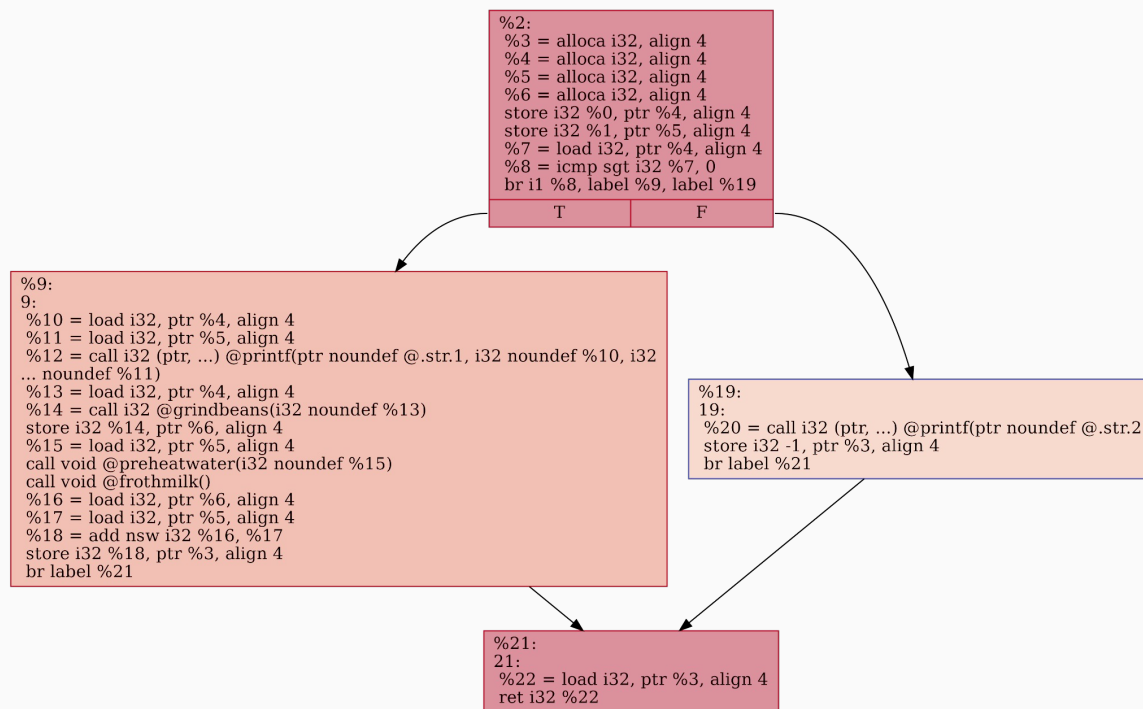


Reverse Engineering and Control Flow Analysis with Intel Processor Trace

Hagen Paul Pfeifer <hagen@jauu.net>

Introduction and Overview

Objective: Intel PT is a surprisingly powerful tool for understanding how software runs, but it's still unfamiliar to many - and not widely documented for reverse engineering use cases. *This talk aims to make Intel PT more accessible and show you how it can transform the way how to analyze program flow* - focusing on RE use cases.



CFG for 'brewcoffee' function

```
$ clang -Xclang -disable-O0-optnone -emit-llvm -S -o f1.ll f1.c
$ opt f1.ll -passes=dot-cfg -cfg-dot-filename-prefix=f1 -disable-output
```

Agenda

1. Control Flow Analysis: Concepts and Techniques
2. Introduction to Intel Processor Trace
3. Using Perf with Intel PT for Control Flow Analysis
4. Practical Tips and Extensions
5. Q&A and Closing Remarks

Basic Elements and Concepts

Control Flow Analysis: Concepts and Techniques

Static Analysis:

- Examines code without execution
- Analyzes structure (control/data flow) for design intent

Dynamic Analysis:

- Observes code behavior during execution
- Traces memory, function calls, and control flow

Basic Block:

- A straight-line sequence of instructions with a single entry point and a single exit point, where the flow of control enters at the beginning and leaves at the end without any possibility of branching except at the end

