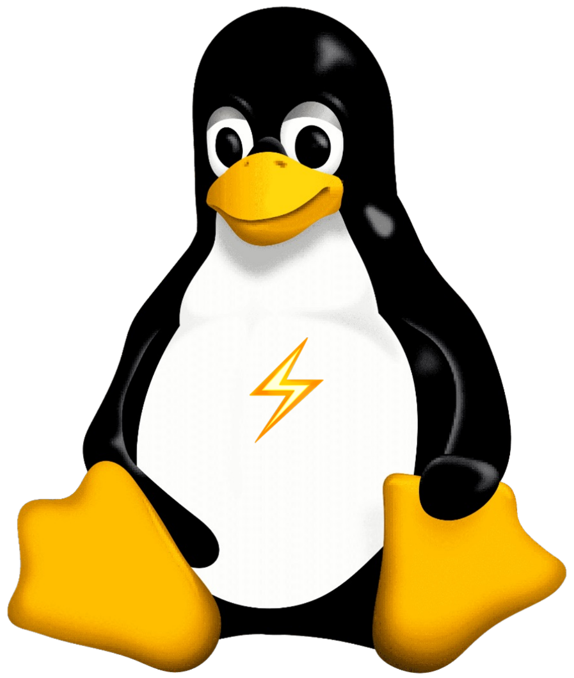
Optimizing the Linux Kernel with LLVM BOLT

