TALOS

PROTECTING YOUR NETWORK

# Evolutionary Kernel Fuzzing

Black Hat USA 2017

Richard Johnson

# Evolutionary Kernel Fuzzing

Richard Johnson | Black Hat USA 2017

# whoami

- Richard Johnson
  Research Lead
  @richinseattle
  rjohnson@moflow.org

- Cisco Talos Vulndev
  - Third party vulnerability research
    - Microsoft
    - Apple
    - Oracle
    - Adobe
    - Google
    - IBM, HP, Intel
  - Security tool development
    - Fuzzers, Crash Triage
  - Mitigation development

- Special Contributor
  - Andrea Allievi, Microsoft

TALOS

# Introduction

- High performance tracing and fuzzing since 2014
  - 2014 – High Performance Fuzzing
    - Input selection
    - Engine design
    - AFL-DYNINST
    - Windows fork()
  - 2015 – Go Speed Tracer
    - Guided Fuzzing
    - Binary translation
    - Hardware tracing

# Introduction

- High performance tracing and fuzzing since 2014
  - 2016 – Harnessing Intel Processor Trace for Vulnerability Discovery
    - Intel Processor Trace internals
    - Usermode fuzzing with Intel Processor Trace
    - Persistent mode fuzzing native Windows binaries

In June 2016 we opensourced Windows driver for Intel Processor Trace

- https://github.com/intelpt

# Introduction

Today we will bring this knowledge to the context of fuzzing the Windows kernel

See http://moflow.org for previous slides and talk videos

# Introduction

- Agenda
  - Evolutionary Fuzzing
  - Kernel Code Coverage
  - Linux Kernel Fuzzing
  - Windows Kernel Fuzzing
- Goals
  - Understand the benefits of guided fuzzing
  - Understand coverage collection techniques for kernels
  - Identify critical Windows Kernel attack surface
  - Learn how to apply state of the art fuzzing to kernels

# Introduction

- Kernels are a critical attack surface
- Modern mitigations utilize isolation and sandboxing
- Weaponized exploits include kernel attacks
  - Pwn2own
  - Leaked government warez
- Kernel vulndev is still in its infancy
  - Room for improvment on fuzzing tech

# Introduction

- Application Sandboxing
  - IE sandbox
  - IE Protected Mode
  - Chrome sandbox
  - Adobe Reader sandbox
  - etc
- Windows Isolation / Sandboxing
  - Driver Signature Verification
  - Patchguard / Kernel Patch Protection
  - AppContainers
  - ProcessMitigationPolicy
  - etc

# Introduction

- Prior Windows Kernel vulndev by the following people
  - Ilja van Sprundel
  - Mateusz Jurczyk / @j00ru
  - Jesse Hertz / @killahertz_
  - Tim Newsham / @newshtwit
  - Nils / @nils
  - Georgi Geshev / @munmap
  - James Loureio / @NerdKernel
  - Peter Hlavaty / @zer0mem
  - Daniel King / @long123king
  - Marco Grassi / @marcograss
  - Nikita Tarakanov / @NTarakanov

# Evolutionary Fuzzing

# Evolutionary Fuzzing

- History
  - 2006: Sidewinder – Sparks & Cunningham
  - 2007: Evolutionary Fuzzing System – Jared Demott
  - 2007: Bunny the Fuzzer – Michal Zalewski
  - 2013: American Fuzzy Lop – Michal Zalewski
  - 2014: Nightmare/BCCF – Joxean Koret
  - 2015: Honggfuzz – Robert Swiecki
  - 2015: covFuzz – Atte Kettunen
  - 2016 : Choronzon – Zisis Sialveras / Nikos Naziridis

# Evolutionary Fuzzing

- Incrementally better mutational dumb fuzzing
- Trace while fuzzing and provide feedback signal
- Evolutionary algorithms
  - Assess fitness of current input
  - Manage a pool of possible inputs

# Evolutionary Fuzzing

- Required
  - Fast tracing engine
    - Block granularity code coverage
  - Fast logging
    - Memory resident coverage map
    - **Not a list of every basic block
  - Fast evolutionary algorithm
    - Minimum of global population map
    - Maximum pool diversity

# Evolutionary Fuzzing

- Desired
  - Portable
  - Easy to use
  - Helper tools
  - Grammar detection

# Evolutionary Fuzzing

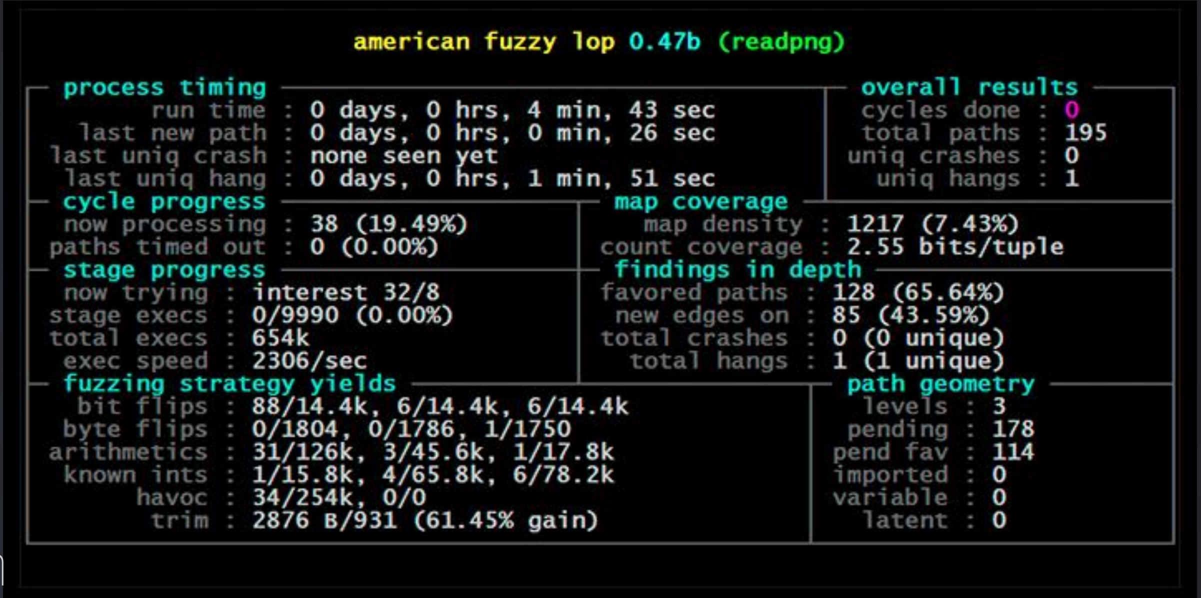
AFL delivers the most complete package
Lets review!

# Amercian Fuzzy Lop

- Michal Zalewski 2013
  - Delivered the first performant opensource evolutionary fuzzer
- Features
  - Variety of traditional mutation strategies
  - Block coverage via compile time instrumentation
  - Simplified approach to genetic algorithm
    - Edge transitions are encoded as tuple and tracked in a bloom filter
    - Includes coverage and frequency
  - Uses portable* Posix API for shared memory, process creation

# Amercian Fuzzy Lop

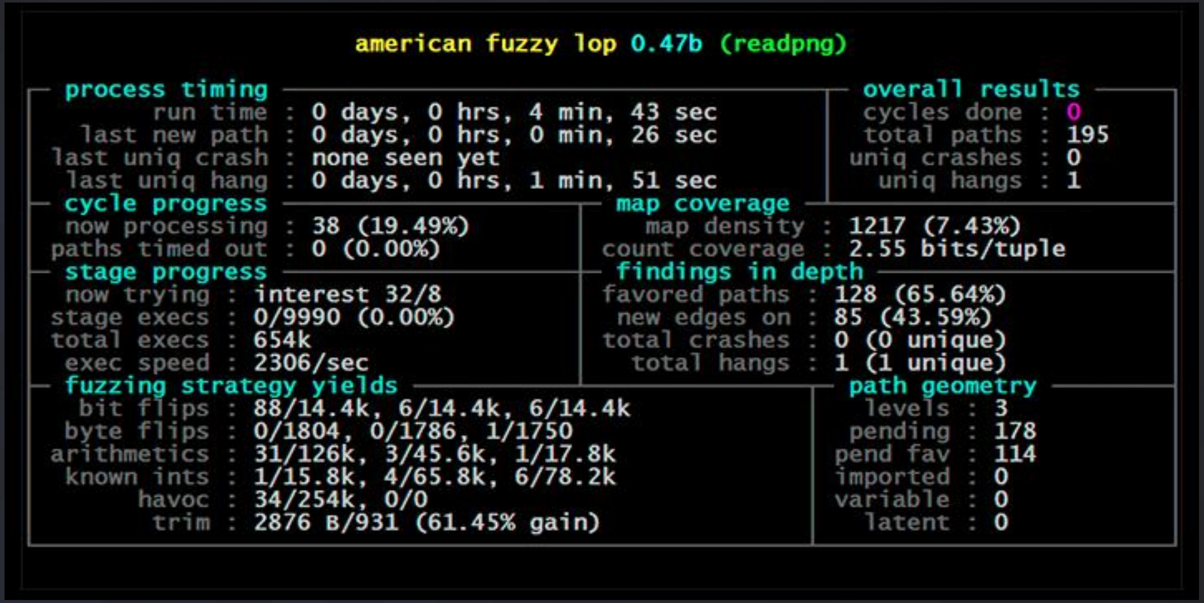
- Contributions
  - Tracks edge transitions
    - Not just block entry
  - Global coverage map
    - Generation tracking
  - Fork server
    - Reduce target initialization
  - Persistent fuzzing
  - Builds corpus of unique inputs reusable in other workflows



```
                     american fuzzy lop 0.47b (readpng)

┌─ process timing ─────────────────────────────────────┬─ overall results ─────┐
│        run time : 0 days, 0 hrs, 4 min, 43 sec        │  cycles done : 0      │
│   last new path : 0 days, 0 hrs, 0 min, 26 sec        │  total paths : 195    │
│ last uniq crash : none seen yet                       │ uniq crashes : 0      │
│  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec        │   uniq hangs : 1      │
├─ cycle progress ─────────────────┬─ map coverage ─────┴───────────────────────┤
│  now processing : 38 (19.49%)    │    map density : 1217 (7.43%)              │
│ paths timed out : 0 (0.00%)      │ count coverage : 2.55 bits/tuple           │
├─ stage progress ─────────────────┼─ findings in depth ────────────────────────┤
│  now trying : interest 32/8      │ favored paths : 128 (65.64%)               │
│ stage execs : 0/9990 (0.00%)     │  new edges on : 85 (43.59%)                │
│ total execs : 654k               │ total crashes : 0 (0 unique)               │
│  exec speed : 2306/sec           │   total hangs : 1 (1 unique)               │
├─ fuzzing strategy yields ────────┴──────────────┬─ path geometry ─────────────┤
│   bit flips : 88/14.4k, 6/14.4k, 6/14.4k        │    levels : 3               │
│  byte flips : 0/1804, 0/1786, 1/1750            │   pending : 178             │
│ arithmetics : 31/126k, 3/45.6k, 1/17.8k         │  pend fav : 114             │
│  known ints : 1/15.8k, 4/65.8k, 6/78.2k         │  imported : 0               │
│       havoc : 34/254k, 0/0                      │  variable : 0               │
│        trim : 2876 B/931 (61.45% gain)          │    latent : 0               │
└─────────────────────────────────────────────────┴─────────────────────────────┘
```