Control Flow Graphs (CFG): logical flow between basic blocks within a function

Call Graphs: relationships between functions, showing which functions call others

```
292: fcn.0000133b (int64_t arg2, int64_t arg3);
; arg int64_t arg2 @ rsi
; arg int64_t arg3 @ rdx
; var signed int64_t var_4h @ rbp-0x4
; var int64_t var_10h @ rbp-0x10
; var signed int64_t var_14h @ rbp-0x14
0x0000133b 55          pushq %rbp
0x0000133c 4889e5      %rsp = %rbp
0x0000133f 4883ec20    subq $0x20,%rsp
0x00001343 897dec      movl %edi,-0x14(%rbp)
0x00001346 c745fc      movl $0,-4(%rbp)
0x0000134d e996        goto loc_0x13e8
; CODE XREF from fcn.0000133b @ 0x13ec(x)
0x00001352 837dec5a    cmpl $0x5a,-0x14(%rbp) ; 'Z'
0x00001356 7e          if (var > 0) goto loc_0x1386
0x00001358 488d05f30d leaq 0xdf3 0x0000135f,%rax
0x0000135f 488945f0    movq %rax,-0x10(%rbp) ; arg3
0x00001363 488b55f0    movq -0x10(%rbp)%rdx ; arg2
0x00001367 8b45ec      movl -0x14(%rbp)%eax
0x0000136a 89c6        movl %eax,%esi
0x0000136c 488d05ed0d leaq 0xded 0x00001373,%rax
0x00001373 4889c7      %rax = %rdi
0x00001376 b8          movl $0,%eax
0x0000137b e8c0fcffff callq sym.imp.printf ;[1]
0x00001380 8345ec02    addl $2,-0x14(%rbp)
0x00001384 eb5e        goto loc_0x13e4
; CODE XREF from fcn.0000133b @ 0x1356(x)
0x00001386 837dec5f    cmpl $0x5f,-0x14(%rbp) ; '1'
0x0000138a 7e2e        if (var <= 0) goto loc_0x13ba
0x0000138c 488d050e    leaq 0xe00 0x00001393,%rax
0x00001393 488945f0    movq %rax,-0x10(%rbp) ; arg3
0x00001397 488b55f0    movq -0x10(%rbp)%rdx ; arg2
0x0000139b 8b45ec      movl -0x14(%rbp)%eax
0x0000139e 89c6        movl %eax,%esi
0x000013a0 488d05f90d leaq 0xdf9 0x000013a7,%rax
0x000013a7 4889c7      %rax = %rdi
0x000013aa b8          movl $0,%eax
0x000013af e88fcfffff callq sym.imp.printf ;[1]
0x000013b4 836dec02    subl $2,-0x14(%rbp)
0x000013b8 eb2a        goto loc_0x13e4
; CODE XREF from fcn.0000133b @ 0x138a(x)
0x000013ba 488d05120e leaq 0xe12 0x000013c1,%rax
0x000013c1 488945f0    movq %rax,-0x10(%rbp) ; arg3
0x000013c5 488b55f0    movq -0x10(%rbp)%rdx ; arg2
0x000013c9 8b45ec      movl -0x14(%rbp)%eax
0x000013cc 89c6        movl %eax,%esi
0x000013ce 488d050b0e leaq 0xeb0 0x000013d5,%rax
0x000013d5 4889c7      %rax = %rdi
0x000013d8 b8          movl $0,%eax
0x000013db e85efcffff callq sym.imp.printf ;[1]
0x000013dd eb0e        goto loc_0x13f2
; CODE XREFS from fcn.0000133b @ 0x1384(x), 0x13b8(x)
0x000013e2 8345ec01    addl $1,-4(%rbp)
; CODE XREF from fcn.0000133b @ 0x1340(x)
0x000013e8 837dfc02    cmpl $2,-4(%rbp)
0x000013ec 0f8e60ffff if (var <= 0) goto loc_0x1352
; CODE XREF from fcn.0000133b @ 0x13e2(x)
0x000013f2 837dfc03    cmpl $3,-4(%rbp)
0x000013f6 7527        if (var) goto loc_0x141f
0x000013f8 837dec5a    cmpl $0x5a,-0x14(%rbp) ; 'Z'
0x000013fb 7e06        if (var <= 0) goto loc_0x1404
0x000013fe 837dec5f    cmpl $0x5f,-0x14(%rbp) ; '1'
0x00001402 7e1b        if (var <= 0) goto loc_0x141f
; CODE XREF from fcn.0000133b @ 0x13fc(x)
0x00001404 8b45ec      movl -0x14(%rbp)%eax
0x00001407 89c6        movl %eax,%esi
0x00001409 488d05f00d leaq 0xdf0 0x00001410,%rax
0x00001410 4889c7      %rax = %rdi
0x00001413 b8          movl $0,%eax
0x00001418 e823fcffff callq sym.imp.printf ;[1]
0x0000141d eb3d        goto loc_0x145c
```