Maksim Panchenko



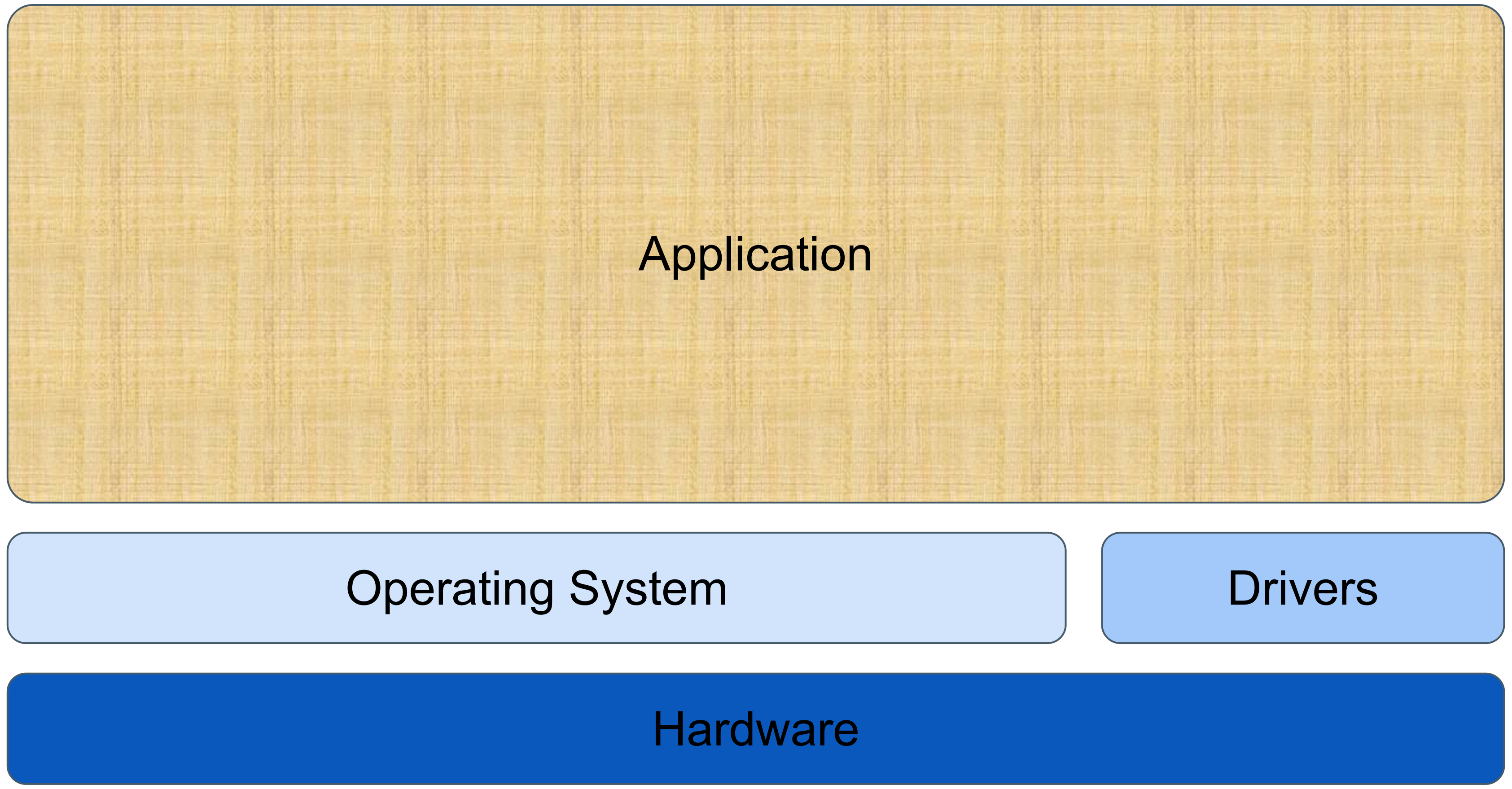
2023 EuroLLVM Developers' Meeting

Meta

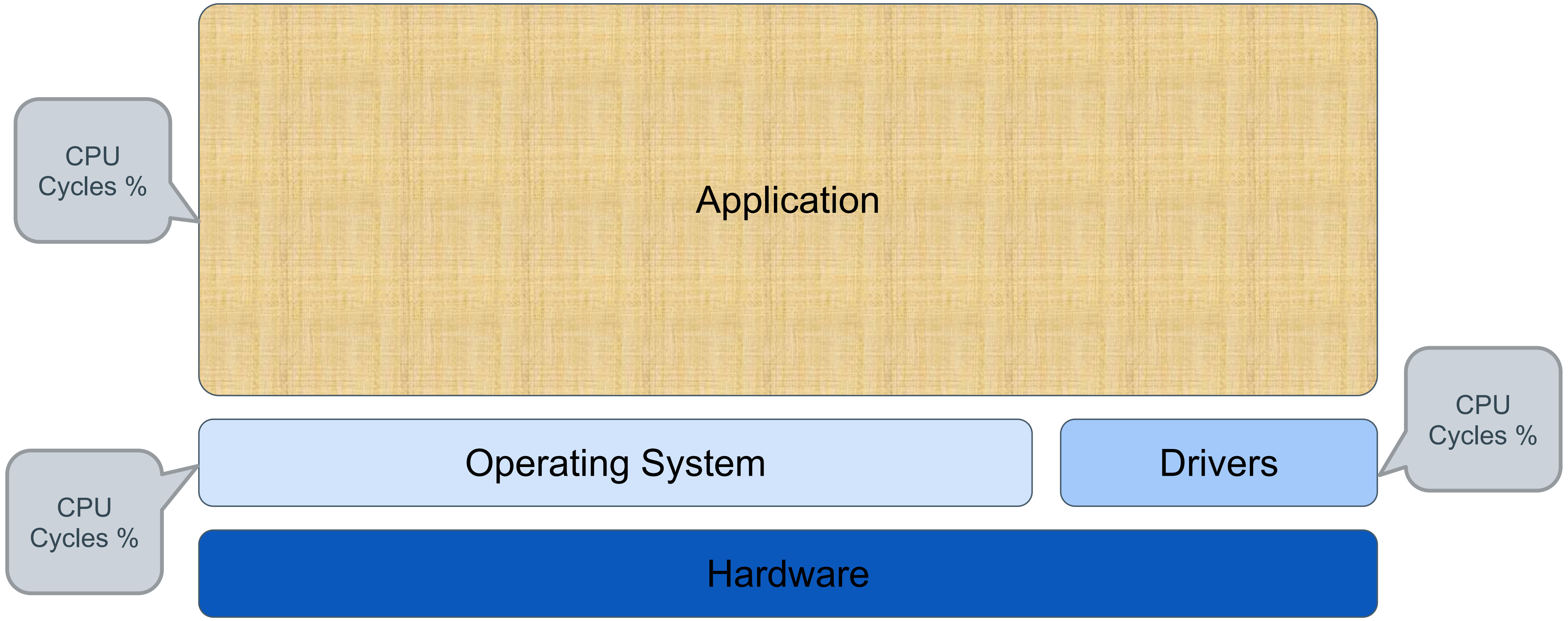
Agenda

- Why Optimize the Kernel?
- Challenges Applying BOLT
- Progress & Plans

Why Optimize the Linux Kernel?



Why Optimize the Linux Kernel?