# Amercian Fuzzy Lop

- Trace Logging
  - Each block gets a unique ID
  - Traversed edges are indexed into a byte map (bloom filter)
  - Create a hash from the src and dst block IDs
  - Increment map for each time an edge is traversed
  - Each trace is easily comparable to the entire session history



```
                    american fuzzy lop 0.47b (readpng)

┌─ process timing ─────────────────────────────────────┬─ overall results ─┐
│        run time : 0 days, 0 hrs, 4 min, 43 sec       │  cycles done : 0  │
│   last new path : 0 days, 0 hrs, 0 min, 26 sec       │  total paths : 195│
│ last uniq crash : none seen yet                      │ uniq crashes : 0  │
│  last uniq hang : 0 days, 0 hrs, 1 min, 51 sec       │   uniq hangs : 1  │
├─ cycle progress ────────────────┬─ map coverage ─────┴───────────────────┤
│  now processing : 38 (19.49%)   │    map density : 1217 (7.43%)          │
│ paths timed out : 0 (0.00%)     │ count coverage : 2.55 bits/tuple       │
├─ stage progress ────────────────┼─ findings in depth ────────────────────┤
│  now trying : interest 32/8     │ favored paths : 128 (65.64%)           │
│ stage execs : 0/9990 (0.00%)    │  new edges on : 85 (43.59%)            │
│ total execs : 654k              │ total crashes : 0 (0 unique)           │
│  exec speed : 2306/sec          │   total hangs : 1 (1 unique)           │
├─ fuzzing strategy yields ───────┴─────────────────┬─ path geometry ──────┤
│   bit flips : 88/14.4k, 6/14.4k, 6/14.4k          │    levels : 3        │
│  byte flips : 0/1804, 0/1786, 1/1750              │   pending : 178      │
│ arithmetics : 31/126k, 3/45.6k, 1/17.8k           │  pend fav : 114      │
│  known ints : 1/15.8k, 4/65.8k, 6/78.2k           │  imported : 0        │
│       havoc : 34/254k, 0/0                        │  variable : 0        │
│        trim : 2876 B/931 (61.45% gain)            │    latent : 0        │
└───────────────────────────────────────────────────┴──────────────────────┘
```

Okay, so lets take a fuzzer that targets userland programs with source code and make it work for closed source Windows kernel targets!

But first how about Windows binaries...

# WinAFL

- Ivan Fratric - 2016
  - First performant windows evolutionary fuzzer
- Features
  - Its American Fuzzy Lop! For Windows!
  - Windows API port for memory and process creation
  - DynamoRIO based code coverage
  - Filter based on module
  - Block and Edge tracing modes
  - Persistent execution mode

# WinAFL-IntelPT

- Richard Johnson - 2016
  - First hardware assisted guided fuzzer for Windows
  - First public guided fuzzer for Windows kernel
- Features
  - Intel Processor Trace based coverage engine
  - Online disassembly engine to decode Intel PT trace
  - Filter based on module
  - Edge tracing mode
  - Persistent execution mode
  - Kernel tracing mode

# Kernel Code Coverage

# Kernel Code Coverage

- Kernel code coverage can be elusive to obtain
- Opensource code can be instrumented by compilers
- Binary code must use runtime instrumentation, static rewriting, or hardware engines

# Kernel Code Coverage

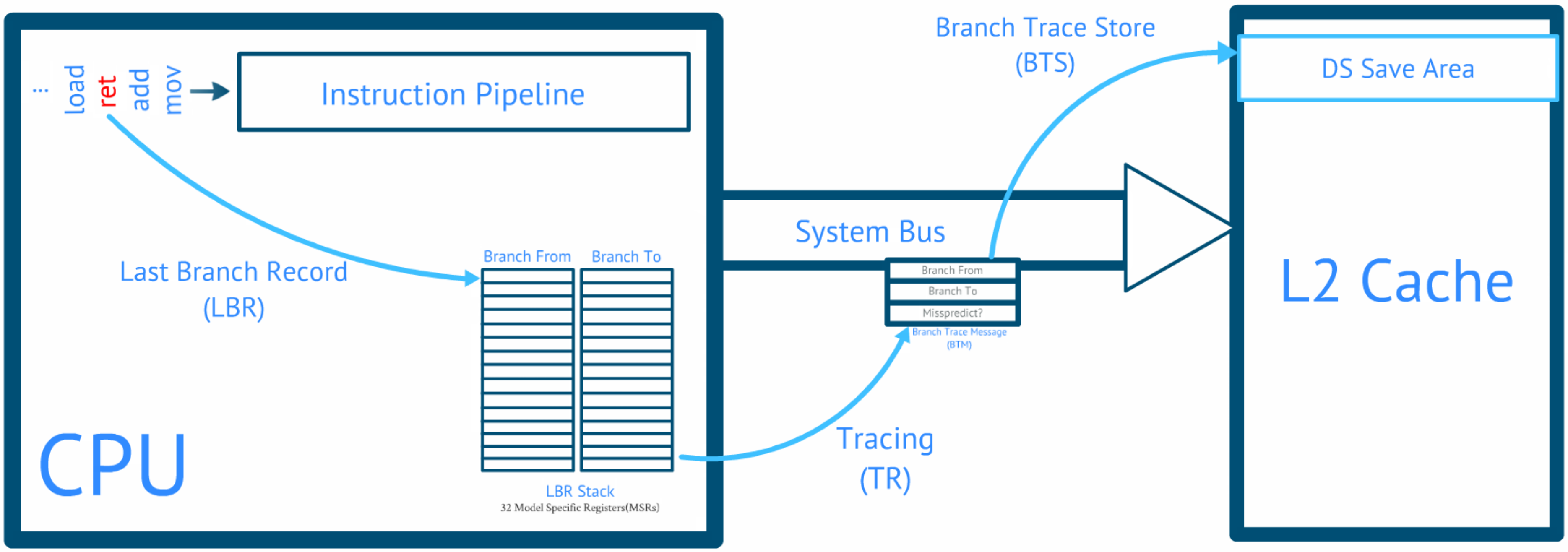
- Existing tools and approaches
  - Source
    - GCC
      - gcc --coverage
      - AFL adds hooks into the .S intermediate files
    - Clang
      - clang -fprofile-instr-generate -fcoverage-mapping
      - afl-clang-fast uses a compiler pass

# Kernel Code Coverage

- Existing tools and approaches
  - Binary
    - QEMU
      - Hook Tiny Code Generator (TCG)
        - translates IR to native ISA
    - BOCHS
      - Seems to work for j00ru ☺
    - syzygy
      - Statically rewrite PE32 binaries with AFL
      - Requires symbols ☹
      - Requires additional dev to make WinAFL kernel aware

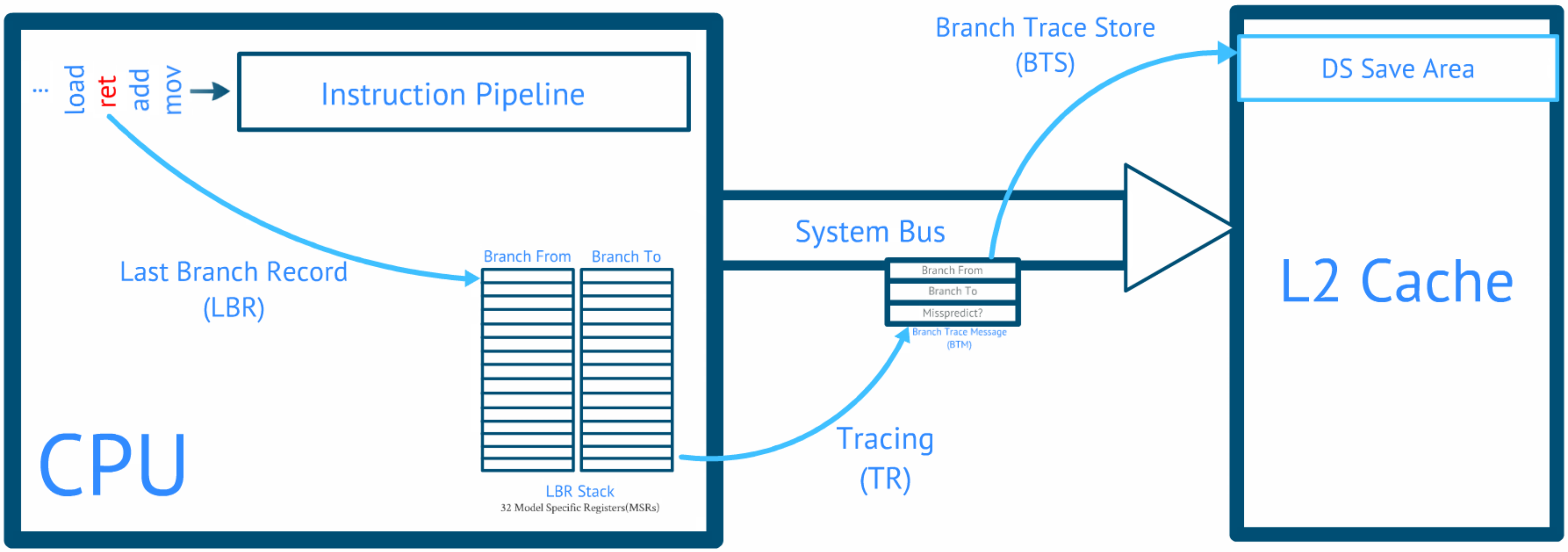
# Kernel Code Coverage

- Intel / AMD CPUs – Branch Trace Store
  - Per-kernel-thread hardware trace
  - Use in combination with Last Branch Record to get edge transition
  - Supported passthrough by some hypervisors