Static Control Flow Analysis

Control Flow Analysis: Concepts and Techniques

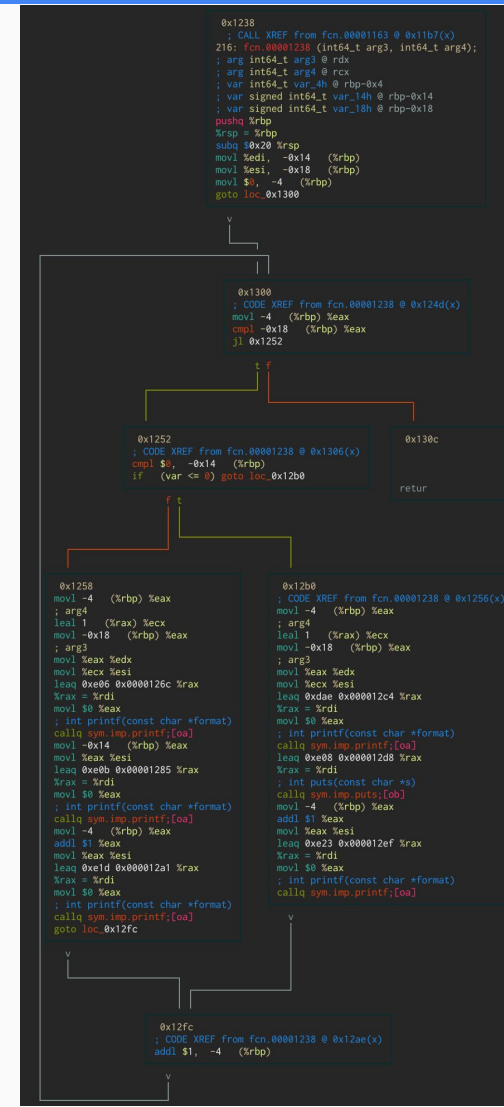
Static Control Flow Analysis

- The flow control graph is an important building block in (static) program analysis
- Control Flow Graph: a map that details execution paths through basic blocks and control paths
- Basic blocks as nodes and jumps/calls/rets/etc as edges
- Control flow can be constructed on local and global basis (function).
- Tools: Radare2, Binary Ninja, Ghidra, BinNavi, Hopper, Angr, IDA, ...

Disassembly → Building the CFG → Analyzing {Basic Blocks, Control Structures, Function Analysis, Cross-Referencing}

Goals: understand the flow of a program

- Understanding Program Logic
- Identifying Key Functions
- Detecting Obfuscation



Dynamic Flow Control Analysis

Control Flow Analysis: Concepts and Techniques

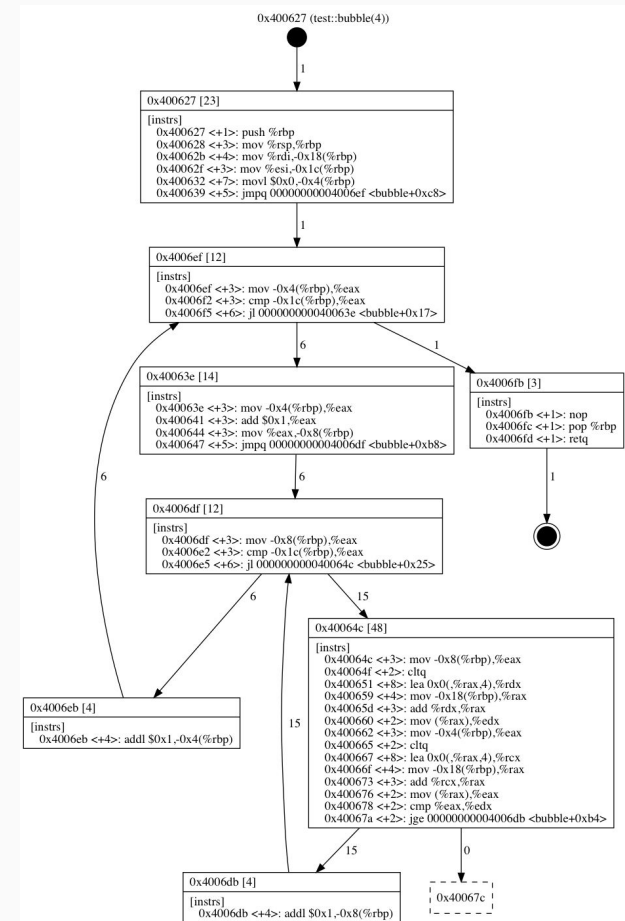
Static control flow has limits

- Certain control-flow transitions are inherently difficult or impossible to resolve:
- Indirect Jumps and Calls (`jmp eax`; `call [ebx]`), Dynamic Code Generation, Self-Modifying Code, Data-Driven Control Flow (wget example.com/.text)

Dynamic Flow Analysis: running it and observing the behavior in real-time

Why dynamic flow control is crucial

- Static control flow represents all possible branches and potential paths, but this doesn't reflect actual execution
- For example, a distribution kernel may have 10MiB of .text section, but only a small fraction of it is executed in reality
- Dynamic flow control captures the real execution paths, providing accurate insights into the code's behavior



Dynamic Control Flow Analysis

Control Flow Analysis: Concepts and Techniques

GDB - best in its class, comprehensive feature set, wide language, os and arch support, though it lacks built-in graphical views

R2 - reverse engineering framework and debugger (gdb too), flexible control flow visualization and scripting. Optional: use clutter for graphical frontend

DCFG (Intel Pin) - plugin for Intel Pin that dynamically generates call flow graphs, showing function relationships in real time, great for visualizing complex binary structure

CFGgrind (Valgrind) - Linux-based profiling tool with exhaustive tracing for detailed control flow graphs, offering depth over speed, ideal for precise analysis

