Go Speed Tracer

Richard Johnson
ToorCon San Diego 2016
Introduction

• Richard Johnson
  – Research Manager
  – Cisco Talos
• Team
  – Aleksandar Nikolich
  – Ali Rizvi-Santiago
  – Marcin Noga
  – Piotr Bania
  – Tyler Bohan
  – Yves Younan
• Special Contributor
  – Andrea Allevi

• Talos Vulndev
  – Third party vulnerability research
    • 170 bug finds in last 12 months
      – Microsoft
      – Apple
      – Oracle
      – Adobe
      – Google
      – IBM, HP, Intel
      – 7zip, libarchive, NTP
  – Security tool development
    • Fuzzers, Crash Triage
  – Mitigation development
    • FreeSentry
Introduction

• Agenda
  – Tracing Applications
  – Guided Fuzzing
  – Binary Translation
  – Hardware Tracing

• Goals
  – Understand the attributes required for optimal guided fuzzing
  – Identify areas that can be optimized today
  – Deliver performant and reusable tracing engines
Applications

• Software Engineering
  – Performance Monitoring
  – Unit Testing

• Malware Analysis
  – Unpacking
  – Runtime behavior
  – Sandboxing

• Mitigations
  – Shadow Stacks
  – Memory Safety checkers
Applications

• **Software Security**
  – **Corpus distillation**
    • Minimal set of inputs to reach desired conditions
  – **Guided fuzzing**
    • Automated refinement / genetic mutation
  – **Crash analysis**
    • Crash bucketing
    • Graph slicing
    • Root cause determination
  – **Interactive Debugging**
Tracing Engines

- OS Provided APIs
  - Debuggers
    - ptrace
    - dbgeng
    - signals
  - Hook points
    - Linux LTT(ng)
    - Linux perf
    - Windows Nirvana
    - Windows AppVerifier
    - Windows Shim Engine
  - Performance counters
    - Linux perf
    - Windows PDH

Check out Alex Ionescu’s RECON 2015 talk
Tracing Engines

- **Binary Instrumentation**
  - Compiler plugins
    - gcc-gcov
    - llvm-cov
  - Binary translation
    - Valgrind
    - DynamoRIO
    - Pin
    - DynInst
    - Frida and others
    - ...
Tracing Engines

- Native Hardware Support
  - Single Step / Breakpoint
  - Intel Branch Trace Flag
  - Intel Last Branch Record
  - Intel Branch Trace Store
  - Intel Processor Trace
  - ARM CoreSight
Guided Fuzzing
Evolutionary Testing

- Early work was whitebox testing
- Source code allowed graph analysis prior to testing
- Fitness based on distance from defined target
- Complex fitness landscape
  - Difficult to define properties that will get from A to B
- Applications were not security specific
  - Safety critical system DoS
Guided 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
- Focused on security bugs
Sidewinder

- Embleton, Sparks, Cunningham 2006
- **Features**
  - Simple genetic algorithm approach
    - crossover, mutation, fitness
  - Mutated context free grammar instead of sample fuzzing
  - Markov process for fitness
    - Analyzes probability of path taken by sample
  - Block coverage via debugger API
    - Reduced overhead by focusing on subgraphs
Sidewinder

- Embleton, Sparks, Cunningham 2006
- **Contributions**
  - Genetic algorithms for fuzzing
  - Markov process for fitness
  - System allows selection of target code locations
- **Observations**
  - Never opensourced
  - Interesting concepts not duplicated
Evolutionary Fuzzing System

• Jared DeMott 2007
• Features
  – Block coverage via Process Stalker
    • Windows Debug API
    • Intel BTF
  – Stored trace results in SQL database
    • Lots of variables required structured storage
  – Traditional genetic programming techniques
    • Code coverage + diversity for fitness
    • Sessions
    • Pools
    • Crossover
    • Mutation
Evolutionary Fuzzing System

• Jared DeMott 2007

• Contributions
  – First opensource implementation of guided fuzzing
  – Evaluated function vs block tracing
    • For large programs found function tracing was equally effective
    • Likely an artifact of doing text based protocols

• Observations
  – Academic
    • Approach was too closely tied to traditional genetic algorithms
    • Not enough attention to performance or real world targets
    • Only targeted text protocols
American Fuzzy Lop

