Enable Intel Control Flow Enforcement Technology

Chen Hu <hu1.chen@intel.com>

Intel System Software Engineering

Oct. 2022

Agenda

- Introduction
  - Shadow Stack (SS)
  - Indirect Branch Tracking (IBT)
- SW stack to enable CET
  - App Execution Flow
  - Kernel Implementation and challenges
- User Space CET Reference Design
- CET Perf Impact
- Q & A

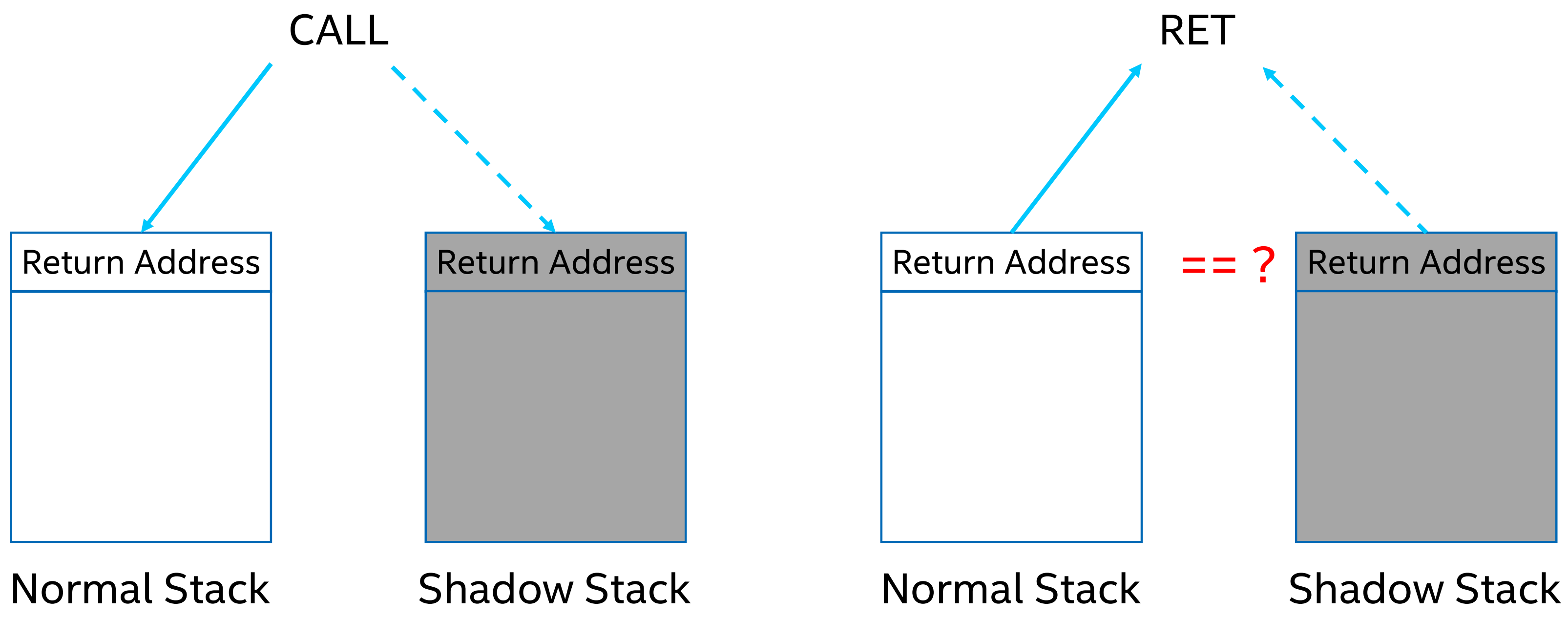
Introduction – Control Flow

Return/jump oriented programming (ROP) attack here. ☹

Intel Control Flow Enforcement (CET) mitigates this. ☺

Control Flow	Mitigation
CALL / RET	Shadow Stack
JMP *%rax	Indirect Branch Tracking (IBT)