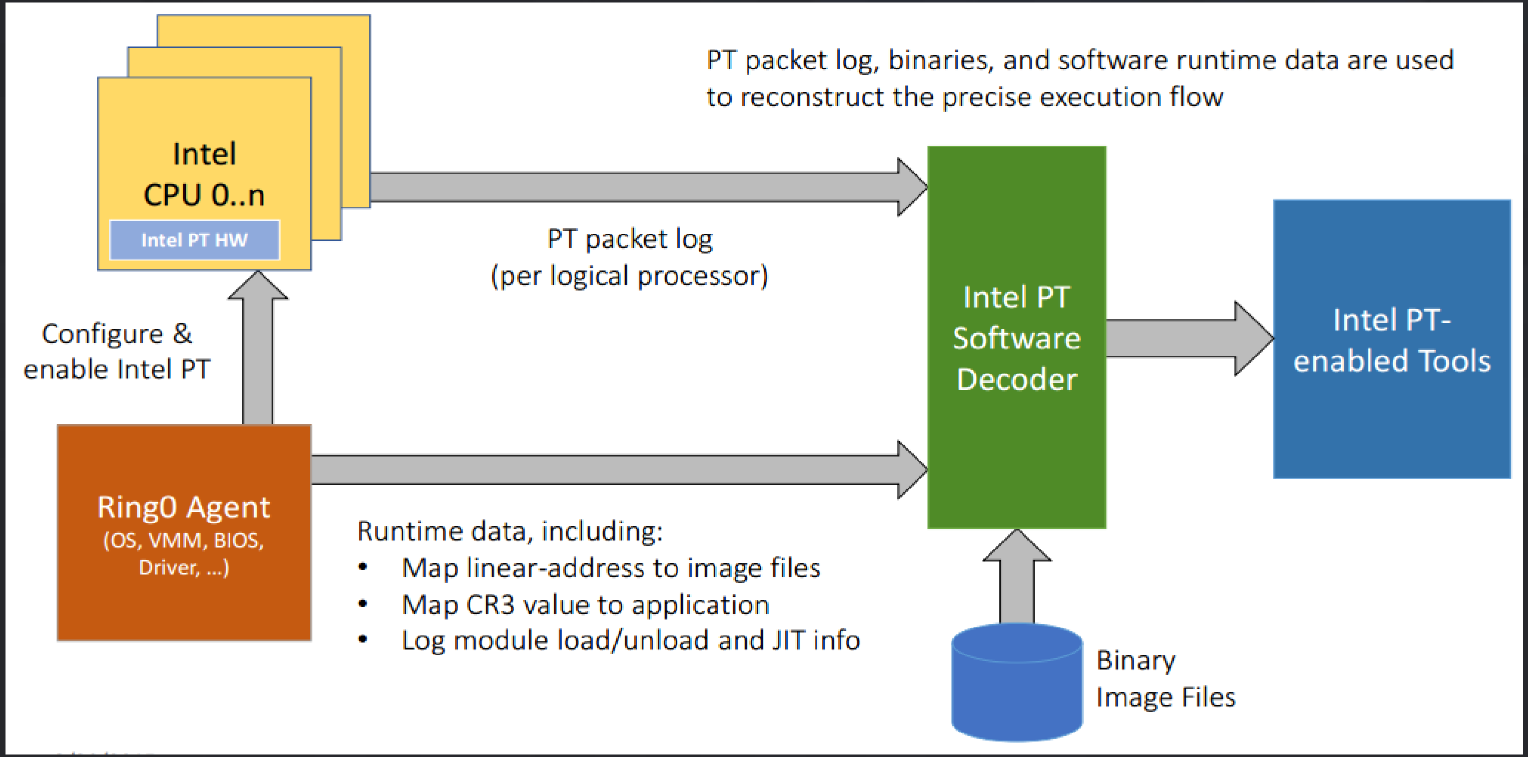
# Kernel Code Coverage

- Intel / AMD CPUs – Branch Trace Store
  - New opensource software recently released for Windows BTS
    - https://github.com/marcusbotacin/BranchMonitoringProject



# Kernel Code Coverage

- Intel CPUs – Intel Processor Trace
  - Introduced in Broadwell / Skylake


Intel
CPU 0..n
Intel PT HW
PT packet log, binaries, and software runtime data are used to reconstruct the precise execution flow
PT packet log
(per logical processor)
Configure & enable Intel PT
Intel PT Software Decoder
Intel PT-enabled Tools
Ring0 Agent
(OS, VMM, BIOS, Driver, ...)
Runtime data, including:
• Map linear-address to image files
• Map CR3 value to application
• Log module load/unload and JIT info
Binary
Image Files

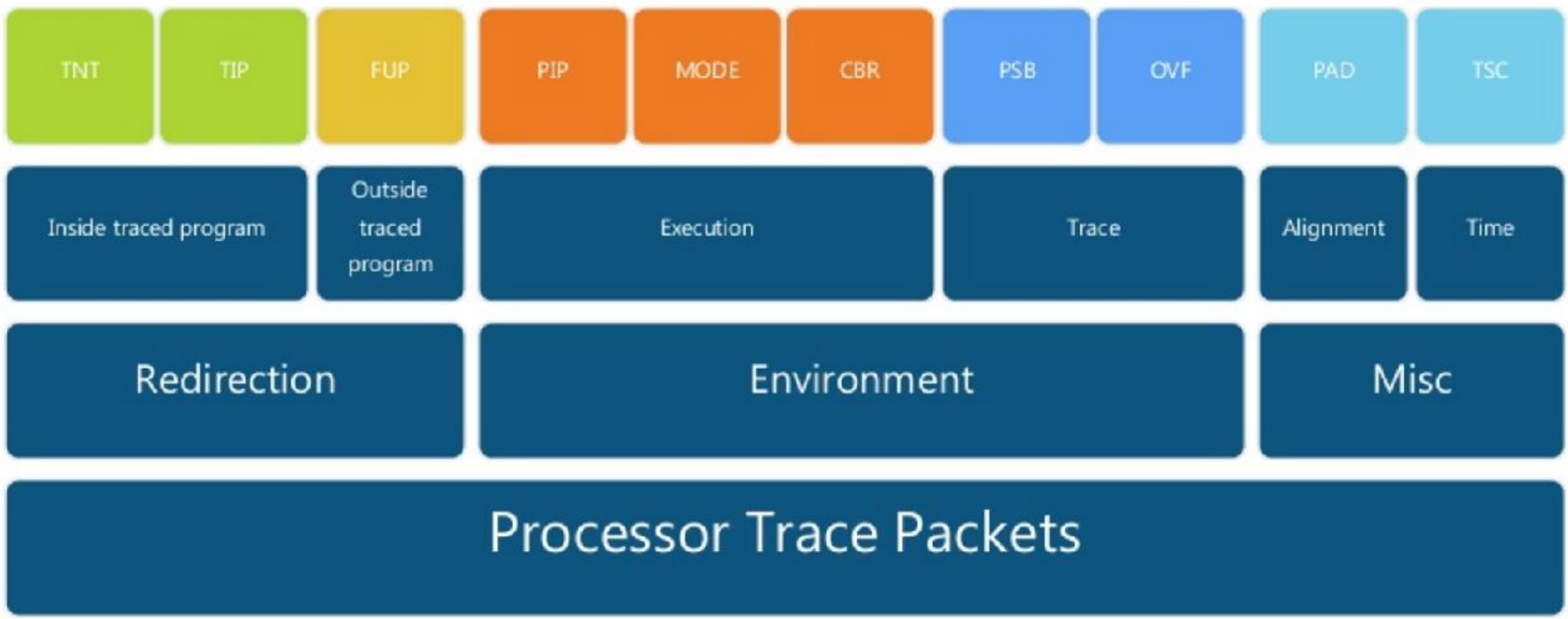
# Kernel Code Coverage

- Intel CPUs – Intel Processor Trace
  - Performance
    - Low over-head (15% CPU perf hit for recording)
    - Logs directly to physical memory
      - Bypass TLB and eliminating cache pollution
    - Minimal log format
      - One bit per conditional branch
      - Only indirect branches log dest address
    - Additional overhead to decode trace, requires disassembly

  - See “Harnessing Intel Processor Trace for Vuln Discovery” for deep dive

# Kernel Code Coverage

- Intel CPUs – Intel Processor Trace
  - Sparse binary packet format



TNT TIP FUP PIP MODE CBR PSB OVF PAD TSC

Inside traced program | Outside traced program | Execution | Trace | Alignment | Time

Redirection | Environment | Misc

Processor Trace Packets

Complex format - decode with Intel's opensource libipt library!