- Michal Zalewski 2013
  - Bunny The Fuzzer 2007
- Features
  - Block coverage via compile time instrumentation
  - Simplified approach to genetic algorithm
    - Edge transitions are encoded as tuple and tracked in global map
    - Includes coverage and frequency
  - Uses variety of traditional mutation fuzzing strategies
  - Dictionaries of tokens/constants
  - First practical high performance guided fuzzer
  - Helper tools for minimizing test cases and corpus
  - Attempts to be idiot proof
Amercian Fuzzy Lop

- Michal Zalewski 2013
  - Bunny The Fuzzer 2007

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

• Michal Zalewski 2013
  – Bunny The Fuzzer 2007

• Observations
  – KISS works when applied to guided fuzzing
  – Performance top level priority in design
    • Source instrumentation can't be beat
    • Evolutionary system hard to beat without greatly increasing complexity / cost
  – Simple to use, finds tons of bugs
  – Fostered a user community
    • Developer contributions somewhat difficult
  – Current state of the art due to good engineering and feature set
  – Only mutational fuzzer system to have many third-party contributions
    • Binary support via QEMU and Dyninst
    • More robust compiler instrumentations, ASAN support
    • Parallelization, client/server targeting
• Robert Swiecki 2010
  – Guided fuzzing added in 2015
• Features
  – Block coverage
    • Hardware performance counters
    • ASanCoverage
  – Bloom filter for trace recording
  – User-supplied mutation functions
  – Linux, FreeBSD, OSX, Cygwin support
• Contributions
  – First guided fuzzer to focus on hardware tracing support
• Observations
  – Naive seed selection for most algorithms, only the elite survive (OTTES)
    • Some modes use bloom filter
  – Easy to extend, active development
Choronzon

• Features
  – Brings back specific genetic programming concepts
  – Contains strategies for dealing with high level input structure
    • Chunk based
    • Hierarchical
    • Containers
  – Format aware serialization functionality
  – Uses DBI engines for block coverage (PIN / DynamoRIO)
  – Attempts to be cross-platform

• Contributions
  – Reintroduction of more complex genetic algorithms
  – Robust handling of complex inputs through user supplied serialization routines

• Observations
  – Performance not a focus
Honorable mentions

- **autodafe**
  - Martin Vuagnoux 2004
  - First generation guided fuzzer using pattern matching via API hooks

- **Blind Code Coverage Fuzzer**
  - Joxean Koret 2014
  - Uses off-the-shelf components to assemble a guided fuzzer
    - radamsa, zzuf, custom mutators
    - drcov, COSEINC RunTracer for coverage

- **covFuzz**
  - Atte Kettunen 2015
  - Simple node.js server for guided fuzzing
  - custom fuzzers, ASanCoverage
Guided Fuzzing

• Required
  – Fast tracing engine
    • Block based granularity
  – Fast logging
    • Memory resident coverage map
  – Fast evolutionary algorithm
    • Minimum of global population map, pool diversity

• Desired
  – Portable
  – Easy to use
  – Helper tools
  – Grammar detection

• AFL and Honggfuzz still most practical options
Binary Translation
Binary Translation

• Binary translation is a robust program modification technique
  – JIT for hardware ISAs

• General overview is straightforward
  – Copy code to cache for translation
  – Insert instructions to modify original binary
  – Link blocks into traces

• Performance comes from smart trace creation
  – Originally profiling locations for hot trace
  – Early optimizations in Dynamo from HP
    • Next Executing Tail
    • Traces begin at backedge or other trace exit
  – Ongoing optimization work happens here
    • VMware - Early Exit guided
Binary Translation

• Advantages
  – Supported on most mainstream OS/archs
  – Can be faster than hardware tracing
  – Can easily be targeted at certain parts of code
  – Can be tuned for specific applications

• Disadvantages
  – Performance overhead
    • Introduces additional context switch
  – ISA compatibility not guaranteed
  – Not always robust against detection or escape
Valgrind

- Obligatory slide
- Lots of deep inspection tools
- VEX IR is well suited for security applications