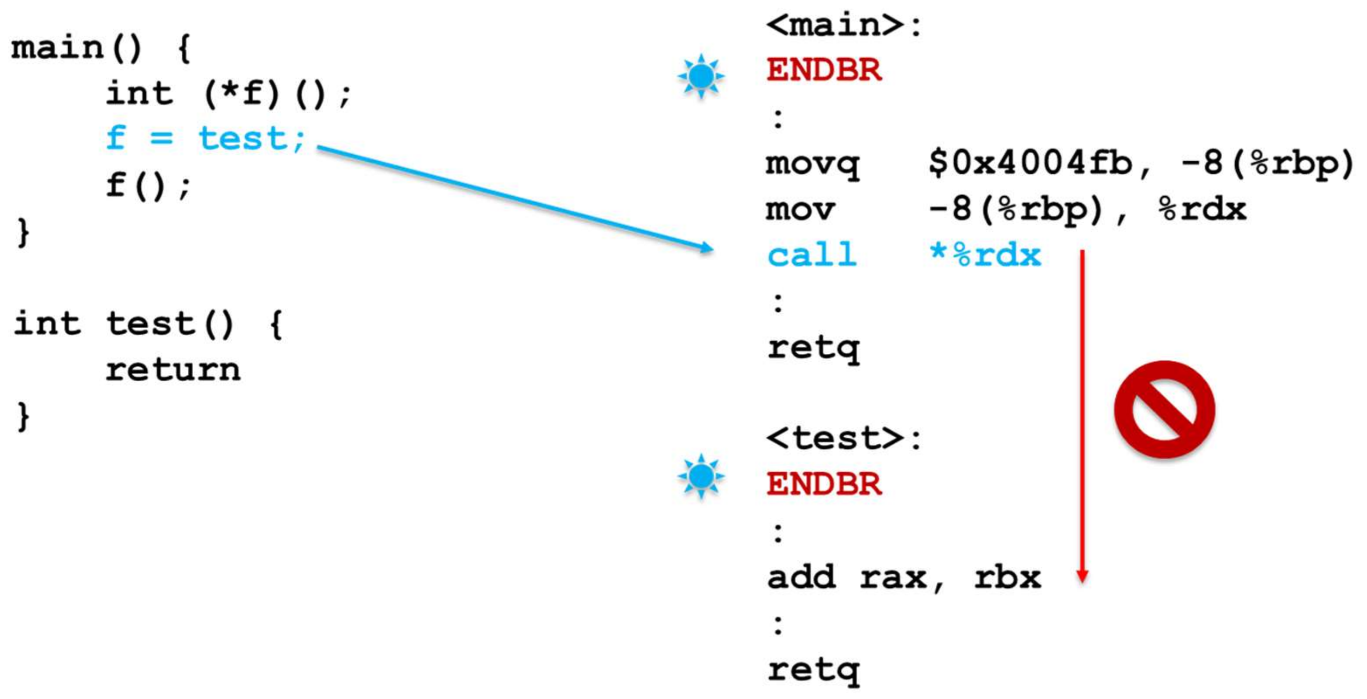
CET – Shadow Stack



- Return addresses popped off both stacks should match or suffer #CP

Credit: hongjiu.lu@intel.com https://lpc.events/event/2/contributions/147/attachments/72/83/CET-LPC-2018.pdf

CET – Indirect Branch Tracking



```
main() {
    int (*f)();
    f = test;
    f();
}

int test() {
    return
}
```

```
<main>:
ENDBR
:
movq    $0x4004fb, -8(%rbp)
mov     -8(%rbp), %rdx
call    *%rdx
:
retq

<test>:
ENDBR
:
add rax, rbx
:
retq
```

- Any indirect CALL/JMP must target an ENDBR instruction or suffer #CP
- ENDBR is NOP on non-CET processor