# Kernel Code Coverage

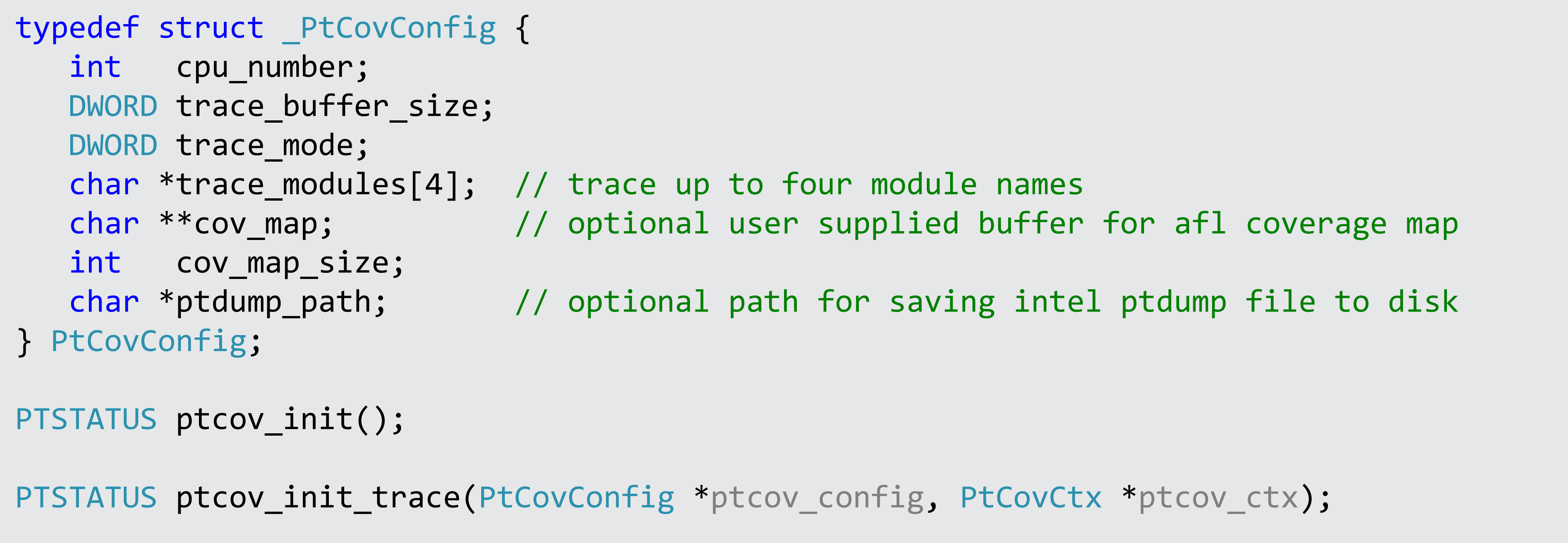
- We have contributed two opensource projects to harness Intel Processor Trace!
  - Get the code! https://github.com/intelpt

- WindowsPtDriver
  - Implements Intel Processor Trace support for Windows

- PtCov Intel Processor Trace Library
  - Userland API for interacting with the kernel mode driver
  - Easily turn any existing file fuzzer into coverage driven fuzzer

# Kernel Code Coverage

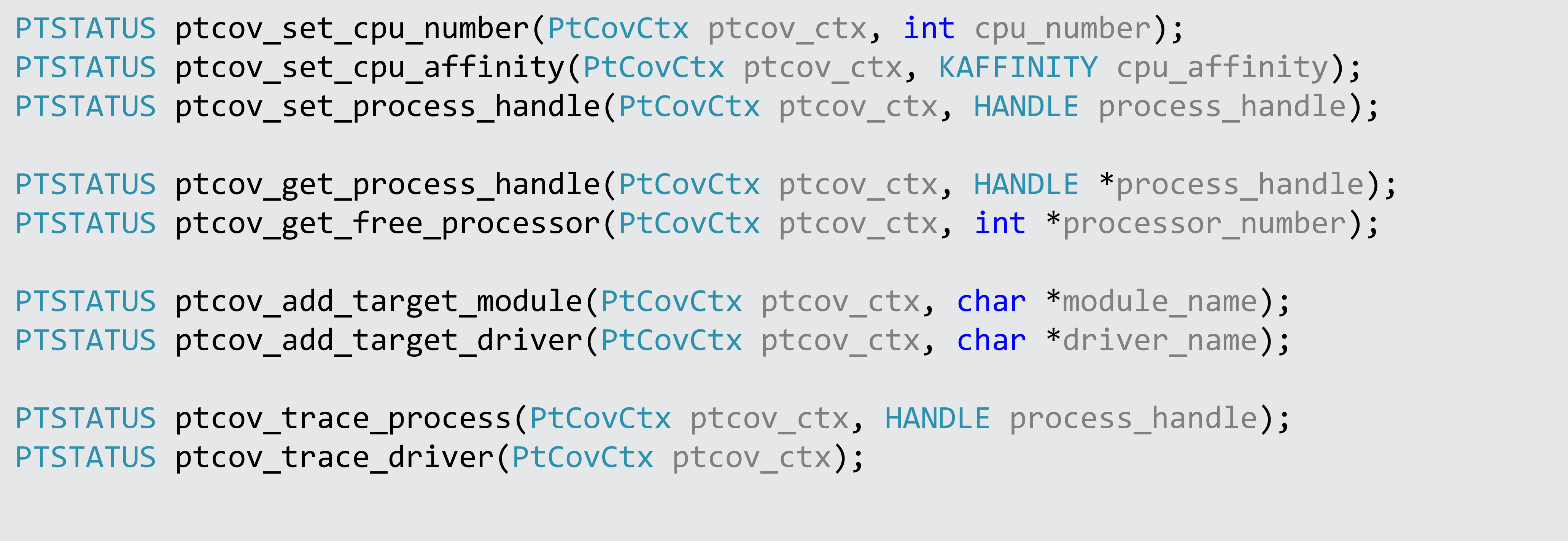
- PtCov Intel Processor Trace Library



```
typedef struct _PtCovConfig {
   int   cpu_number;
   DWORD trace_buffer_size;
   DWORD trace_mode;
   char *trace_modules[4];  // trace up to four module names
   char **cov_map;          // optional user supplied buffer for afl coverage map
   int   cov_map_size;
   char *ptdump_path;       // optional path for saving intel ptdump file to disk
} PtCovConfig;

PTSTATUS ptcov_init();

PTSTATUS ptcov_init_trace(PtCovConfig *ptcov_config, PtCovCtx *ptcov_ctx);
```

# Kernel Code Coverage

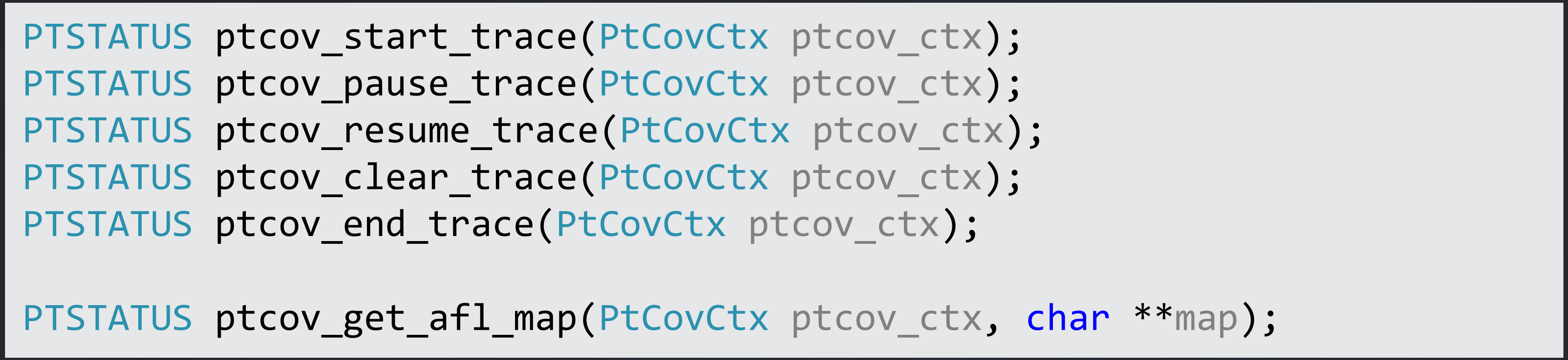
- PtCov Intel Processor Trace Library



```
PTSTATUS ptcov_set_cpu_number(PtCovCtx ptcov_ctx, int cpu_number);
PTSTATUS ptcov_set_cpu_affinity(PtCovCtx ptcov_ctx, KAFFINITY cpu_affinity);
PTSTATUS ptcov_set_process_handle(PtCovCtx ptcov_ctx, HANDLE process_handle);

PTSTATUS ptcov_get_process_handle(PtCovCtx ptcov_ctx, HANDLE *process_handle);
PTSTATUS ptcov_get_free_processor(PtCovCtx ptcov_ctx, int *processor_number);

PTSTATUS ptcov_add_target_module(PtCovCtx ptcov_ctx, char *module_name);
PTSTATUS ptcov_add_target_driver(PtCovCtx ptcov_ctx, char *driver_name);

PTSTATUS ptcov_trace_process(PtCovCtx ptcov_ctx, HANDLE process_handle);
PTSTATUS ptcov_trace_driver(PtCovCtx ptcov_ctx);
```

# Kernel Code Coverage

- PtCov Intel Processor Trace Library



```
PTSTATUS ptcov_start_trace(PtCovCtx ptcov_ctx);
PTSTATUS ptcov_pause_trace(PtCovCtx ptcov_ctx);
PTSTATUS ptcov_resume_trace(PtCovCtx ptcov_ctx);
PTSTATUS ptcov_clear_trace(PtCovCtx ptcov_ctx);
PTSTATUS ptcov_end_trace(PtCovCtx ptcov_ctx);

PTSTATUS ptcov_get_afl_map(PtCovCtx ptcov_ctx, char **map);
```

# Kernel Code Coverage

- Other methods
  - Single step / branch step (BTF)
    - Int 0x1 enabled on each instruction to singlestep
    - DbgCtrl MSR flag to interrupt only on branch
  - PMU Sampling
    - Can be forced to interrupt on each branch
    - Asynchronous but slow
    - Works everywhere (including ARM)
  - Dynamic binary translation
    - Attempts with PIN for drivers, not public

# Demo

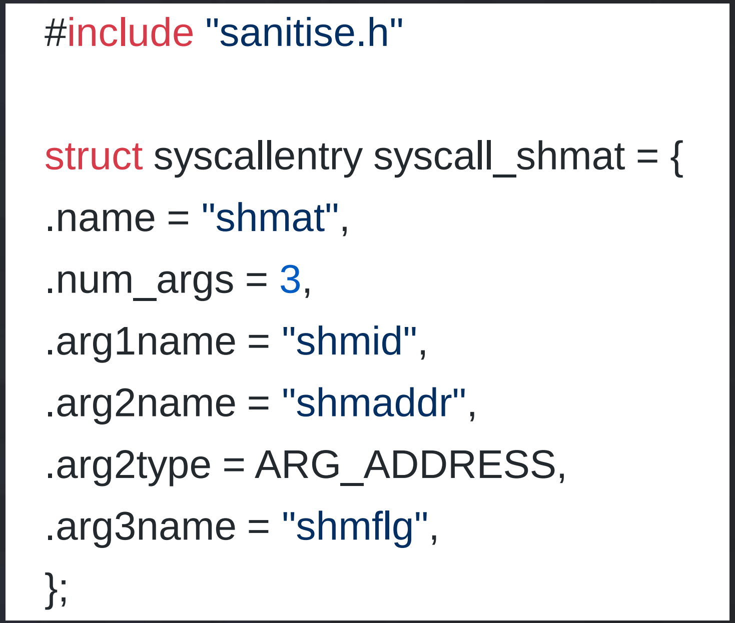
## Windows Kernel Code Coverage

# Linux Kernel Fuzzing

# Linux Kernel Fuzzing

- Trinity
  https://github.com/kernelslacker/trinity

  - Built into the Linux kernel tree
  - Type aware via templates
  - Not coverage driven

