Try to figure out the root cause

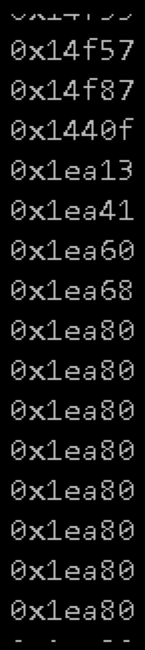
Case study (2)

- CNG
  - Entry point

    IOCTL
  - Applicable on any kind of IOCTL fuzzing
  - Coverage is stuck at the beginning
  - Benefit from SBI, it is easy to dump execution trace



Coverage %	Function Name	Address	Blocks Hit	Instructions Hit	Function Size	Complexity
0.00	CngQueryVolumeInformation(x,x,x)	0x14C34	0 / 4	0 / 14	42	2
0.00	_CngDebugOut	0x14C64	0 / 10	0 / 65	178	5
0.00	AesSelfTest()	0x14D1C	0 / 4	0 / 51	146	3
0.00	CngCreateProcessNotifyRoutine(x,x,x)	0x14DB4	0 / 6	0 / 27	81	4
16.48	CngDeviceControl(x,x,x,x,x)	0x14E0A	6 / 45	30 / 182	654	30
0.00	CngDispatch(x,x)	0x1509E	0 / 25	0 / 101	270	12
0.00	DriverEntry(x,x)	0x151B2	0 / 28	0 / 130	405	15
0.00	GenerateKey(_DES3TABLE *,uchar *,ulong)	0x1534C	0 / 5	0 / 81	246	3
0.00	GenerateAESKey(AESTable_128 *,uchar *,u…	0x15448	0 / 3	0 / 47	108	2



```
0x14f57
0x14f87
0x1440f
0x1ea13
0x1ea41
0x1ea60
0x1ea68
0x1ea80
0x1ea80
0x1ea80
0x1ea80
0x1ea80
0x1ea80
0x1ea80
0x1ea80
```

Import into lighthouse

Case study (2)

- CNG
  - Entry point

    IOCTL
  - Applicable on any kind of IOCTL fuzzing
  - Coverage is stuck at the beginning
  - Benefit from SBI, it is easy to dump execution trace
  - It needs valid object

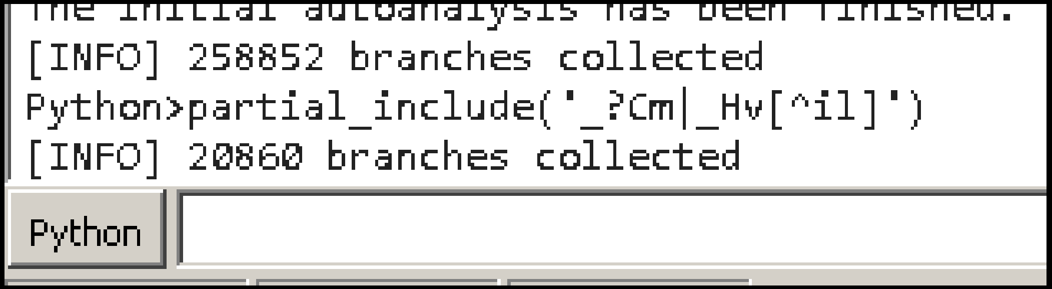
    eg. CreateEvent()
  - It needs magic header

    eg. 0x1a2b3c4d
  - 1 week, 8 vms
  - 1 CVE
    - CVE-2018-8207, pool OOB read

Case study (3)

- Registry Hive
  - Parsing un-document registry hive format in ntoskrnl.exe
  - Entry point

    RegLoadAppKey(“input.dat”)
  - Have to instrument around 7MB ntoskrnl.exe
  - Support and use partial instrument here



```
The initial autoanalysis has been finished.
[INFO] 258852 branches collected
Python>partial_include('_?Cm|_Hv[^il]')
[INFO] 20860 branches collected
```