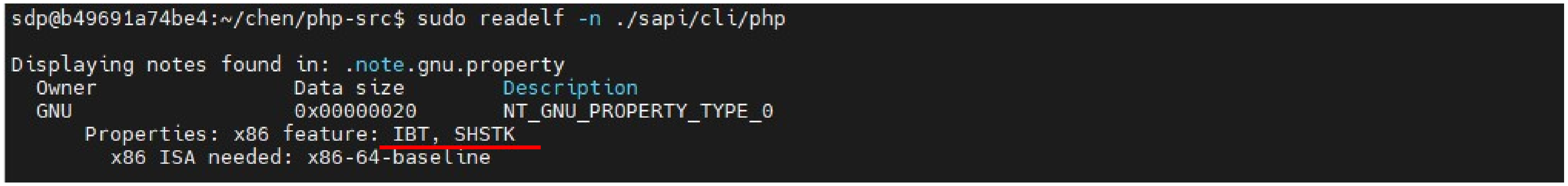
SW Stack to Enable CET

Compile	Run	
Compiler	Application	
Linker	Libc	
	Kernel	IBT for kernel
		IBT for user
		SS for kernel
		SS for user

More complex than imagined!

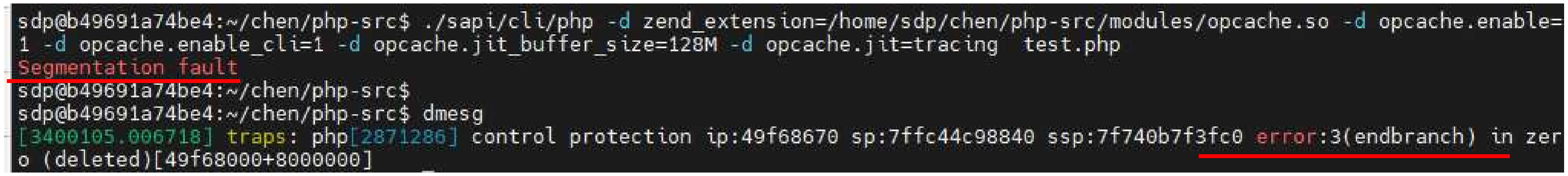
CET violation example

- Compile with "*-fcf-protection=full/branch/ret/none*"
- ELF header indicates if CET enabled



```
sdp@b49691a74be4:~/chen/php-src$ sudo readelf -n ./sapi/cli/php

Displaying notes found in: .note.gnu.property
  Owner                 Data size        Description
  GNU                   0x00000020       NT_GNU_PROPERTY_TYPE_0
      Properties: x86 feature: IBT, SHSTK
        x86 ISA needed: x86-64-baseline
```

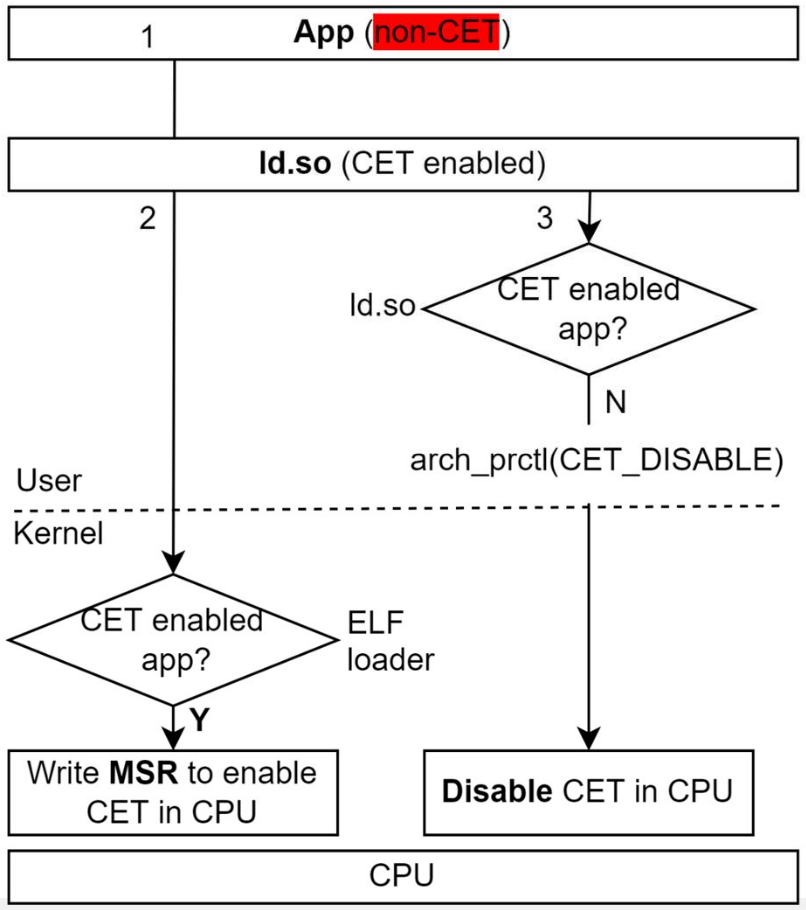
- Suffer #CP while CET is violated



```
sdp@b49691a74be4:~/chen/php-src$ ./sapi/cli/php -d zend_extension=/home/sdp/chen/php-src/modules/opcache.so -d opcache.enable=
1 -d opcache.enable_cli=1 -d opcache.jit_buffer_size=128M -d opcache.jit=tracing  test.php
Segmentation fault
sdp@b49691a74be4:~/chen/php-src$
sdp@b49691a74be4:~/chen/php-src$ dmesg
[3400105.006718] traps: php[2871286] control protection ip:49f68670 sp:7ffc44c98840 ssp:7f740b7f3fc0 error:3(endbranch) in zer
o (deleted)[49f68000+8000000]
```