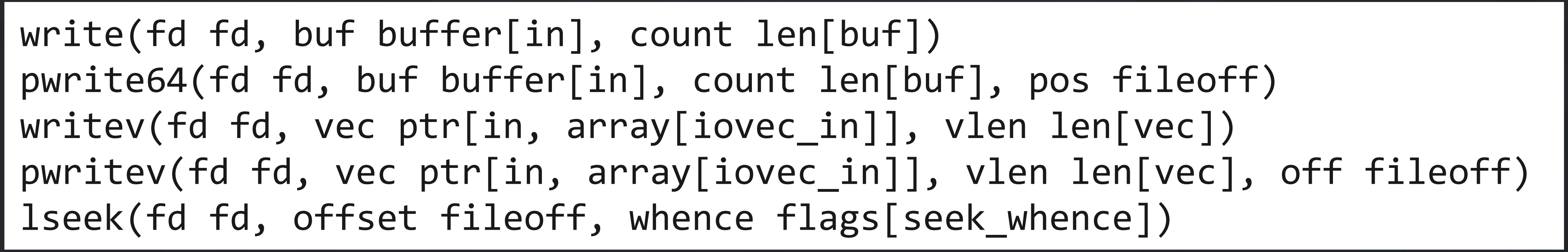


```
#include "sanitise.h"

struct syscallentry syscall_shmat = {
.name = "shmat",
.num_args = 3,
.arg1name = "shmid",
.arg2name = "shmaddr",
.arg2type = ARG_ADDRESS,
.arg3name = "shmflg",
};
```

- "Jones has considered feedback-guided fuzzing for Trinity in the past, but found the coverage tools that were available at the time to be too slow."

# Linux Kernel Fuzzing

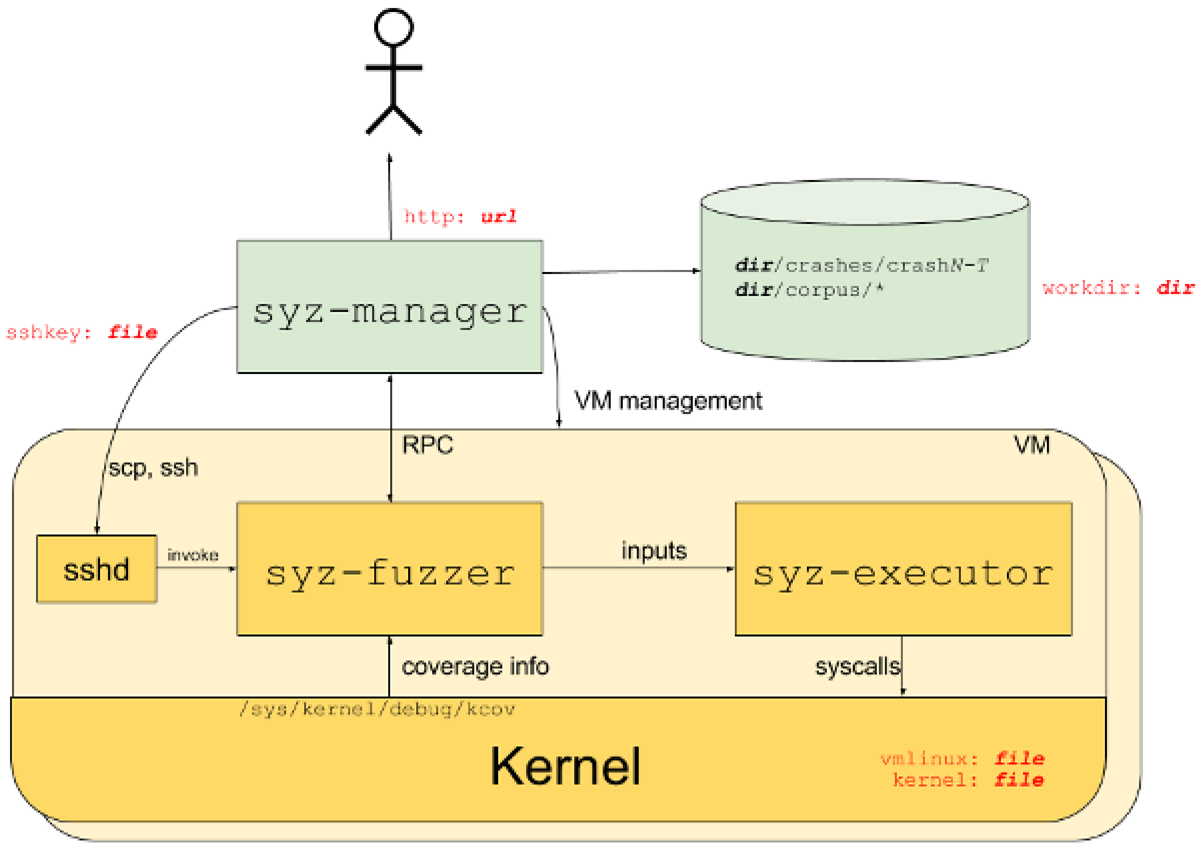
- Syzkaller – 2016
  - Coverage driven system call fuzzing
    - Uses built in GCC port of ASAN coverage
    - gcc -fsanitize-coverage=trace-pc
  - Exposes coverage via /sys/kernel/debug/kcov
  - Template driven for system call fuzzing
  - Relies heavily on KASAN to catch bugs



```
write(fd fd, buf buffer[in], count len[buf])
pwrite64(fd fd, buf buffer[in], count len[buf], pos fileoff)
writev(fd fd, vec ptr[in, array[iovec_in]], vlen len[vec])
pwritev(fd fd, vec ptr[in, array[iovec_in]], vlen len[vec], off fileoff)
lseek(fd fd, offset fileoff, whence flags[seek_whence])
```

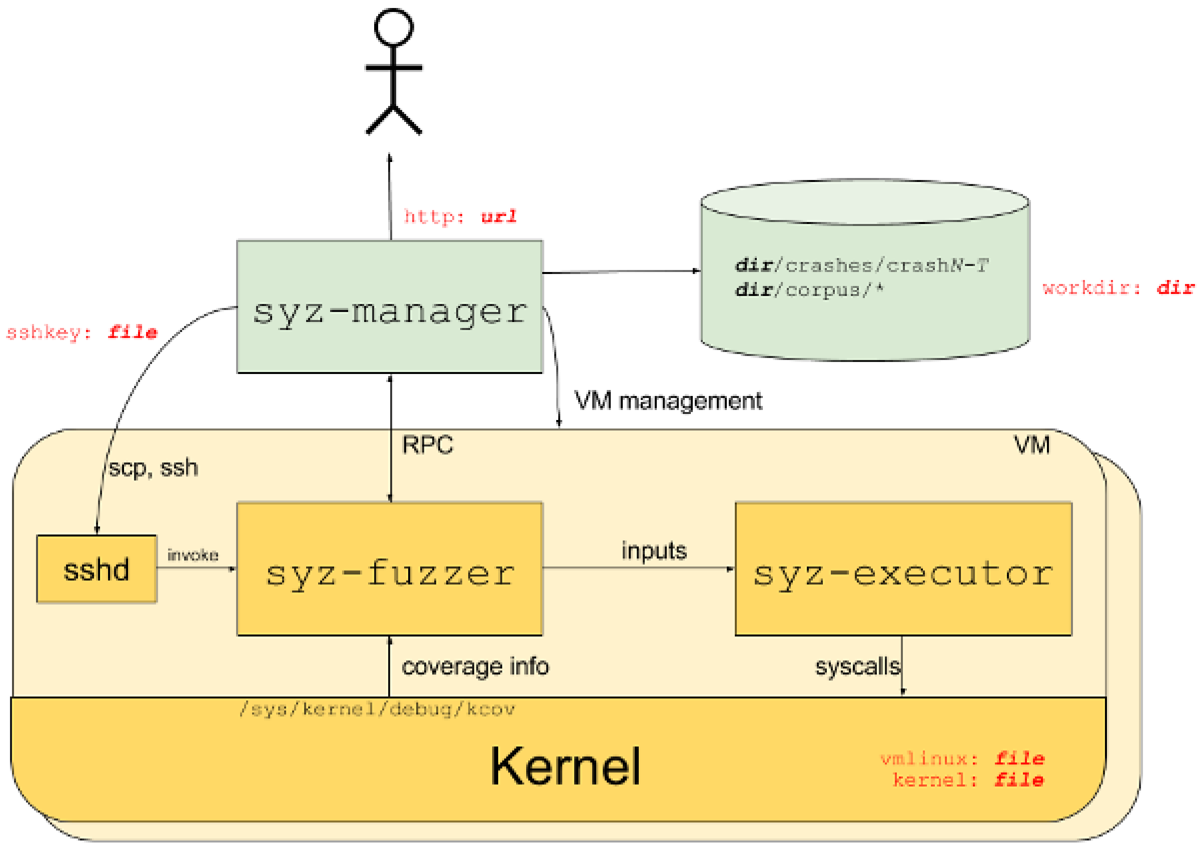
# Linux Kernel Fuzzing

- Syzkaller - 2016
  - Good support tooling
  - WebUI for monitoring
  - Good logging
  - Repro minimizer



# Linux Kernel Fuzzing

- Syzkaller - 2016
  - Very effective, but..
  - Complicated to get setup properly
  - Complex workflow
  - Not easily retargetable