Unicorn Emulation (QEMU) - emulates specific code segments efficiently, useful for isolated control flow analysis without full-system

```
RSI 0x7fffffffcd88 -> 0x7fffffffe06f <- '/home/pfeifer/foo'
R8 0
R9 0x7ffffff7fcbf40 (<_dl_fini>) <- pushq %rbp
R10 0x7fffffffd8b0 <- 0x800000
R11 0x202
R12 0
R13 0x7fffffffcd98 -> 0x7fffffffe081 <- 'HOME=/home/pfeifer'
R14 0x7fffffffd000 (<_rld_global>) -> 0x7fffffffe2e0 -> 0x55555554000 <- 0x10102464c457f
R15 0x555555557e08 (<__do_global_ctors_aux_fini_array_entry>) -> 0x5555555550e0 (<__do_global_ctors_aux>) <- endbr64
RBP 0x7fffffffd870 <- 1
RSP 0x7fffffffd870 <- 1
<RIP> 0x55555555142 (main+25) <- cmpl $0, -4(%rbp)

[ DISASM / x86-64 / set emulate on ]
0x55555555134 <main+11> movl $0x23, -4(%rbp) [0x7fffffffd86c] <= 0x23
0x5555555513b <main+18> movl -4(%rbp), %eax EAX, [0x7fffffffd86c] => 0x23
0x5555555513e <main+21> ptwritel %eax
> 0x55555555142 <main+25> cmpl $0, -4(%rbp) 0x23 - 0x0 EFLAGS => 0x202 [ cf pf af zf sf if df of ]
0x55555555146 <main+29> je main+37 <main+37>
0x55555555148 <main+31> addl $1, -4(%rbp) [0x7fffffffd86c] <= 36 (0x23 + 0x1)
0x5555555514c <main+35> jmp main+41 <main+41>
↓
0x55555555152 <main+41> movl $0x666, %eax EAX => 0x666
0x55555555157 <main+46> ptwritel %eax
0x5555555515b <main+50> movl -4(%rbp), %eax
0x5555555515e <main+53> popq %rbp

[ SOURCE (CODE) ]
In file: /home/pfeifer/pt-coffee-machine-sim/foo.c:9
4 {
5     int i = 0x23;
6
7     __asm__ volatile ("ptwrite %0" : : "r"(i));
8
9     if (i)
10         i++;
11     else
12         i--;
13
14     __asm__ volatile ("ptwrite %0" : : "r"(0x666));

[ STACK ]
00:0000 | rbp | rsp | 0x7fffffffd870 <- 1
01:0008 | +008 | 0x7fffffffd878 -> 0x7ffffffdd3d68 (<__libc_start_call_main+120>) <- movl %eax, %edi
02:0010 | +010 | 0x7fffffffd880 -> 0x7fffffffd870 -> 0x7fffffffd878 <- 0x38 /* '8' */
03:0018 | +018 | 0x7fffffffd888 -> 0x55555555129 (main) <- pushq %rbp
04:0020 | +020 | 0x7fffffffd890 <- 0x15554040
05:0028 | +028 | 0x7fffffffd898 -> 0x7fffffffd888 -> 0x7fffffffe06f <- '/home/pfeifer/foo'
06:0030 | +030 | 0x7fffffffd8a0 -> 0x7fffffffd888 -> 0x7fffffffe06f <- '/home/pfeifer/foo'
07:0038 | +038 | 0x7fffffffd8a8 <- 0x9915664ddede452

[ BACKTRACE ]
> 0 0x55555555142 main+25
1 0x7ffffffdd3d68 __libc_start_call_main+120
2 0x7ffffffdd3e25 __libc_start_main+133
3 0x55555555061 _start+33
```

pwndbg (gef, ...)

Complexity Layers in Reverse Engineering

Control Flow Analysis: Concepts and Techniques

Code Obfuscation: alters code structure to make it unreadable with techniques like control flow changes, modifies control flow to create complex, non-linear paths

Anti-Debugging: detects and disrupts debuggers using breakpoint and timing checks

Anti-Tampering: uses checksums or signatures to prevent unauthorized code changes

Packers/Cryptors: compress or encrypt code, only decrypting at runtime

Virtualization Protection: runs code in a custom virtual machine, complicating analysis

Self-Modifying Code: dynamically changes its own instructions during execution

Environmental Checks: alters behavior based on system configuration to evade analysis

Anti-Virtualization: detects VMs and behaves differently or refuses execution

Resource Obfuscation: encrypts data and resources to hide information

Note: Intel PT won't solve all challenges, but it can assist in certain areas and expand the personal toolkit.

What is Intel Processor Trace

Intel Processor Trace

Overview: Intel PT is a hardware-based tracing tool that provides highly detailed insights into program execution. No sampling

Tracing Scope: Intel PT traces branches, calls, returns, and special events like timestamps or exceptions in a highly compressed format. Note that unconditional jumps are not traced

Data Decoding: compressed format is decoded to reconstruct the full instruction stream. Since all control-flow-influencing instructions and their targets are traced, the complete control flow can be accurately reconstructed.