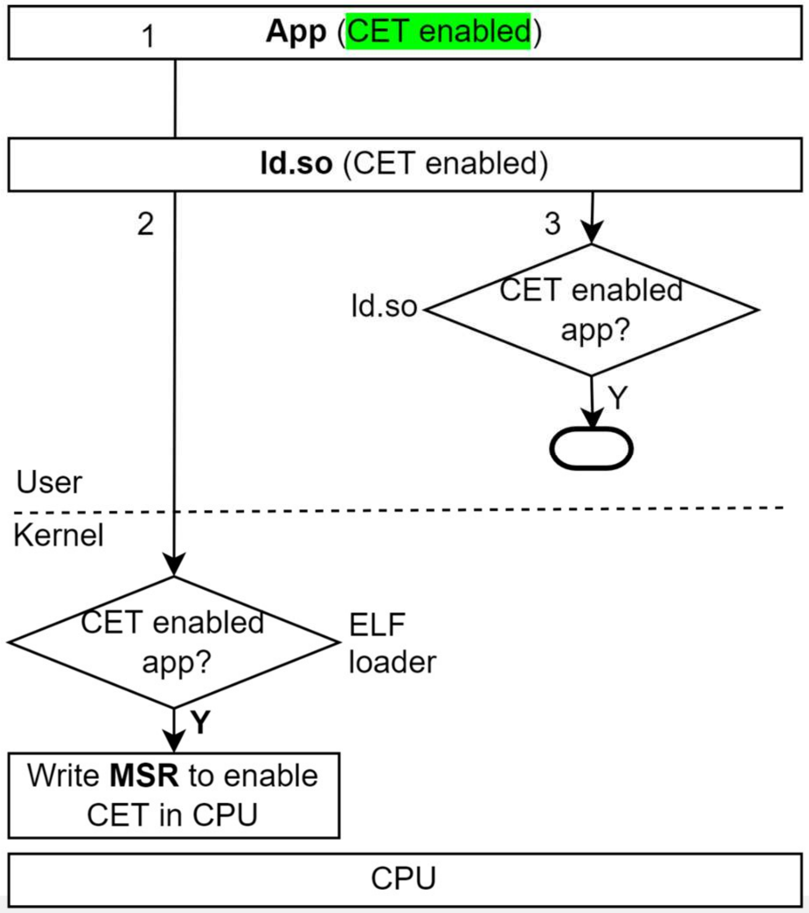
Ubuntu22.04 on Intel Sapphire Rapid with 5.15 kernel