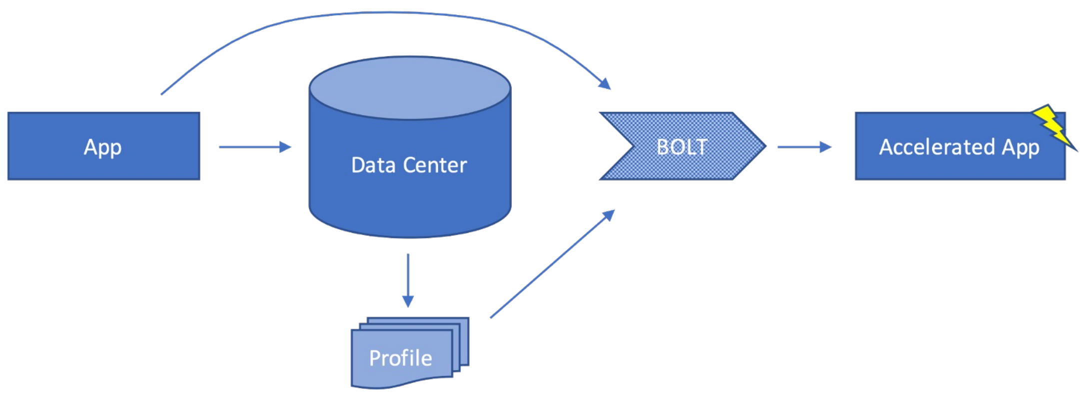
Why Optimize the Linux Kernel?

- Often optimized for size
- Heavily hand-tuned for performance
  - Plenty of assembly code
- Slow PGO adoption
- Even slower LTO adoption

BOLT Overview

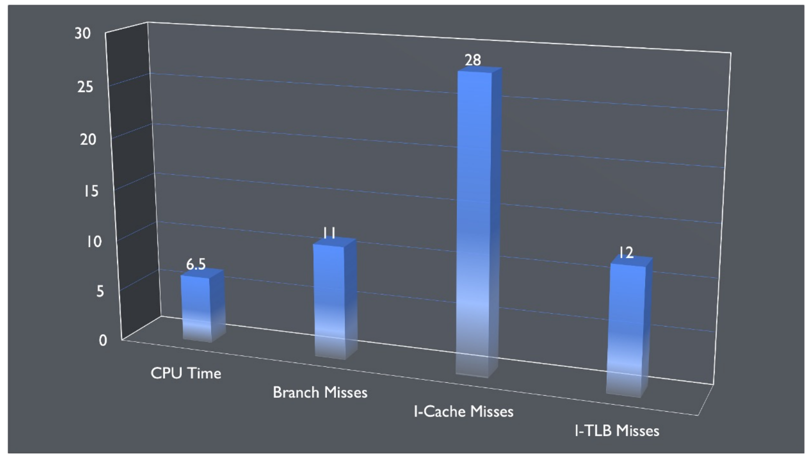
What?

Binary Optimization & Layout Tool



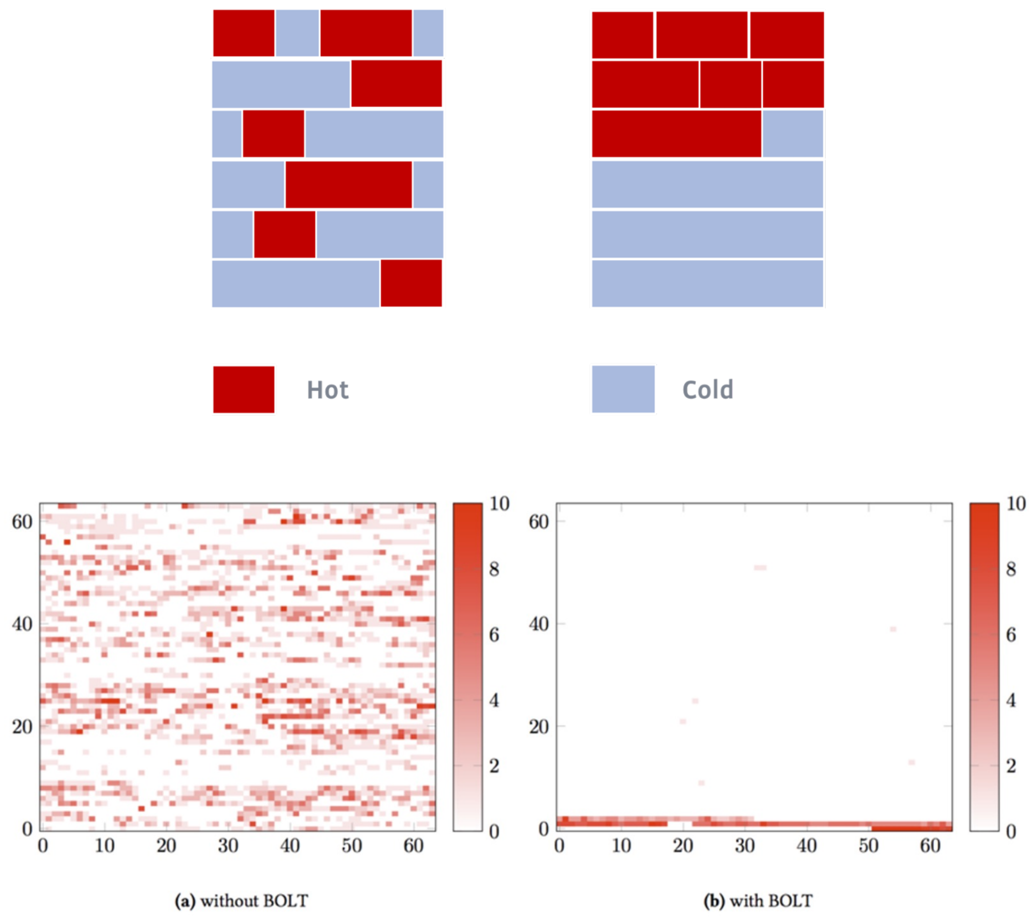
Why?

