PROTECTING YOUR NETWORK

Richard Johnson
Offensive Summit 2016
Go Speed Tracer
Introduction

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

• 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
    • Competitive analysis
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
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

![Diagram showing a mutated context free grammar and block coverage via debugger API.](image)
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 (complex) 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
American 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
American 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, more robust compiler instrumentations
honggfuzz

- 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
Choronzon

• Features
  – Aims to be cross platform
  – 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
  – Attempts to be cross-platform

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

- 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
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
- 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
Pin

Example

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
  - Powerful tools like Dr Memory
    - Shadow memory, taint tracking
    - Twice as fast as Valgrind memcheck
DynamoRIO

• Example

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

“Kitchen sink binary translation”

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
- Inject a fork server if repeatedly executing DBT
  - See our turbotrace tool
CPU Event Monitoring

- Modern CPUs contain Performance Monitoring Units (PMU)
- Model Specific Registers (MSR) used for configuration
- 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
- Special registers allow OS to configure interrupt handlers
  - CPU Exceptions
    - GPF, SINGLE_STEP
  - Hardware Interrupts
    - Memory mapped or IRQ based
    - All Device I/O
  - Software Interrupts
    - System calls (int 0x80)
    - Breakpoints
Interrupt Programming

- Interrupt Service Routines (ISR)
  - Registered by Operating systems and drivers
- CPU checks IF flag after each instruction
  - cli and sti instructions control IF
- CPU indexes the interrupt descriptor table to find appropriate handler
  - Context stored / restored while servicing interrupt
- Special Interrupts
  - int 1   - Single Step (TF)
  - int 3   - Single opcode, specifically designed for debugging
  - int 10h - Any Demosceners?
  - int 24h - DOS Critical Error Handler
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
  – sloooooooow, not applicable

• 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
IA32_DEBUGCTL

- 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
Branch Trace Store

- First generation hardware branch tracing via PMU
- Allows configurable memory buffer for trace storage
- MSR_IA32_DS_AREA MSR defines storage location

```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];
};

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

- First generation hardware branch tracing via PMU
- Allows configurable memory buffer for trace storage

MSR_IA32_DS_AREA MSR defines storage location

```
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];
};
```

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

PU

- u64 pebs_absolute_maximum;
- u64 pebs_interrupt_threshold;
- u64 pebs_event_reset[4];

L2 Cache

Last Branch Record (LBR)

Instruction Pipeline

System Bus

Branch Trace Store (BTS)

DS Save Area

Tracing (TR)
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
    - \texttt{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
- Goal: full system tracing with 5-15% overhead
- Available in
  - Linux 4.1 perf subsystem
  - Standalone Linux reference driver simple-pt
  - Intel VTune / System Studio**
    - Does not seem to work with Windows 10
Intel Processor Trace

- **Features**
  - Ring -3? 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

Processor Trace Packets

Redirection

Environment

Misc

Execution

Trace

Alignment

Time

Outside traced program

Inside traced program
Intel Processor Trace

- 90+ pages in Intel Software Developer Manuals

- 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

- 90+ pages in Intel Software Developer Manuals
- Opensource parsing library! – Libipt
Intel Processor Trace

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

PT packet log (per logical processor)

Intel PT Software Decoder

Intel PT-enabled Tools

Runtime data, including:
- Map linear-address to image files
- Map CR3 value to application
- Log module load/unload and JIT info

Binary Image Files

Configure & enable Intel PT

RingO Agent (OS, VMM, BIOS, Driver, ...)

Intel CPU 0..n

Intel PT HW
Intel Processor Trace

How to use

$ 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

```bash
% sptcmd --c tcall taskset --c 0 --ctcall

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
...
```
Next Step / Conclusions
Thank You!