Workflow of non-CET App



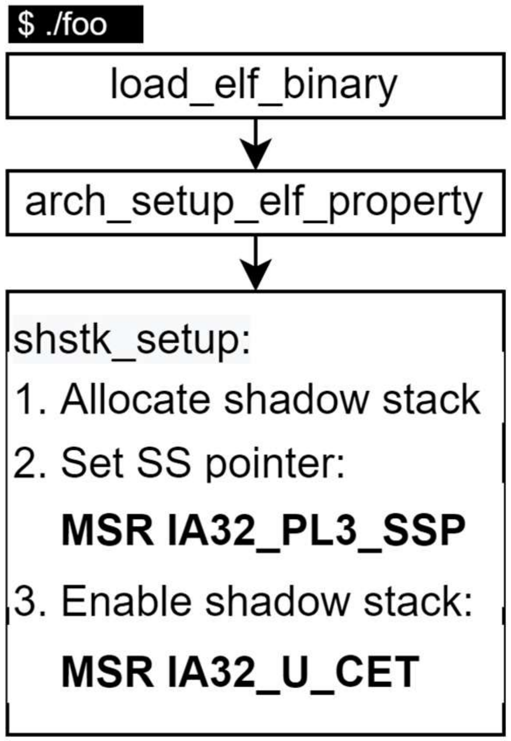
1. ELF header of app indicates non-CET.
2. Since ld.so is CET enabled, all apps start as CET enabled. Kernel set MSR to enable CET.
3. ld.so find that app is non-CET. Issue syscall to disable CET.

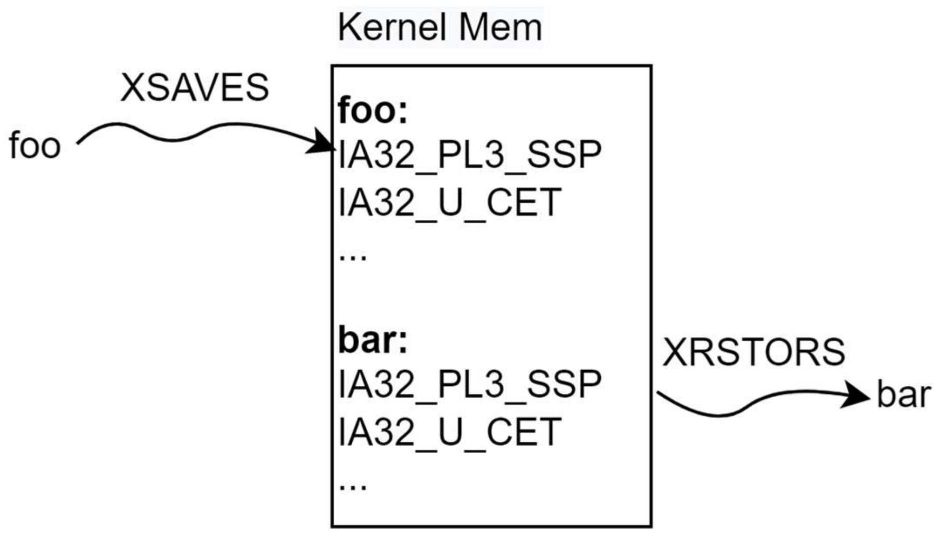
Workflow of CET Enabled App



1. ELF header of app indicates CET enabled.
2. Since ld.so is CET enabled, all apps start as CET enabled. Kernel set MSR to enable CET.
3. ld.so find that app is CET enabled. Do nothing.