Accurate Binary Profile



How?

Break Binary and Assemble a Better One



Opportunities for BOLT

- Focusing on X86-64 for now
- 16 MiB *.text*
- High I$ miss ratio
- Exposure to Assembly
- Indirect Call Promotion
  - To offset retpoline penalties
- Specialized Kernel?

```
 24907705 : executed forward branches (+4.4%)
  1830988 : taken forward branches (-77.8%)
  6954275 : executed backward branches (-13.2%)
  3594639 : taken backward branches (-21.0%)
  1049173 : executed unconditional branches (-46.6%)
 15478597 : all function calls (-5.8%)
        0 : indirect calls (=)
        0 : PLT calls (=)
275956063 : executed instructions (-0.7%)
 74095406 : executed load instructions (=)
 32940929 : executed store instructions (=)
        0 : taken jump table branches (=)
        0 : taken unknown indirect branches (=)
 32911153 : total branches (-2.7%)
  6474800 : taken branches (-56.1%)
 26436353 : non-taken conditional branches (+38.5%)
  5425627 : taken conditional branches (-57.5%)
 31861980 : all conditional branches (-0.0%)
```

Challenges Applying BOLT

- Code Volatility
- Updating ELF
- Testing
- Debugging

Dealing with Code Volatility

- What You See Is *NOT* What You Get
- Simple disassembly does not reflect the state of the code while running the kernel
- Code is patched at boot time
  - SMP (lock prefix) vs single core (NOP)
  - Arch-specific instruction sequences
- Code is patched at run time
  - Ftrace
  - Static Keys
  - Static Calls

Static Keys

```
.Ltmp1319 (7 instructions, align : 1)
  Predecessors: .Ltmp1317
      00000123:   movq    %rbp, %rdx
      00000126:   movl    $0xc0000100, %ecx
      0000012b:   movl    %ebp, %eax
      0000012d:   shrq    $0x20, %rdx
      00000131:   wrmsr
      00000133:   nop  # Size: 5
      00000138:   jmp     .Ltmp1320
 Successors: .Ltmp1320 (mispreds: 0, count: 2)
```

Static Keys

```
.Ltmp1319 (7 instructions, align : 1)
  Predecessors: .Ltmp1317
      00000123:   movq    %rbp, %rdx
      00000126:   movl    $0xc0000100, %ecx
      0000012b:   movl    %ebp, %eax
      0000012d:   shrq    $0x20, %rdx
      00000131:   wrmsr
      00000133:   nop  # Size: 5
      00000138:   jmp     .Ltmp1320
 Successors: .Ltmp1320 (mispreds: 0, count: 2)
```

LBR Branch
Start ???