- Slow and Linux only, DynamoRIO good replacement

- Many cool tools already exist
  - Flayer
  - Memgrind
Pin

• “DBT with training wheels”
• Features
  – Trace granularity instrumentation
    • Begin at branch targets, end at indirect branch
  – Block/instruction level hooking supported
  – Higher level C++ API w/ helper routines
  – Closed source
• Observations
  – Delaying instrumentation until trace generation is slower
  – Seems most popular with casual adventurers
  – Limited inlining support
  – Less tuning options
  – Cannot observe blocks added to cache so ‘hit trace’ not possible
VOID Trace(TRACE trace, VOID *v)
{
    for (BBL bbl = TRACE_BblHead(trace); BBL_Valid(bbl); bbl = BBL_Next(bbl))
    {
        BBL_InsertCall(bbl, IPOINT_ANYWHERE, AFUNPTR(basic_block_hook),
                       IARG_FAST_ANALYSIS_CALL, IARG_END);
    }
}
**DynamoRIO**

- "A connoisseur's DBT"
- **Features**
  - Block level instrumentation
    - Blocks are directly copied into code cache
  - Direct modification of IL possible
  - Portable
    - Linux, Windows, Android
    - x86/x64, ARM
  - C API / BSD Licensed (since 2009)
- **Observations**
  - Much more flexible for block level instrumentation
  - Performance is a priority, Windows is a priority
  - Powerful tools like Dr Memory
    - Shadow memory, taint tracking
    - Twice as fast as Valgrind memcheck
DynamoRIO

**Example**

```c
event_basic_block(void *drcontext, void *tag, instrlist_t *bb, 
    bool for_trace, bool translating)
{
    instr_t *instr, *first = instrlist_first(bb);
    uint flags;
    /* Our inc can go anywhere, so find a spot where flags are dead. */
    for (instr = first; instr != NULL; instr = instr_get_next(instr))
    {
        flags = instr_get_arith_flags(instr);
        /* OP_inc doesn't write CF but not worth distinguishing */
        if (TESTALL(EFLAGS_WRITE_6, flags) && !TESTANY(EFLAGS_READ_6, flags))
            break;
    }
...
```
Example

```c
if (instr == NULL)
    dr_save_arith_flags(drcontext, bb, first, SPILL_SLOT_1);

instrlist_meta_preinsert(bb,
    (instr == NULL) ? first : instr,
    INSTR_CREATE_inc(drcontext,
        OPND_CREATE_ABSMEM((byte *)&global_count, OPSZ_4)));

if (instr == NULL)
    dr_restore_arith_flags(drcontext, bb, first, SPILL_SLOT_1);
return DR_EMIT_DEFAULT;
```
DynInst

• “Static rewriting IS possible!”
• Features
  – Static rewriting support
    • Dynamically linked binaries only
    • Eliminates issues with instruction cache misses common to DBT engines
  – Function level analysis
    • Tools must manually walk Dyninst provided CFG to instrument blocks
  – Modular C++ API / LGPL
• Observations
  – Fastest binary instrumentation out there
  – Development is slow
    • Patches we sent in for PE relocation support still not merged
  – Building Dyninst is NP-Hard
    • Use my Dockerfile on github.com/talos-vulndev/afl-dyninst
bool insertBBCallback(BPatch_binaryEdit * appBin, BPatch_function * curFunc, 
char *funcName, BPatch_function * instBBIncFunc,int *bbIndex)
{
    unsigned short randID;
    BPatch_flowGraph *appCFG = curFunc->getCFG();
    BPatch_Set <BPatch_basicBlock *> allBlocks;
    BPatch_Set <BPatch_basicBlock *>::iterator iter;
    for (iter = allBlocks.begin (); iter != allBlocks.end (); iter++)
    {
        unsigned long address = (*iter)->getStartAddress ();

        randID = rand() % USHRT_MAX;
        BPatch_Vector <BPatch_snippet *> instArgs;
        BPatch_constExpr bbId (randID);
        instArgs.push_back (&bbId);
        ...
}

DynInst

• Example

```c
...
BPatch_point *bbEntry = (*iter)->findEntryPoint();
BPatch_funcCallExpr instIncExpr (*instBBIncFunc, instArgs);
BPatchSnippetHandle *handle =
    appBin->insertSnippet (instIncExpr, *bbEntry, BPatch_callBefore,
                         BPatch_lastSnippet);

(*bbIndex)++;
}
return true;
```
Tuning Binary Translation