Performance Efficiency: Data is captured with minimal impact on performance and detectability, making it valuable for reverse engineering, with an overhead typically between 2% and 15%

Data Volume: Despite the compression, the amount of captured data can still be large (e.g., assuming a branch every 5 instructions)

Processor Support: Intel PT has been supported since Broadwell CPUs (2014) as a successor to Branch Trace Store (BTS)

Processor Trace Packets

Intel Processor Trace

TNT Taken Not Taken	TIP Target IP	FUP Flow Update Packet	PIP Paging Information Packet	MODE Mode	CBR Core Bus Ratio	PSB Packet Stream Boundary	OVF Overflow	TSC Time-Stamp Counter	MTC Mini Time Counter	CYC Cycle-Accurate Mode	PAD Padding
Inside		Outside	Execution			Trace		Time			Alignment
Redirecton			Environment					Misc			

Packet Variety: more than 10 packet types are available, each encoding specific types of data

Extended Packet Types: since Skylake, several additional packet types have been introduced, primarily for time-related data

Configurable Options: configuration options allow some control over some of the generated packet types

Examples:

- **TNT (Taken/Not Taken):** Encodes taken and not taken branches in two variants with 6 or 47 decision bits and also encodes returns
- **TIP (Target IP):** Encodes target addresses for indirect jumps, exceptions, and interrupts
- **CBR (Core-Bus Ratio):** Indicates changes in the ratio between core and bus clock speeds
- **CYC (Cycle Count):** Provides elapsed time in core clock cycles, relative to the last CYC packet

Raw Recording Information

Intel Processor Trace

```
objdump --disassemble=main  
--visualize-jumps=color foo
```

```
000000000001129 <main>:  
1129: 55          push    %rbp  
112a: 48 89 e5    mov     %rsp,%rbp  
112d: 89 7d ec    mov     %edi,-0x14(%rbp)  
1130: 48 89 75 e0  mov     %rsi,-0x20(%rbp)  
1134: c7 45 fc 23 00 00 00  movl    $0x23,-0x4(%rbp)  
113b: 8b 45 fc    mov     -0x4(%rbp),%eax  
113e: f3 0f ae e0  ptwrite %eax  
1142: 83 7d fc 00  cmpl    $0x0,-0x4(%rbp)  
1146: 74 06       je      114e <main+0x25>  
1148: 83 45 fc 01  addl    $0x1,-0x4(%rbp)  
114c: eb 04       jmp     1152 <main+0x29>  
114e: 83 6d fc 01  subl    $0x1,-0x4(%rbp)  
1152: b8 66 06 00 00  mov     $0x666,%eax  
1157: f3 0f ae e0  ptwrite %eax  
115b: 8b 45 fc    mov     -0x4(%rbp),%eax  
115e: 5d          pop     %rbp  
115f: c3          ret
```

```
int main(int ac, char **av)  
{  
    int i = 0x23;  
    __asm__ volatile ("ptwrite %0" : : "r"(i));  
    if (i)  
        i++;  
    else  
        i--;  
    __asm__ volatile ("ptwrite %0" : : "r"(0x666));  
    return i;  
}
```

```
r2 -Aq -c "s main; agf" foo
```

```
0x1129  
39: int main (int argc, char **argv, char **envp);  
; var uint32_t var_4h @ rbp-0x4  
; var int64_t var_20h @ rbp-0x20  
addb %al, (%rax)  
addb %al, (%rax)  
addb %al, (%rax)  
addb %al, (%rax)  
movl %esi, -0x20 (%rbp)  
movl $0x23, -4 (%rbp)  
cmpl $0, -4 (%rbp)  
if (lvar) goto loc_0x1147
```

```
0x1141  
addl $1, -4 (%rbp)  
goto loc_0x114b  
  
0x1147  
; CODE XREF from main @ 0x113f(x)  
subl $1, -4 (%rbp)
```

```
0x114b  
; CODE XREF from main @ 0x1145(x)  
movl -4 (%rbp) %eax  
popq %rbp  
retur
```

PT Recording Start

```
16 . 00001f0c: 59 d7          MTC 0xd7  
15 . 00001f0e: 8c            TNT NNNTTN (6)  
14 . 00001f0f: 04            TNT N (1)  
13 . 00001f10: 4d 1b ee 7a 5f TIP 0x5f7ae1b  
12 . 00001f15: 14            TNT NTN (3)  
11 . 00001f16: 6d 00 70 e4 ce 9f 55 TIP 0x559fcee47000  
10 . 00001f1d: 38            TNT TTNN (4)  
9 . 00001f1e: 2d 20 71      TIP 0x7120  
8 . 00001f21: 59 d8          MTC 0xd8  
7 . 00001f23: 1c            TNT TTN (3)  
6 . 00001f24: 6d 00 d9 9a 5f 7b 7f TIP 0x7f7b5f9ad900  
5 . 00001f2b: 14            TNT NTN (3)  
4 . 00001f2c: 59 d9          MTC 0xd9  
3 . 00001f2e: 14            TNT NTN (3)  
2 . 00001f2f: 6d 29 71 e4 ce 9f 55 TIP 0x559fcee47129  
1 . 00001f36: 02 12 23 00 00 00 PTWRITE 0x23 IP:0  
8893 . 00001f3c: 04            TNT N (1)  
1 . 00001f3d: 02 12 66 06 00 00 PTWRITE 0x666 IP:0  
2 . 00001f43: 0e            TNT TT (2)  
3 . 00001f44: 59 da          MTC 0xda  
4 . 00001f46: e0            TNT TTNNNN (6)  
5 . 00001f47: 0c            TNT TN (2)  
6 . 00001f48: 6d 20 ae 99 5f 7b 7f TIP 0x7f7b5f99ae20  
7 . 00001f4f: 04            TNT N (1)  
8 . 00001f50: 4d 70 48 81 5f TIP 0x5f814870  
9 . 00001f55: 04            TNT N (1)  
10 . 00001f56: 59 db          MTC 0xdb
```