Case study (3)

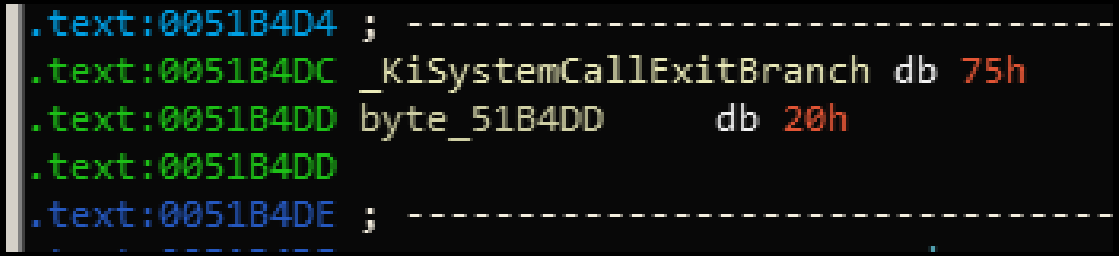
- Registry Hive
  - Parsing un-document registry hive format in ntoskrnl.exe
  - Entry point

    RegLoadAppKey(“input.dat”)
  - Have to instrument around 5MB ntoskrnl.exe
  - Support and use partial instrument here

    RE = ’_?Cm|_Hv[^il]’
  - No CVE
    - Global state in registry brings the non-deterministic on fuzzing

Case study (3) – post story

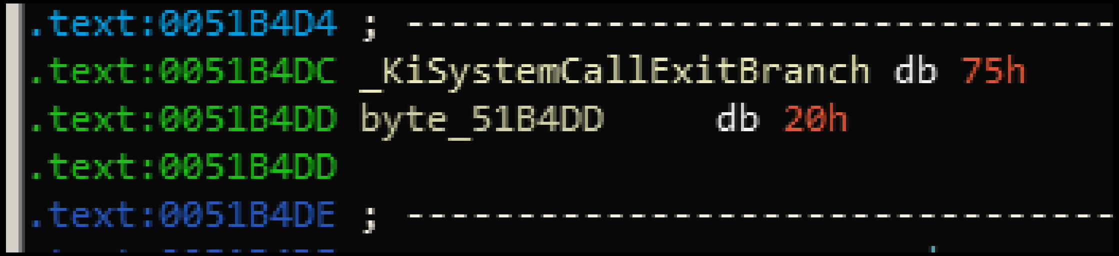
- Full instrumentation on ntoskrnl.exe
- Everything works except one
  - Self-modifying branch ☹



```
.text:0051B4D4 ; ---------------------------------------
.text:0051B4DC _KiSystemCallExitBranch db 75h
.text:0051B4DD byte_51B4DD     db 20h
.text:0051B4DD
.text:0051B4DE ; ---------------------------------------
```

Case study (3) – post story

- Full instrumentation on ntoskrnl.exe
- Everything works except one
  - Self-modifying branch ☹
- Detectable
- Skip with partial instrumentation
- Workaround



```
.text:0051B4D4 ; ---------------------------------------
.text:0051B4DC _KiSystemCallExitBranch db 75h
.text:0051B4DD byte_51B4DD     db 20h
.text:0051B4DD
.text:0051B4DE ; ---------------------------------------
```

Conclusion

- Show the possibility and limitation of SBI on PE file and fuzzing
- Not so reliable and elegant, but it works and high performance
- Benefit from SBI
  - Not only feedback code coverage, but also data, stack depth …
  - Not only for fuzzing, but also for bug detection, tracing …
- Open source
  - https://github.com/wmliang/pe-afl

Thanks

- Thanks
  - AFL, WINAFL
  - Lays, Steward Fu, Serena Lin
  - Bluehat IL conference team

- Contact
  - https://twitter.com/_wmliang_
  - lucas_leong@trendmicro.com