Static Keys

```
.Ltmp1319 (7 instructions, align : 1)
  Predecessors: .Ltmp1317
      00000123:   movq    %rbp, %rdx
      00000126:   movl    $0xc0000100, %ecx
      0000012b:   movl    %ebp, %eax
      0000012d:   shrq    $0x20, %rdx
      00000131:   wrmsr
      00000133:   jmp     .Ltmp46490 # STATICKEY
      00000138:   jmp     .Ltmp1320
 Successors: ?
```

LBR Branch Start

Static Keys

- Eliminates global variable condition check
  - More than removing a branch
- Assembler macro: `STATIC_JUMP_IF_{TRUE|FALSE} target, key`
- Metadata placed in *.rodata*
- __jump_table "section" is marked by [*__start___jump_table, __stop___jump_table*)
- Entry Contents:

```
PCREL32 JumpAddress
PCREL32 TargetAddress
PCREL64 KeyAddress
```

```
/* Documentation/staging/static-keys.rst */
if (static_branch_unlikely(&key))
    printk("I am the true branch\n");
```

Static Keys Support

```
.Ltmp1319 (7 instructions, align : 1)
  Predecessors: .Ltmp1317
      00000123:   movq    %rbp, %rdx
      00000126:   movl    $0xc0000100, %ecx
      0000012b:   movl    %ebp, %eax
      0000012d:   shrq    $0x20, %rdx
      00000131:   wrmsr
      00000133:   nop OR jmp     .Ltmp46490
      00000138:   jmp     .Ltmp1320
 Successors: ?
```

Static Keys Support

```
.Ltmp1319 (7 instructions, align : 1)
  Predecessors: .Ltmp1317
      00000123:   movq    %rbp, %rdx
      00000126:   movl    $0xc0000100, %ecx
      0000012b:   movl    %ebp, %eax
      0000012d:   shrq    $0x20, %rdx
      00000131:   wrmsr
      00000133:   jCC     .Ltmp46490
      00000138:   jmp     .Ltmp1320
Successors: .Ltmp46490 (mispreds: 0, count: 42), .Ltmp1320 (mispr: 0, count: 2)
```

Static Keys Support

```
.Ltmp1319 (7 instructions, align : 1)
  Predecessors: .Ltmp1317
      00000123:   movq    %rbp, %rdx
      00000126:   movl    $0xc0000100, %ecx
      0000012b:   movl    %ebp, %eax
      0000012d:   shrq    $0x20, %rdx
      00000131:   wrmsr
      00000133:   jit     .Ltmp46490 # STATICKEY: 0 # Size: 5
      00000138:   jmp     .Ltmp1320
Successors: .Ltmp46490 (mispreds: 0, count: 42), .Ltmp1320 (mispr: 0, count: 2)
```

Static Keys Support Implementation

- Disassemble Functions
- Read Static Keys ”Jump Table”
- Convert JMP/NOP into JCC with a custom CC (it)
- Build CFG

  …
- Convert JIT into JMP/NOP
  - Preserve the original 5-byte size
- Emit function code
- Update “Jump Table” with new Jump and Target addresses
- Update Key Address low bit if the condition was reversed

Static Keys Opportunities

- ~20% of hot functions have static key jumps
- Always 5-byte NOP/JMP
  - 2-byte might work in many cases
  - BOLT can detect short vs long JMP codegen
- Invalid static prediction
  - Reverse the key condition

```
queued_spin_lock_slowpath:
.LBB0815 (2 instructions, align : 1)
Exec Count : 13836
  00000000:   callq   __fentry__
  00000005:   jit     .Ltmp46692 # STATICKEY: 0 # Size: 5
Successors: .Ltmp46692 (miss: 1, count: 13836), .LFT3516 (m: 0, count: 0)
```

Static Calls

- Way to overcome overheads of indirect calls
  - Critical for kernels hardened with retpolines
- CALL to __SCT__tp_func_* in disassembly
  - Replaced with new target or NOP
- LBR confusion
  - Might affect function ordering

```
tools/include/linux/static_call_types.h
  /*
   * The static call site table needs to be created by external tooling (objtool
   * or a compiler plugin).
   */
  struct static_call_site {
    s32 addr;
    s32 key;
  };
```

Alternative Instructions

- *ALTERNATIVE oldinstr, newinstr, feature*
- Padding required
- NOPs are optimized by *optimize_nops()*
- Jumps and calls are recognized and fixed
  - By manually checking the first byte/opcode

```
# Emit oldinstr to current section
L1:
  oldinstr
  < pad with nops if newinstr is larger>

# Emit newinstr to .altinstr_replacement
L2:
  newinstr

# Emit description to .altinstructions
  .long L1 - .
  .long L2 - .
  .word \feature
  .byte \old_len # with padding
  .byte \new_len
  .byte \pad_len
```