- Only instrument indirect branches
- Delay instrumentation until input is seen
- Only instrument threads that access the data
- Move instrumentation logic to analysis routines
  - Some APIs provide IF-THEN-ELSE analysis with optimization
- Avoid trampolines
  - Be aware of code locality and instruction cache
  - Directly inline instructions, modify AST if possible
- Inject a fork server if repeatedly executing DBT
  - See our turbotrace tool
Hardware Tracing
CPU Event Monitoring

• Modern CPUs contain Performance Monitoring Units (PMU)
• Model Specific Registers (MSR) used for configuration
  – Requires privileged execution (kernel or better) to access
• Types
  – Event Counters
    • Polled on-demand
  – Event Sampling (non-precise)
    • Interrupts triggered when counters hit modulus value
  – Precise Event Sampling (PEBS)
    • Uses 'Debug Store'
    • Physical memory buffers
    • Interrupt when full
• Use Linux perf / pmu-tools to experiment
Interrupt Programming

- Interrupts - low level messaging system for system devices
  - **CPU Exceptions**
    - GPF, SINGLE_STEP
  - **Hardware Interrupts**
    - Memory mapped or IRQ based
    - All Device I/O
  - **Software Interrupts**
    - System calls (int 0x80)
    - Breakpoints

- OS/hypervisor drivers required to configure interrupt handlers
  - Privileged registers or interrupt vector tables
Interrupt Programming

• Interrupt Service Routines (ISR)
  – Registered by operating systems and drivers as callbacks
• CPU checks interrupt flag (IF) register after each instruction
  – cli and sti instructions control whether IF is checked
• CPU indexes the interrupt vector table to find appropriate handler
  – Context stored / restored while servicing interrupt

• Historically Familiar Interrupts:
  – int 1 - Single Step (TF)
  – int 3 - Single opcode, specifically designed for debugging
  – int 10h - Any Demosceners?
  – int 24h - DOS Critical Error Handler

Who remembers:
I/O Device Specific Error Message
Abort, Retry, Ignore, Fail?
Interrupt Programming

- Programmer checklist
  - Memory must not be swapped
  - Use static variables if necessary
  - Must wrap functions with assembly
    • disable interrupts
    • push all registers
    • call interrupt handler
    • pop all registers
    • iretd
Its a Trap

• Single Stepping
  – Enabled by setting the Trap Flag
  – After each instruction, CPU checks flag and fires exception if enabled
  – Accessible from userspace

• Branch Trace Flag
  – Modifies single step behavior to trap on branch
  – Single flag in IA32_DEBUGCTL MSR
  – Requires kernel privileges to write to MSR
  – Windows includes a mapping from DR7 to set MSR

  – SS/BTF traps are slooooooooow, not applicable for vulnerability research
IA32_DEBUGCTL Register

- MSR Address 0x1d9
  - LBR [0] - Enable Last Branch Record mechanism
  - BTF [1] - when enabled with TF in EFLAGS does single stepping on branches
  - TR [6] - enables Tracing (sending BTMs to system bus)
  - BTS [7] - enables sending BTMs to memory buffer from system bus
  - BTINT [8] - full buffer generates interrupt otherwise circular write
  - BTS_OFF_OS [9] - does not count for priv. level 0
  - BTS_OFF_USR [10] - does not count for priv. level 1,2,3
  - FRZ_PERFMON_ON_PMI [12] - disable all performance counters on a PMI
  - UNCORE_PMI_EN [13] - uncore counter interrupt generation
  - SMM_FRZ [14] - event counters are frozen during SMM
• First generation hardware branch tracing via PMU
• Allows configurable memory buffer for trace storage
• MSR_IA32_DS_AREA MSR defines storage location
• DS_AREA_RECORD stored for each branch

```c
struct DS_AREA {
    u64 bts_buffer_base;
    u64 bts_index;
    u64 bts_absolute_maximum;
    u64 bts_interrupt_threshold;
    u64 pebs_buffer_base;
    u64 pebs_index;
    u64 pebs_absolute_maximum;
    u64 pebs_interrupt_threshold;
    u64 pebs_event_reset[4];
};
```

```c
struct DS_AREA_RECORD {
    u64 flags;
    u64 ip;
    u64 regs[16];
    u64 status;
    u64 dla;
    u64 dse;
    u64 lat;
};
```
Branch Trace Store

CPU

Instruction Pipeline

Last Branch Record (LBR)