PT Recording End

Introduction to Perf

Using perf with Intel PT for Control Flow Analysis

Linux Perf: in-kernel performance monitoring framework and userspace tool for profiling system and application performance, supporting hardware counters, tracepoints and custom event tracking

Events: hw, sw, tracepoint, pmu, sdt and metric's

Recording:

Generic Syntax



```
$ perf record --event pmu/config=M,config1=N,config3=K/ -- <workload>
```

Intel PT Specific



```
$ perf record -e intel_pt/ptw,cyc,cyc_thresh=5/u -o perf.data -- foo
```

Setting up Perf with Intel PT – Decoding

Using perf with Intel PT for Control Flow Analysis

Decoding is Key: recording is essential, but the true potential unfolds through decoding. Drill deep and examine data from different angles to analyze vast amounts of information and gain valuable insights

Advice: start with high-level analysis, then drill down to the instruction level as needed

- **Call Trace:** `perf script --call-trace --ns -F -cpu,-tid,-time`
- **Assembly Stream:** `perf script --insn-trace=disasm -F -cpu,-tid,-time`

Miscellaneous Decoding: various helpful decoding methods

- **Raw Trace:** `perf script --dump-raw-trace`
- **Mmap Events:** `perf script --no-itrace --show-mmap-events`
- **Branch Focus:** `perf script --itrac=iybxwpe -F+flags`
- **Power Events:** `perf script --itrac=p`
- **Decoder Debug:** `perf script --itrac=d`