Talos

# Linux Kernel Fuzzing

- TriforceAFL – 2016
  - Tim Newsham & Jesse Hertz (NCC Group)

  - AFL compatible QEMU based coverage fuzzer
  - Added fork server to QEMU post-boot
  - Added a great serialization technique for APIs
    - Allows to fuzz APIs via a file format

# Linux Kernel Fuzzing

- TriforceAFL – 2016
  - Tim Newsham & Jesse Hertz (NCC Group)

  - Extends QEMU trace support in AFL to target kernel
  - COW fork() of QEMU after boot for performance
  - Extends native ISA with custom hypercalls (aflCall)
    - startForkserver
    - getWork
    - startWork
    - endWork

# Linux Kernel Fuzzing

- TriforceAFL – 2016
  - Tim Newsham & Jesse Hertz (NCC Group)

  - Uses syscall templates / shapes
  - Serializes system calls into files to fuzz with AFL
  - Supports sequences of system calls

Syscall shapes
buffer, len, int
fd, buffer
fd, buffer, buffer, int
fd, buffer, buffer, int, int
fd, buffer, int
etc

Syscall types
Int
Buffer
BufferLength
FileContents
FileName
FileTableNumber

# Demo

## TriforceAFL

# Windows Kernel Fuzzing

# Kernel Attack Surface

- Kernels attack surface includes any untrusted input
  - Userland
    - System calls, file parsers, software interrupts
  - Devices
    - Network, USB, Firewire, etc
- Two categories: structured input or APIs

# Windows Kernel Attack Surface

- System Calls
  - ntoskrnl.sys
    - Windows system services
    - ~465 system calls
  - win32k.sys
    - Kernel mode Graphics Display Interface support
    - ~1216 system calls

# Windows Kernel Attack Surface

- win32k.sys File Parsers
  - Fonts
    - TTF, OTF, FON
  - Images
    - BMP, JPEG, CUR, ANI, ICO
  - Metafiles
    - EMF, WMF

# Windows Kernel Attack Surface

- Other attack surface
  - Graphics drivers
  - Audio drivers
  - Network drivers
  - Print drivers

- See other publications for deeper dives into attack surface

# Windows Kernel Fuzzing

- Legacy
  - ioctlfuzzer – Dimitry Oleksander (cr4sh)
  - Misc Syscall fuzzers
  - Misc file format fuzzers

- Techniques
  - Random syscall arguments or ioctl input
  - Hooking and interception (ioctlfuzzer)
  - Dumb or structured file fuzzing

# Windows Kernel Fuzzing

- KernelFuzzer – 2016
  - James Loureiro and Georgi Geshev
  - Windows system API fuzzer

- Techniques
  - Type aware API fuzzing
  - Manual definition of generators per-type
  - Pre-generated HANDLE tables
  - Outputs C code for each testcase to repro after crash

# Windows Kernel Fuzzing

- KernelFuzzer – 2016
    - James Loureiro and Georgi Geshev
    - Windows system API fuzzer

- Observations
    - Type aware API fuzzing is effective
    - Manual definition of generators is tedious
    - Can benefit from TriforceAfl style API sequence generation

# Windows Kernel Fuzzing

GOOD NEWS!

API fuzzing has a type-aware strategy and tools

BAD NEWS!

IOCTLs and Graphics drivers are opaque blobs

Sounds like we need for evolutionary fuzzing!

# Windows Graphics Driver Fuzzing

# Windows Graphics Driver Fuzzing

- Windows Graphics Hierarchy
  - Gdi32.dll -> Dxgkrnl.sys -> HW driver

- Interesting Direct3D functions
  - D3DKMTEscape
  - D3DKMTRender
  - D3DKMTCreateAllocation
  - D3DKMTCreateContext
  - etc

# Windows Graphics Driver Fuzzing

- D3DKMTEscape

  ```
  NTSTATUS D3DKMTEscape(
    _In_ const D3DKMT_ESCAPE *pData
  );
  ```

  - Entry point for internal graphics functionality
  - Each driver implements a proprietary format for *pData
    - A few header fields and command data
  - This is a perfect target for evolutionary file format style fuzzing

# Windows Graphics Driver Fuzzing

- D3DKMTEscape

```
NTSTATUS D3DKMTEscape(
  _In_ const D3DKMT_ESCAPE *pData
);
```

  - Entry point for internal graphics functionality
  - Each driver implements a proprietary format for *pData
    - A few header fields and command data
  - This is a perfect target for evolutionary file format style fuzzing

# Windows Graphics Driver Fuzzing

- Search for usage of D3DKMTEscape:

```
"C:\Program Files\Git\bin\bash.exe"
export output="/tmp/dumpbin.txt"
rm $output
for i in `find . -type d` ; \
    do echo $i ; dumpbin -imports "$i/*.exe" ; dumpbin "$i/*.dll" ; \
done | tee $output

export srch="Dump|D3DKMT"
egrep $srch $output | grep -B2 D3D

Dump of file ./DisplaySwitch.exe
                          B1 D3DKMTNetDispStopMiracastDisplayDevice
                          AD D3DKMTNetDispQueryMiracastDisplayDeviceSupport
Dump of file ./igfxCUIService.exe
                          65 D3DKMTCloseAdapter
                          A7 D3DKMTOpenAdapterFromDeviceName
                          81 D3DKMTEscape
Dump of file ./ProximityUxHost.exe
                          AD D3DKMTNetDispQueryMiracastDisplayDeviceSupport
```

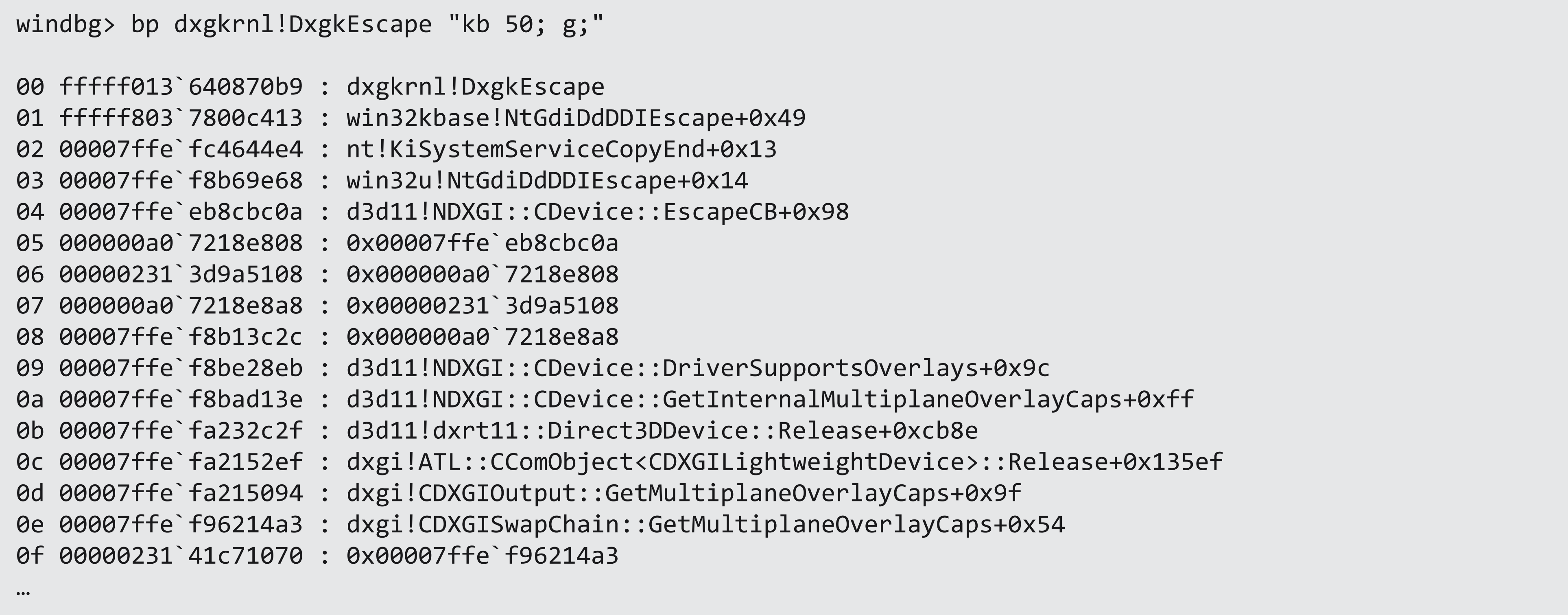
# Windows Graphics Driver Fuzzing

- Search for usage of D3DKMTEscape:

```
windbg> bp dxgkrnl!DxgkEscape ".echo DxgkEscape; kb 50; g;"

3: kd> kb 30
 # RetAddr           : Call Site
00 fffff803`7800c413  : win32kbase!NtGdiDdDDIEscape
01 00007ffe`fc4644e4  : nt!KiSystemServiceCopyEnd+0x13
02 00007ffe`f8b69e68  : win32u!NtGdiDdDDIEscape+0x14
03 00007ffe`ebb595f7  : d3d11!NDXGI::CDevice::EscapeCB+0x98
04 00000000`00000000  : igd10iumd64!OpenAdapter10_2+0x64a7b7
```

# Windows Graphics Driver Fuzzing

- Search for usage of D3DKMTEscape:



```
windbg> bp dxgkrnl!DxgkEscape "kb 50; g;"

00 fffff013`640870b9 : dxgkrnl!DxgkEscape
01 fffff803`7800c413 : win32kbase!NtGdiDdDDIEscape+0x49
02 00007ffe`fc4644e4 : nt!KiSystemServiceCopyEnd+0x13
03 00007ffe`f8b69e68 : win32u!NtGdiDdDDIEscape+0x14
04 00007ffe`eb8cbc0a : d3d11!NDXGI::CDevice::EscapeCB+0x98
05 000000a0`7218e808 : 0x00007ffe`eb8cbc0a
06 00000231`3d9a5108 : 0x000000a0`7218e808
07 000000a0`7218e8a8 : 0x00000231`3d9a5108
08 00007ffe`f8b13c2c : 0x000000a0`7218e8a8
09 00007ffe`f8be28eb : d3d11!NDXGI::CDevice::DriverSupportsOverlays+0x9c
0a 00007ffe`f8bad13e : d3d11!NDXGI::CDevice::GetInternalMultiplaneOverlayCaps+0xff
0b 00007ffe`fa232c2f : d3d11!dxrt11::Direct3DDevice::Release+0xcb8e
0c 00007ffe`fa2152ef : dxgi!ATL::CComObject<CDXGILightweightDevice>::Release+0x135ef
0d 00007ffe`fa215094 : dxgi!CDXGIOutput::GetMultiplaneOverlayCaps+0x9f
0e 00007ffe`f96214a3 : dxgi!CDXGISwapChain::GetMultiplaneOverlayCaps+0x54
0f 00000231`41c71070 : 0x00007ffe`f96214a3
…
```

# Windows Graphics Driver Fuzzing

- Intel HD Graphics Driver – igdkmd64.sys
  - 7.5 MB graphics driver

  - This won't end well ...