Branch From
Branch To

LBR Stack
32 Model Specific Registers (MSRs)

System Bus

Branch Trace Store (BTS)

Branch From
Branch To
Misspredicted

Branch Trace Message (BTH)

Tracing (TR)

L2 Cache

DS Save Area
Branch Trace Store

• Branches in LBR registers spill to DS_AREA
• Interrupts only when buffer is full
• Steps to enable BTS
  – Allocate memory and set MSR_IA32_DS_AREA
  – Add interrupt handler to IDT
  – Register interrupt vector with APIC
    • apic_write(APIC_LVTPC, pebs_vector);
  – Select events with MSR_IA32_EVNTSEL0
    • EVTSEL_EN | EVTSEL_USR | EVTSEL_OS
  – Enable PEBS mode with MSR_IA32_PEBS_ENABLE
  – Enable CPU perf recording with MSR_IA32_GLOBAL_CTRL
• Significantly faster than BTF
• Still impractical for high speed tracing
**Intel Processor Trace**

- Next generation hardware tracing support
  - Introduced in Broadwell/Skylake architecture
  - Per-hardware tracing thread

- Goal: full system branch tracing with 5-15% overhead

- Software support available in
  - Linux 4.1+ perf subsystem
  - Standalone Linux reference driver simple-pt
  - Intel VTune / System Studio**
    - Remote debugging only
  - Talos IntelPT driver!
    - Windows localhost high speed hardware tracing FTW!
Intel Processor Trace

Features

- Can trace SMM, HyperVisor, Kernel, Userspace [CPL -2 to 3]
- Logs directly to physical memory
  - Bypasses CPU cache and eliminates TLB cache misses
  - Can be a contiguous segment or a set of ranges
  - Ringbuffer snapshot or interrupt mode supported
- Minimal log format
  - One bit per conditional branch
  - Only indirect branches log dest address
  - Interrupts log source and destination
  - Decoding log requires original binaries and memory map
- Filter logging based on CR3
- Linux can automatically add log to coredump
- GDB Support
Intel Processor Trace

- 90+ pages in Intel Software Developer Manuals
- Randomly flipping bits doesn’t work here 😞

Check with CPUID
- EAX = 0x14 - Intel Processor Trace
  - EBX
    - Bit 00: If 1, Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1, and that IA32_RTIT_CR3_MATCH MSR can be accessed.
    - Bit 01: If 1, Indicates support of Configurable PSB and Cycle-Accurate Mode.
    - Bit 02: If 1, Indicates support of IP Filtering, TraceStop filtering, and preservation of Intel PT MSRs across warm reset.
    - Bit 03: If 1, Indicates support of MTC timing packet and suppression of COFI-based packets.
  - ECX
    - Bit 00: If 1, Tracing can be enabled with IA32_RTIT_CTL.ToPA = 1, hence utilizing the ToPA output scheme; IA32_RTIT_OUTPUT_BASE and IA32_RTIT_OUTPUT_MASK_PTRS MSRs can be accessed.
    - Bit 01: If 1, ToPA tables can hold any number of output entries, up to the maximum allowed by the MaskOrTableOffset field of IA32_RTIT_OUTPUT_MASK_PTRS.
    - Bit 02: If 1, Indicates support of Single-Range Output scheme.
    - Bit 03: If 1, Indicates support of output to Trace Transport subsystem.
    - Bit 31: If 1, Generated packets which contain IP payloads have LIP values, which include the CS base component

Packet Generation (ECX = 1)
- EAX
  - Bits 2:0: Number of configurable Address Ranges for filtering.
  - Bit 31:16: Bitmap of supported MTC period encodings
- EBX
  - Bits 15:0: Bitmap of supported Cycle Threshold value encodings
  - Bit 31:16: Bitmap of supported Configurable PSB frequency encodings
Intel Processor Trace (for programmers)

- Hardware support detection
  - CPUID with leaf 0x7 indicates support for Intel PT
  - If supported, CPUID with leaf 0x14 can return the supported PT features

- Trace Record Filtering
  - Code Privileged Level (CPL) - kernel vs userspace
  - PML4 Page Table – single process / CR3 (page-table) filtering
  - Instruction Pointer – up to 4 ranges of addresses can be specified