Setting up Perf with Intel PT – Decoding

Using perf with Intel PT for Control Flow Analysis

Call Trace

Disassembly Trace

```
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_init
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) call_init
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) call_init
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _init_first
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __init_misc
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) strchr@plt
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) check_stdfiles_vtables
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _IO_stdfiles_init
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) call_init
foo (/home/pfeifer/foo) _start
foo (/home/pfeifer/foo) __libc_start_main@GLIBC_2.34
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __cxa_atexit
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __cxa_atexit
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) _new_exitfn
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) _init
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) frame_dummy
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) _dl_audit_preinit@plt
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __libc_start_call_main
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) _setjmp
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) main
foo IP: 0 payload: 0x23 #
foo IP: 0 payload: 0x666
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) exit
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __run_exit_handlers
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __call_tls_dtors
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) _dl_fini
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) pthread_mutex_lock@GLIBC_2.2.5
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_audit_activity_nsuid
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_sort_maps
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) dfs_traversal.part.0
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) dfs_traversal.part.0
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) dfs_traversal.part.0
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) dfs_traversal.part.0
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) memmove
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) pthread_mutex_unlock@GLIBC_2.2.5
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_call_fini
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) __do_global_dtors_aux
foo (/home/pfeifer/foo) __cxa_finalize@plt
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __unregister_atfork
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __unregister_atfork
foo (/home/pfeifer/foo) deregister_tm_clones
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_audit_objclose
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_call_fini
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_audit_objclose
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_call_fini
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_audit_objclose
foo (/usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2) _dl_audit_activity_nsuid
foo (/usr/lib/x86_64-linux-gnu/libc.so.6) __run_exit_handlers
```

```
7fd27a799d72 __libc_start_call_main@0x32 (/usr/lib/x86_64-linux-gnu/libc.so.6) testl %eax, %eax
7fd27a799d74 __libc_start_call_main@0x34 (/usr/lib/x86_64-linux-gnu/libc.so.6) jne __libc_start_call_main@0x81
7fd27a799d76 __libc_start_call_main@0x36 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq %fs:0x300, %rax
7fd27a799d7f __libc_start_call_main@0x3f (/usr/lib/x86_64-linux-gnu/libc.so.6) movq %rax, 0x68(%rsp)
7fd27a799d84 __libc_start_call_main@0x44 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq %fs:0x2f8, %rax
7fd27a799d8d __libc_start_call_main@0x4d (/usr/lib/x86_64-linux-gnu/libc.so.6) movq %rax, 0x70(%rsp)
7fd27a799d92 __libc_start_call_main@0x52 (/usr/lib/x86_64-linux-gnu/libc.so.6) leaq 0x20(%rsp), %rax
7fd27a799d97 __libc_start_call_main@0x57 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq %rax, %fs:0x300
7fd27a799da0 __libc_start_call_main@0x60 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq 0x182209(%rip), %rax
7fd27a799da7 __libc_start_call_main@0x67 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq 0x18(%rsp), %rsi
7fd27a799dac __libc_start_call_main@0x6c (/usr/lib/x86_64-linux-gnu/libc.so.6) movl 0x14(%rsp), %edi
7fd27a799db0 __libc_start_call_main@0x70 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq (%rax), %rdx
7fd27a799db3 __libc_start_call_main@0x73 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq 8(%rsp), %rax
7fd27a799db8 __libc_start_call_main@0x78 (/usr/lib/x86_64-linux-gnu/libc.so.6) callq %rax
55ae54c7f129 main+0x0 (/home/pfeifer/foo) pushq %rbp
55ae54c7f12a main+0x1 (/home/pfeifer/foo) movq %rsp, %rbp
55ae54c7f12d main+0x4 (/home/pfeifer/foo) movl %edi, -0x14(%rbp)
55ae54c7f130 main+0x7 (/home/pfeifer/foo) movq %rsi, -0x20(%rbp)
55ae54c7f134 main+0xb (/home/pfeifer/foo) movl $0x23, -4(%rbp)
55ae54c7f13b main+0x12 (/home/pfeifer/foo) movl -4(%rbp), %eax
55ae54c7f13e main+0x15 (/home/pfeifer/foo) f3 0f ae e0
55ae54c7f142 main+0x19 (/home/pfeifer/foo) cmpl $0, -4(%rbp)
55ae54c7f146 main+0x1d (/home/pfeifer/foo) je main+0x25
55ae54c7f148 main+0x1f (/home/pfeifer/foo) addl $1, -4(%rbp)
55ae54c7f14c main+0x23 (/home/pfeifer/foo) jmp main+0x29
55ae54c7f152 main+0x29 (/home/pfeifer/foo) movl $0x666, %eax
55ae54c7f157 main+0x2e (/home/pfeifer/foo) f3 0f ae e0
55ae54c7f15b main+0x32 (/home/pfeifer/foo) movl -4(%rbp), %eax
55ae54c7f15e main+0x35 (/home/pfeifer/foo) popq %rbp
55ae54c7f15f main+0x36 (/home/pfeifer/foo) retq
7fd27a799dba __libc_start_call_main@0x7a (/usr/lib/x86_64-linux-gnu/libc.so.6) movl %eax, %edi
7fd27a799dbb __libc_start_call_main@0x7c (/usr/lib/x86_64-linux-gnu/libc.so.6) callq exit+0x0
7fd27a7b1b90 exit+0x0 (/usr/lib/x86_64-linux-gnu/libc.so.6) subq $8, %rsp
7fd27a7b1b94 exit+0x4 (/usr/lib/x86_64-linux-gnu/libc.so.6) movl $1, %ecx
7fd27a7b1b99 exit+0x9 (/usr/lib/x86_64-linux-gnu/libc.so.6) movl $1, %edx
7fd27a7b1b9e exit+0xe (/usr/lib/x86_64-linux-gnu/libc.so.6) leaq 0x19aadb(%rip), %rsi
7fd27a7b1ba5 exit+0x15 (/usr/lib/x86_64-linux-gnu/libc.so.6) callq __run_exit_handlers+0x0
7fd27a7b1b99 __run_exit_handlers+0x0 (/usr/lib/x86_64-linux-gnu/libc.so.6) pushq %r15
7fd27a7b1b92 __run_exit_handlers+0x2 (/usr/lib/x86_64-linux-gnu/libc.so.6) pushq %r14
7fd27a7b1b94 __run_exit_handlers+0x4 (/usr/lib/x86_64-linux-gnu/libc.so.6) movq %rsi, %r14
7fd27a7b1b92 __run_exit_handlers+0x7 (/usr/lib/x86_64-linux-gnu/libc.so.6) pushq %r13
7fd27a7b1b92 __run_exit_handlers+0x9 (/usr/lib/x86_64-linux-gnu/libc.so.6) movl %edi, %r13d
7fd27a7b1b92 __run_exit_handlers+0xc (/usr/lib/x86_64-linux-gnu/libc.so.6) pushq %r12
7fd27a7b1b92 __run_exit_handlers+0xe (/usr/lib/x86_64-linux-gnu/libc.so.6) pushq %rbp
7fd27a7b1b92 __run_exit_handlers+0xf (/usr/lib/x86_64-linux-gnu/libc.so.6) pushq %rbx
7fd27a7b1b93 __run_exit_handlers+0x10 (/usr/lib/x86_64-linux-gnu/libc.so.6) subq $0x28, %rsp
7fd27a7b1b94 __run_exit_handlers+0x14 (/usr/lib/x86_64-linux-gnu/libc.so.6) movl %edx, 0x1c(%rsp)
```

Note: Intel XED is no longer required; starting from version v6.8-rc1-303-g8b767db33095, libcapstone is used as the disassembly engine

Custom Tooling

Using perf with Intel PT for Control Flow Analysis

Post-Processing Needs: sometimes, custom post-processing or integration with other tools is required. Two options exist