# Windows Graphics Driver Fuzzing

- TALOS-2016-0087 (Piotr Bania)
  - Intel HD Graphics Windows Kernel Driver (igdkmd64) RCE Vulnerability

```
igdkmd64!hybDriverEntry+1485b0
fffff801`61fd0920 ff9050020000    call    qword ptr [rax+250h]
…
fffff801`61fb33b1 : igdkmd64!hybDriverEntry+0x1485b0
fffff801`61ee4166 : igdkmd64!hybDriverEntry+0x12b041
fffff801`61edfa4a : igdkmd64!hybDriverEntry+0x5bdf6
fffff801`61ed5b1f : igdkmd64!hybDriverEntry+0x576da
fffff801`61edc798 : igdkmd64!hybDriverEntry+0x4d7af
fffff801`61ed51b5 : igdkmd64!hybDriverEntry+0x54428
fffff801`61e48613 : igdkmd64!hybDriverEntry+0x4ce45
fffff801`61e48507 : igdkmd64+0x26613
fffff801`60d1ea34 : igdkmd64+0x26507
fffff801`60ceffef : dxgkrnl!DXGADAPTER::DdiEscape+0x48
fffff960`002c563b : dxgkrnl!DxgkEscape+0x54f
fffff800`ac5d41b3 : win32k!NtGdiDdDDIEscape+0x53
00000000`770574aa : nt!KiSystemServiceCopyEnd+0x13
00000000`00000000 : 0x770574aa
```

# Windows Graphics Driver Fuzzing

- NVIDIA Graphics Driver – nvlddmkm.sys
  - ~800 graphics handling functions

  - This also won't end well …

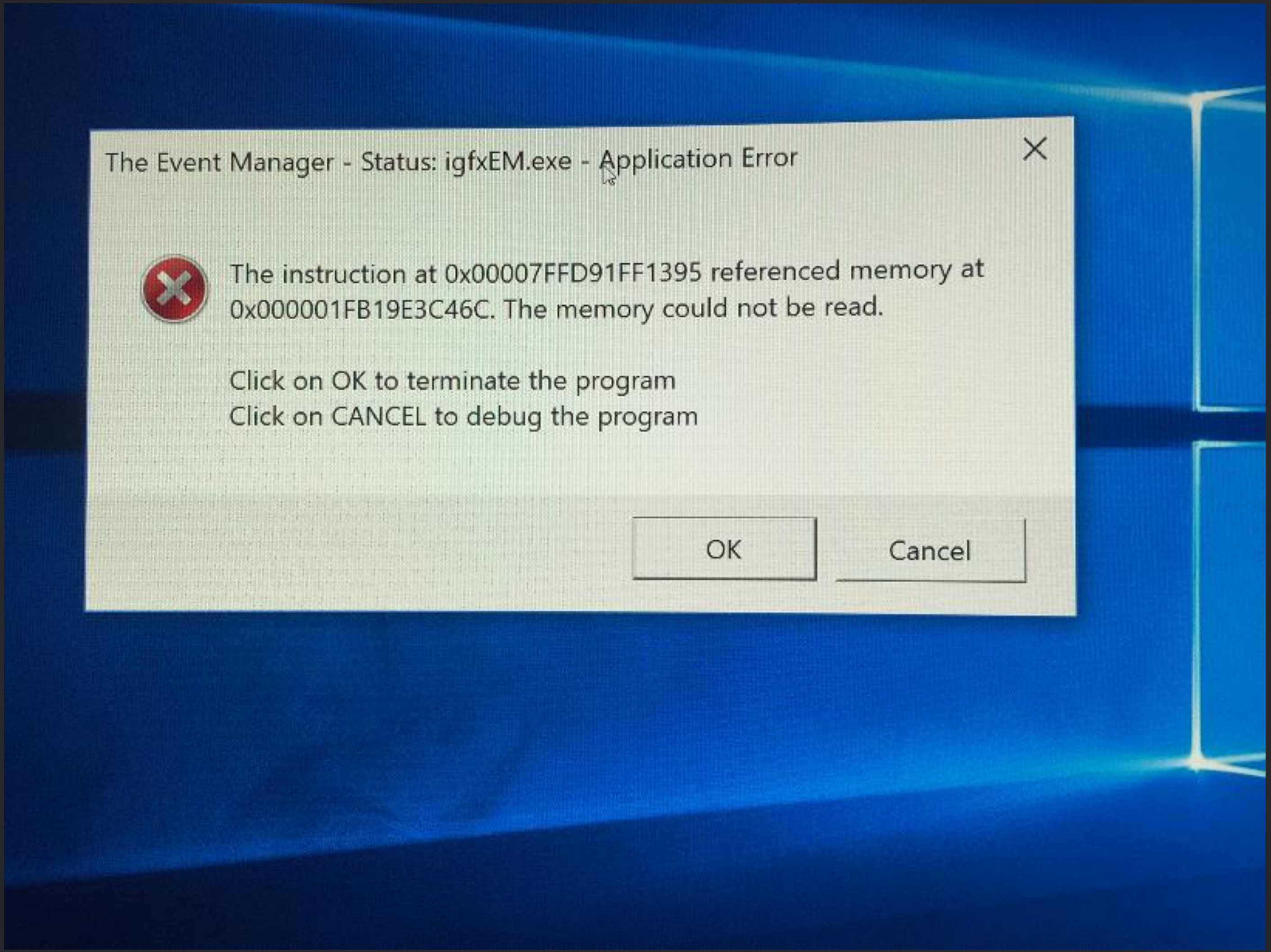
# Windows Graphics Driver Fuzzing

- TALOS-2016-0217 (Piotr Bania)
  - Nvidia Windows Kernel Mode Driver ZwSetValueKey Denial Of Service

```
nt!memcpy+0xa0:
fffff801`b0bcfc20 f30f6f040a      movdqu  xmm0,xmmword ptr [rdx+rcx] ds:ffffd000`26a45ff8=??
…
ffffd000`26a44408 fffff801`b0bde42c : nt!KeBugCheckEx
…
ffffd000`26a44808 fffff801`b0f26473 : nt!memcpy+0xa0
ffffd000`26a44810 fffff801`b0fbcd18 : nt!CmpSetValueDataNew+0x157
ffffd000`26a44860 fffff801`b0f0f588 : nt! ?? ::NNGAKEGL::`string'+0x27928
ffffd000`26a448d0 fffff801`b0e3a977 : nt!CmSetValueKey+0x784
ffffd000`26a449e0 fffff801`b0bcebb3 : nt!NtSetValueKey+0x55f
ffffd000`26a44bb0 fffff801`b0bc7020 : nt!KiSystemServiceCopyEnd+0x13
ffffd000`26a44db8 fffff801`4175a51a : nt!KiServiceLinkage
ffffd000`26a44dc0 fffff801`4175a051 : nvlddmkm+0xb751a
…
ffffd000`26a44f70 fffff801`41f44769 : nvlddmkm+0xc0faf
ffffd000`26a44fb0 fffff801`41f39e24 : nvlddmkm!nvDumpConfig+0x1253a1
…
ffffd000`26a45580 fffff801`413604f8 : nvlddmkm!nvDumpConfig+0xdc075
ffffd000`26a45650 fffff801`413c5b4e : dxgkrnl!DXGADAPTER::DdiEscape+0x48
ffffd000`26a45680 fffff960`002d41d3 : dxgkrnl!DxgkEscape+0x802
ffffd000`26a45ab0 fffff801`b0bcebb3 : win32k!NtGdiDdDDIEscape+0x53
…
```