- Log Output Configuration
  - Single range
  - Table of Physical Addresses (ToPA)
Intel Processor Trace (for programmers)

- Single Buffer Trace Logging
  - Circular or Interrupt modes (Hardware logging support)
  - Reserve memory – *MmAllocateContiguousMemory* (Windows Drivers)
  - Set the proper MSRs
    - MSR_IA32_RTIT_OUTPUT_BASE
    - MSR_IA32_RTIT_OUTPUT_MASK_PTRS
  - Start the Tracing setting the “TraceEn” flag in the control register
  - Processor logs to in a circular-manner unless interrupt flag configured
Table of Physical Address (ToPA) Trace Logging

- For large traces, non-contiguous physical memory must be used
- ToPA is compatible with Windows Memory Descriptor List (MDL)
- MDL is a Windows data structure for tracking physical->linear mappings
  - ToPA is compatible with Windows MDL data structure!

```c
// Grab the physical address:
PHYSICAL_ADDRESS physAddr = MmGetPhysicalAddress(lpBuffVa);
perCpuData.u.Simple.lpTraceBuffPhysAddr = (ULONG_PTR)physAddr.QuadPart;

// Allocate the relative MDL
PMDL pPtMdl = IoAllocateMdl(lpBuffVa, (ULONG)perCpuData.qwBuffSize, FALSE, FALSE, NULL);
if (pPtMdl) perCpuData.pTraceMdl = pPtMdl;
```
Intel Processor Trace

Complex log format - decode with opensource libipt library!
Intel Processor Trace (for programmers)

- **Packet Types**
  - **Packet Stream Boundary (PSB)**
    - Heartbeat packet generated at regular intervals (configurable)
  - **Paging Information (PIP)**
    - Notification of CR3 Page Table changes
  - **Timing (TSC, MTC & CYC)**
    - Useful for wall-clock comparisons or synchronization of logs across CPU threads
  - **Control Flow (TNT, TIP, FUP)**
    - TNT – Taken/Not-Taken for conditional branches
    - TIP – Taken IP address for indirect branches
    - FUP – Flow Update
Intel Processor Trace

PT packet log, binaries, and software runtime data are used to reconstruct the precise execution flow.

Intel CPU 0..n

Configure & enable Intel PT

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

Intel PT HW

Intel PT Software Decoder

Intel PT-enabled Tools

Binary Image Files
**Intel Processor Trace**

- **How to use: Linux perf tools** *(apt: linux-tools-common)*

```
$ perf list | grep intel_pt
intel_pt/ [Kernel PMU event]

$ perf record -e intel_pt//u date
Sun Oct 11 11:35:07 EDT 2015
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.027 MB perf.data ]

$ perf report
...
# Samples: 1 of event 'instructions:u'
# Event count (approx.): 157207
# Overhead  Command  Shared Object  Symbol
# ........  .......  .............  ..........................
# 100.00%  date     libc-2.21.so   [.] _nl_intern_locale_data
  |   _nl_intern_locale_data
  |   _nl_load_locale_from_archive
  |   _nl_find_locale
  |   setlocale
...
```
**Intel Processor Trace**

- **How to use:** simple-pt reference driver

```
% sptcmd -c tcall taskset -c 0 ./tcall
cpu 0 offset 1027688, 1003 KB, writing to ptout.0
...
Wrote sideband to ptout.sideband
% sptdecode --sideband ptout.sideband --pt ptout.0 | less

TIME      DELTA  INSNs   OPERATION
frequency 32
0   [+0]     [+   1] _dl_aux_init+436
     [+   6] __libc_start_main+455 -> _dl_discover_osversion
...
    [+  13] __libc_start_main+446 -> main
    [+  9]     main+22 -> f1
    [+  4]     f1+9  -> f2
    [+  2]     f1+19 -> f2
    [+  5]     main+22 -> f1
    [+  4]     f1+9  -> f2
    [+  2]     f1+19 -> f2
    [+  5]     main+22 -> f1
...
```
Intel Processor Trace

- Talos IntelPT driver

```c
struct PERPROCESSOR_PT_DATA {
    LPVOID lpTraceBuffVa; // + 0x00 - VA Pointer to a contiguous memory buffer
    ULONG_PTR lpTraceBuffPhysAddr; // + 0x08 - The physical address of the contiguous memory
    DWORD dwBuffSize; // + 0x10 - The physical buffer size
    ULONG_PTR lpTargetProcCr3; // + 0x18 - The process to monitor CR3
};
```
Intel Processor Trace