- **Parse perf script Output:** simple and effective for basic tasks, but can be time-consuming and computationally intensive
- **Integrate directly with perf script:** for more complex analyses, it's recommended to connect directly within perf script using mechanisms like `dlfilter` (and `dlarg`).

```
$ cat filter.c
int filter_event(void *data, const struct perf_dlfilter_sample *sample, void *ctx)
{
    if (sample->ip) {
        printf("IP: 0x%" PRIx64 "\n", sample->ip);
    }
    return 0;
}

$ gcc -o filter.so -shared -fPIC filter.c -ldl
$ perf script --dlfilter=filter.so
```


Tips for Overcoming Common Challenges

Using perf with Intel PT for Control Flow Analysis

%RIP Filter: limit tracing to specific areas

- `perf record -e intel_pt//u --filter 'start 0x1149 @ foo' -- foo`
- `perf record -e intel_pt//u --filter 'filter main @ foo' -- foo`

Disable Return Compression: use `/noretcomp/` to trace all "ret" instructions, helpful for obfuscated code

Limit Recording: restrict to specific processes, CPUs or use snapshot mode to manage data size

Adjust Cycle Threshold: increase `cyc_thresh` if cycle accuracy isn't critical to reduce data

PTWRITE for Custom Markers: add PTWRITE instructions to tag and track key events in the trace

Use PEBS: combine with PEBS (if supported) to capture memory access patterns alongside control flow. Technically works, but combining tracing and sampling is somehow awkward

Tips for Overcoming Common Challenges - II

Using perf with Intel PT for Control Flow Analysis

Decoding is intensive: decoding traces can produce significant output files and decoding time

```
$ perf record -a -e intel_pt// -o perf.data -- sleep 0.01 (10ms on 32 core system)
$ perf script --itrace=i0ns --ns > report.txt
$ ls -lh report.txt perf.data
-rw----- 1 pfeifer pfeifer 2.7M Nov  2 14:17 perf.data
-rw-rw-r-- 1 pfeifer pfeifer 862M Nov  2 14:18 report.txt  (--> ~330)
```

Limit Decoding Timeframe: use `--time <start>,<stop>` to reduce both data size and decoding time by focusing on specific sections

Exclude Unneeded Fields: Use: `perf script --call-trace -F -cpu,-tid,-time` to skip unnecessary fields, reducing report size and processing load

Mitigate Record Overloads: increase buffer size, limit recording, switch recording setup

Limitations of Intel PT in Control Flow Analysis

Conclusions

Limited Instruction Visibility: Intel PT captures only metadata (branches, timestamps) rather than actual instructions. For JIT-compiled code, self-modifying applications or loaded shellcode the decoder will fail as it relies on the available ELF objects

No Data Capture: captures instruction flow but lacks functionality to track data flow or data manipulation and transformations

High Data Volume: Intel PT generates vast amounts of data, making it essential to use filtering or snapshot mode to manage size and processing time

Platform Limitations:

- **ARM Alternative:** Intel PT's counterpart on ARM is CoreSight ETM, which has different configurations and features
- **No AMD Equivalent:** AMD lacks an equivalent tracing tool; → exec on Intel(R)

Thank you very much!

Let's Tackle Your Questions!

Got more questions? Feel free to catch me at the event or email me!
hagen@jauu.net

Correlate Dynamically Mapped DSO

Appendix

Linux will map DSO pseudo-randomly, see ELF and Address Space Layout Randomization

- Addresses like 0x5583ca2fa2a1 or 7f4180447792 becoming meaningless

Luckely: perf will record mmap events:

```
$ perf script --no-itrace --show-mmap-events
cms 108632 [023] 124501.882184: PERF_RECORD_MMAP2 108632/108632: [0x5583ca2fa000(0x1000) @ 0x1000 103:02
25874047 2058819998]: r-xp cms
cms 108632 [023] 124501.882202: PERF_RECORD_MMAP2 108632/108632: [0x7f4180604000(0x27000) @ 0x1000 103:02
27829660 3107761507]: r-xp /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
cms 108632 [023] 124501.882211: PERF_RECORD_MMAP2 108632/108632: [0x7f4180601000(0x2000) @ 0 00:00 0 0]:
r-xp [vdso]
cms 108632 [023] 124501.882297: PERF_RECORD_MMAP2 108632/108632: [0x7f4180419000(0x15a000) @ 0x28000 103:02
27829672 2718021217]: r-xp /usr/lib/x86_64-linux-gnu/libc.so.6
```

$\text{file-offset} = [\text{address}] - \text{start} + \text{offset}$

$4769 = 0x5583ca2fa2a1 - 0x5583ca2fa000 + 0x1000$

Copyright © 2024 Hagen Paul Pfeifer

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Names and brands may be claimed as the property of others. Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

All statements here are made in a personal capacity and are not affiliated with my employer.

This presentation was created with the highest quality standards, but errors may still be present. No warranties are made for any damages resulting from text or code examples.

License: This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0). Redistribution is encouraged! Not for use in AI/LLM training or commercial applications