# Demo

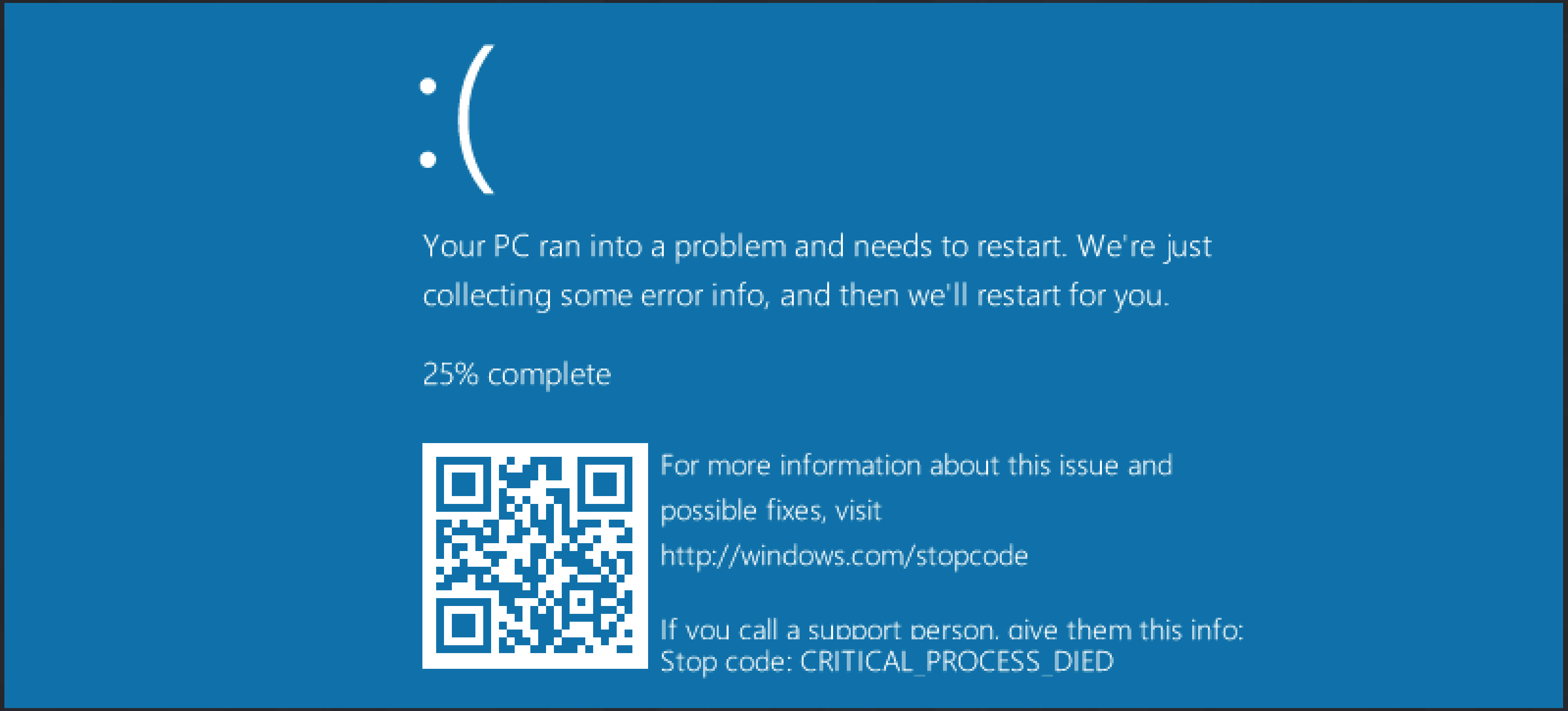
## winafl-intelpt vs idgkmd64.sys



The Event Manager - Status: igfxEM.exe - Application Error

The instruction at 0x00007FFD91FF1395 referenced memory at 0x000001FB19E3C46C. The memory could not be read.

Click on OK to terminate the program
Click on CANCEL to debug the program



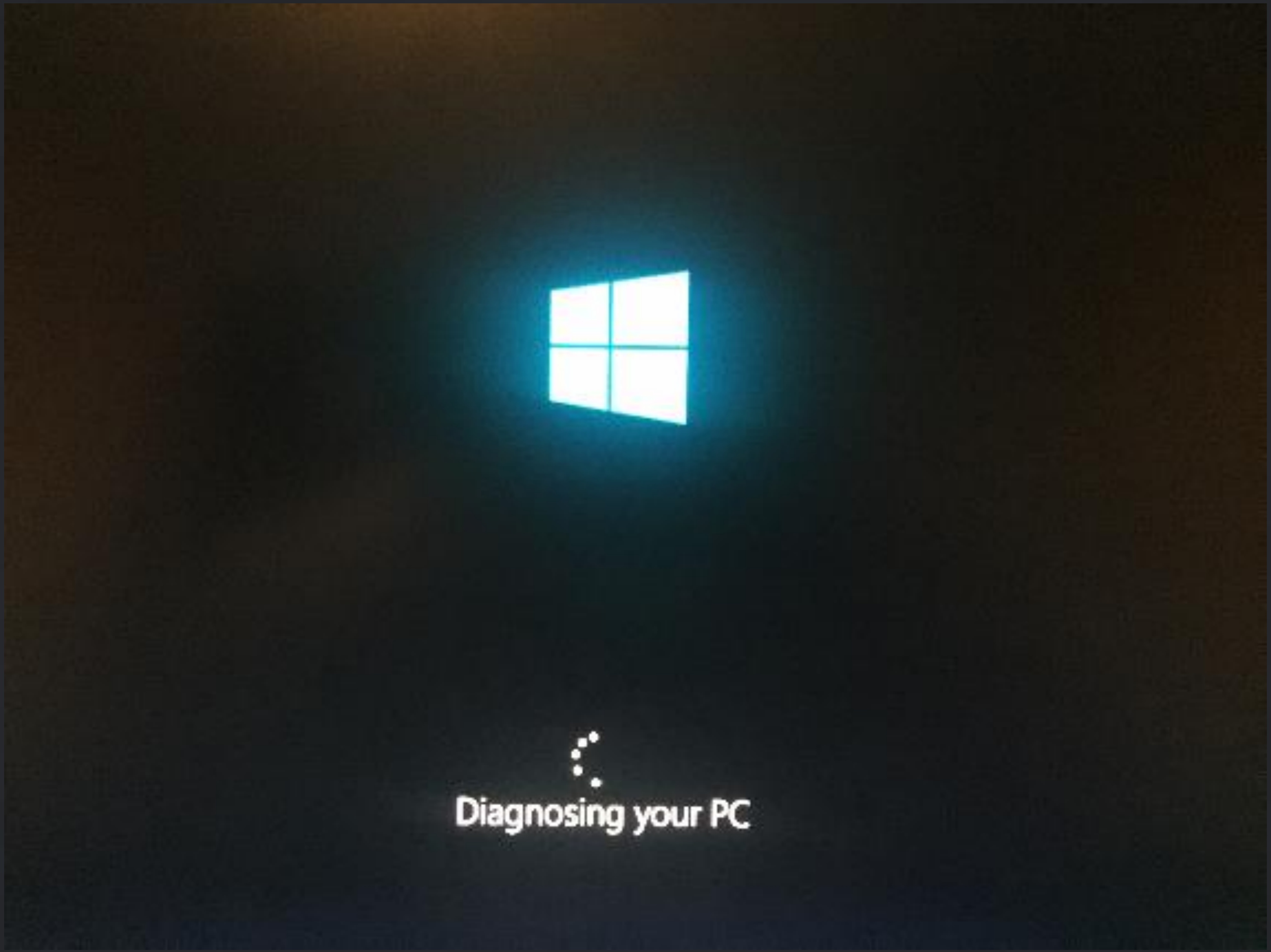
:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

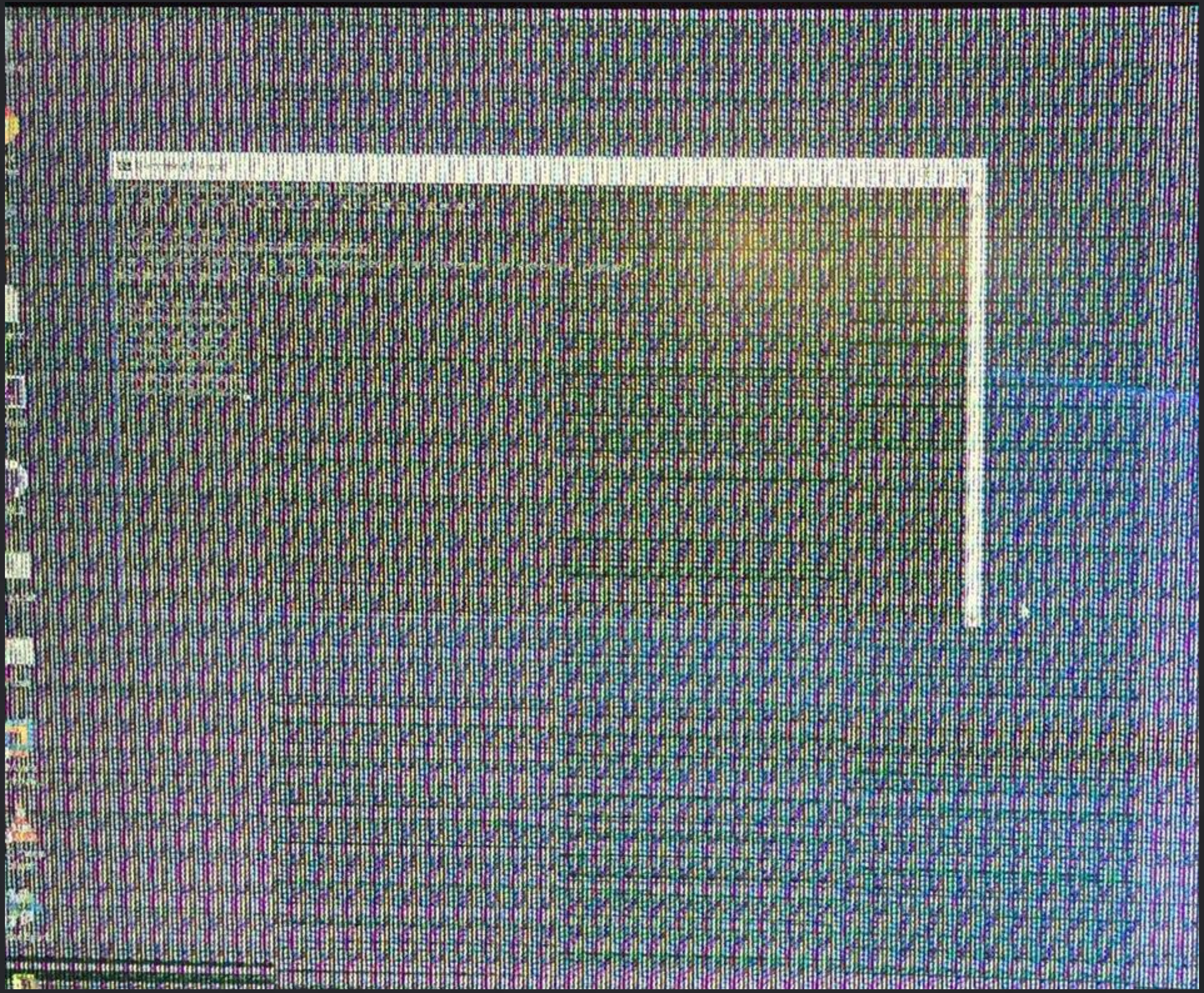
25% complete

For more information about this issue and possible fixes, visit http://windows.com/stopcode

If you call a support person, give them this info:
Stop code: CRITICAL_PROCESS_DIED



Diagnosing your PC



# Conclusions

- Kernels expose a massive amount of attack surface
- Hardware tracing enables code coverage for tricky targets
- Coverage guided kernel fuzzing is new and promising

- Get the code! – https://github.com/intelpt
  - Windows PT Driver available since Jan 2017
  - WinAFL-IntelPT available today
  - PtCov library available next week

# Talos

talosintel.com
blogs.cisco.com/talos
@talossecurity

@richinseattle
rjohnson@moflow.org

cisco