• Talos IntelPT driver

```c
struct INTEL_PT_CAPABILITIES {
    BOOLEAN bCr3Filtering : 1; // [0] - CR3 Filtering Support (Indicates that IA32_RTIT_CTL.CR3Filter can be set to 1)
    BOOLEAN bConfPsbAndCycSupported : 1; // [1] - Configurable PSB and Cycle-Accurate Mode
    BOOLEAN bIpFiltering : 1; // [2] - IP Filtering and TraceStop supported, and Preserve Intel PT MSRs across warm reset
    BOOLEAN bMtcSupport : 1; // [3] - IA32_RTIT_CTL.MTCEn can be set to 1, and MTC packets will be generated (section 36.2.5)
    BOOLEAN bTopaOutput : 1; // [4] - Utilize the ToPA output scheme
    BOOLEAN bTopaMultipleEntries : 1; // [5] - ToPA tables maximum allowed (MaskOrTableOffset)
...
Intel Processor Trace

• Talos IntelPT driver

```c
BOOLEAN bTransportOutputSupport : 1;  // [7] - Output to Trace Transport Subsystem Supported
// (Setting IA32_RTIT_CTL.FabricEn to 1 is supported)
BOOLEAN bIpPcksAreLip : 1;  // [8] - IP Payloads are LIP
BYTE numOfAddrRanges;  // + 0x01 - Number of Address Ranges
SHORT mtcPeriodBmp;  // + 0x02 - Bitmap of supported MTC Period Encodings
SHORT cycThresholdBmp;  // + 0x04 - Bitmap of supported Cycle Threshold values
SHORT psbFreqBmp;  // + 0x06 - Bitmap of supported Configurable PSB
```

Frequency encoding
Intel Processor Trace

• Talos IntelPT driver

```c
// Write the target CR3 value
__writemsr(MSR_IA32_RTIT_CR3_MATCH, targetCr3);

// Start tracing:
rtitCtlDesc.Fields.CR3Filter = 1;
rtitCtlDesc.Fields.FabricEn = 0;
rtitCtlDesc.Fields.Os = 0;
rtitCtlDesc.Fields.User = 1; // Trace the user mode process
rtitCtlDesc.Fields.ToPA = 0; // We use the single-range output scheme
rtitCtlDesc.Fields.BranchEn = 1;

// If (ptCap.bMtcSupport) {
//    rtitCtlDesc.Fields.MTCEn = 1;
//    rtitCtlDesc.Fields.MTCFreq = 10;
//}
rtitCtlDesc.Fields.TSCEn = 1;
rtitCtlDesc.Fields.TraceEn = 1; // Switch the tracing to ON dude :-)
__writemsr(MSR_IA32_RTIT_CTL, rtitCtlDesc.All);
```
Intel Processor Trace

- Talos IntelPT driver

C:\code\intelpt\instdrv.exe /I windowsptdriver.sys
C:\code\intelpt\testintelpt.exe c:\windows\system32\notepad.exe
C:\code\intelpt>..\libipt\ptdump pt_dump.bin | findstr /V pad | more

00000000000006e8  psb
00000000000006fe  tsc  4e1ef46cbc
0000000000000708  cbr  1f
000000000000070c  psbend
0000000000000716  tsc  4e1ef8af9

... 
0000000000000ce0  cbr  1c
0000000000000cf0  tip  2: ?????????4d515400
0000000000000cf5  tnt.8  ..!
0000000000000cf8  tip  2: ?????????4bb10ca0
0000000000000cfd  tnt.8  !....
0000000000000cfe  tnt.8  
0000000000000d00  tip  2: ?????????4d515400
0000000000000d05  tnt.8  ..!
0000000000000d08  tip  2: ?????????1a91e4f0
0000000000000d0d  tnt.8  !!!!!!
Conclusion

• Evoloutionary algorithms have a lot to offer for automation
  – https://github.com/talos-vulndevelop

• Initial investment in development pays dividends
  – Use correct engine for long term deployment
  – Designing tracing engines is not for everyone

• Hardware tracing is approaching software performance

• This code is opensource software
  – https://github.com/talos-vulndevelop
Thank You!