Alternative Instructions

- *ALTERNATIVE_2 oldinstr, newinstr1, feature1, newinstr2, feature2*
  - Generates 2 entries for the same oldinstr
  - E.g. one instructions for Intel another for AMD

Alternative Instructions Support

- Optimize conservatively
  - Preserve padding in main code
  - Update alternative instruction targets in *.altinstr_replacement*
- Ignore LBR discrepancies in “alternative regions”
- Advanced: hint BOLT what features are enabled to optimize alternative sequences
- Not as common as e.g. static keys

More Code Variants

- `.parainstructions`
- `__fentry__`
  - LBR profile discrepancy
- `.smp_locks`
  - Locations of SMP lock prefixes
- New/Undiscoverd Sections
  - *vmlinux* linked with relocations
  - Ignore functions with unknown references in the middle of code

More Sections to Update

- ORC
  - `.orc_unwind_ip`, `.orc_unwind`, `.orc_lookup`
- `__ex_table`
- `__bug_table`
- `.pci_fixup`
- `__ksymtab{_gpl}`

Updating ELF

- Incremental Rewriting
- No HFSort / Function Reordering
- In-place mode with relocations
  - Or work w/o relocations with caveats
- No new PHDR / Segment
- No function splitting

Debugging

- Easier to debug in a VM
  - Turnaround time few seconds vs several minutes on HW
  - Some issues only come up on HW
    - Usage of modules/drivers
- Kernel panic
  - Normally easier to debug due to included stack trace
- Kernel stuck at boot time
- Bisecting in BOLT
  - Limit optimizations to a set of functions
  - Works well with VM turnaround time

Kernel Panic

```
[   21.406279] Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)
[   21.406292] random: fast init done
[   21.422779] CPU: 7 PID: 1 Comm: swapper/0 Not tainted 5.12.0-04819-ge63cdf46ae90-dirty #6
[   21.445902] Hardware name: Wiwynn  Tioga Pass Single Side /Tioga Pass Single Side, BIOS TPM10 05/20/2020
[   21.464829] Call Trace:
[   21.469720]  dump_stack+0x64/0x7c
[   21.476339]  panic+0xfb/0x2cb
[   21.482267]  mount_block_root+0x2aa/0x332
[   21.490276]  ? rdinit_setup+0x2c/0x2c
[   21.497589]  prepare_namespace+0x135/0x164
[   21.505770]  kernel_init_freeable+0x21f/0x22c
[   21.514472]  ? rest_init+0xb4/0xb4
[   21.521268]  kernel_init+0xa/0x10c
[   21.528062]  ret_from_fork+0x22/0x30
[   21.535337] Kernel Offset: disabled
[   21.600307] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0) ]---
```

Stuck Kernel

```
[   21.944465] Run /init as init process
[   58.920048] random: fast init done
[  110.971490] random: crng init done
```

- Broken *vmlinux* had the exact loadable contents as “good” *vmlinux*
- *.vvar* section header was truncated
  - was followed by *.data..percpu* section that **starts at address 0** in the middle of data segment

```
[21] .vvar            PROGBITS        ffffffff83b7a000 2d7a000 001000 00  WA  0   0 16
[22] .data..percpu    PROGBITS        0000000000000000 2e00000 02d000 00  WA  0   0 4096
[23] .init.text       PROGBITS        ffffffff83ba8000 2fa8000 056ba8 00  AX  0   0 16
[24] .altinstr_aux    PROGBITS        ffffffff83bfeba8 2ffeba8 000be4 00  AX  0   0  1
```

Progress

- Milestone 1: booting VM
- Milestone 2: booting 5.12 on HW
  - *vmlinux/bzImage* is not enough
    - *initramfs* contains kernel modules necessary to mount the real rootfs
    - Kernel modules should be compatible with *vmlinux*
    - (Re-)build the kernel package
  - Regular and “hardened” flavors
- Current (May ‘23): start prod testing & performance measurement

Plans

- Find a good open-source benchmark/suite
  - Stress I$/iTLB
  - Microbenchmark might not work
  - Server-type or DB workloads seem to fit the bill
    - Rocksdb/LevelDB/Apache/Nginx/MySQL/PostgreSQL (*)
  - Achieve decent SNR
- Gradually increase code coverage
- Full binary rewrite with 100% coverage

(*) *One Profile Fits All: Profile-Guided Linux Kernel Optimizations for Data Center Applications,* M. Ugur, et al., ACM SIGOPS Operating System Review, 2022

Meta