Kernel Implementation – Shadow Stack





- Create process

- Context Switch

Kernel Implementation – Challenges

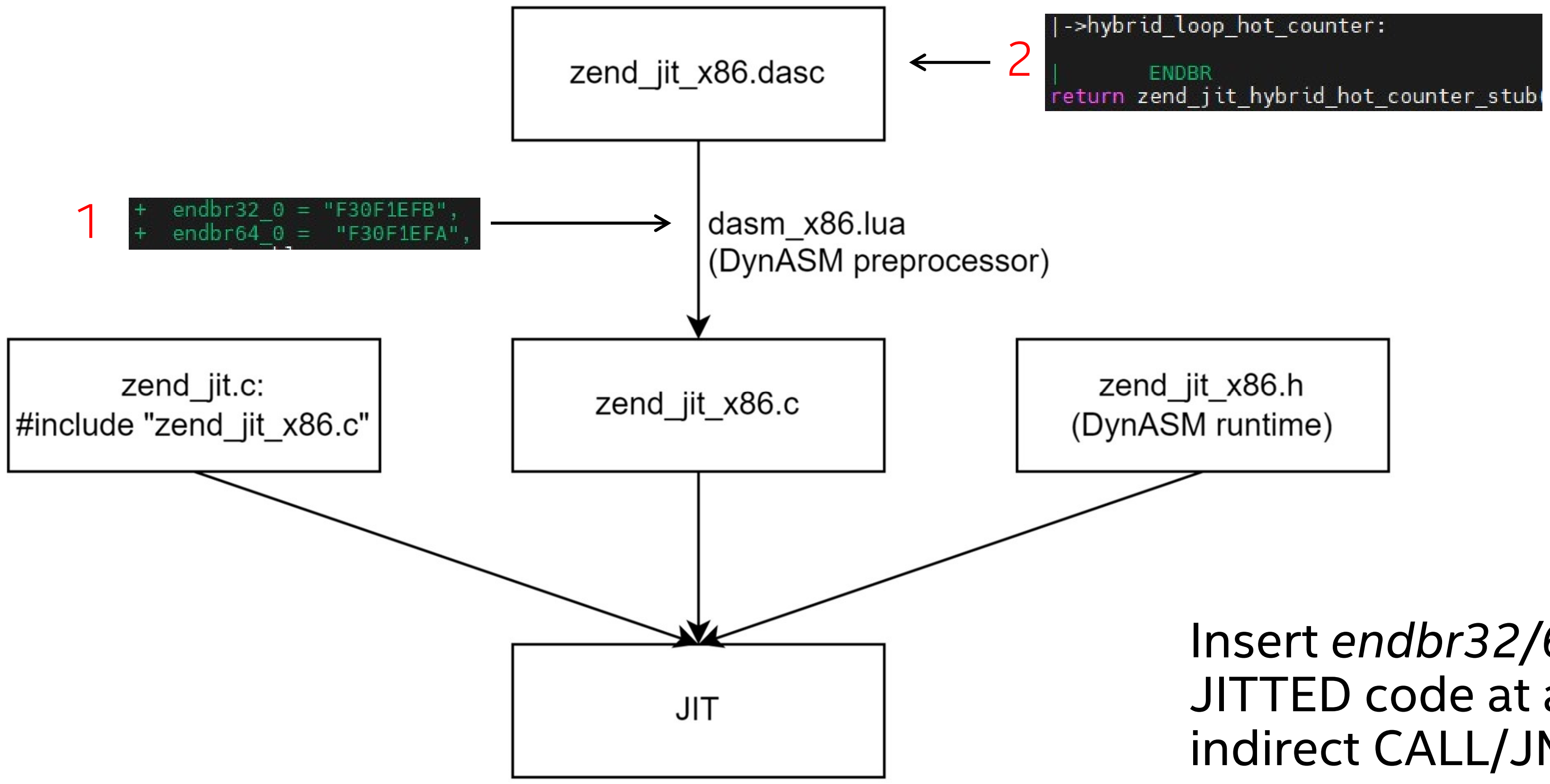
1. Shadow stack has “write=0, dirty=1 PTEs”
2. Signal
3. ABI issues
4. CRIU (checkpoint and restore in user space) support

Status

Compiler	GCC 8	Application		See Ref
Linker	Binutils 2.31	Libc		Glibc 2.28
		Kernel	IBT for kernel	5.18
			IBT for user	Rev 30+
			SS for kernel	N/A
			SS for user[1]	Rev 30+

1. https://lkml.org/lkml/2022/9/29/1149

User Space CET Ref Design: PhP JIT



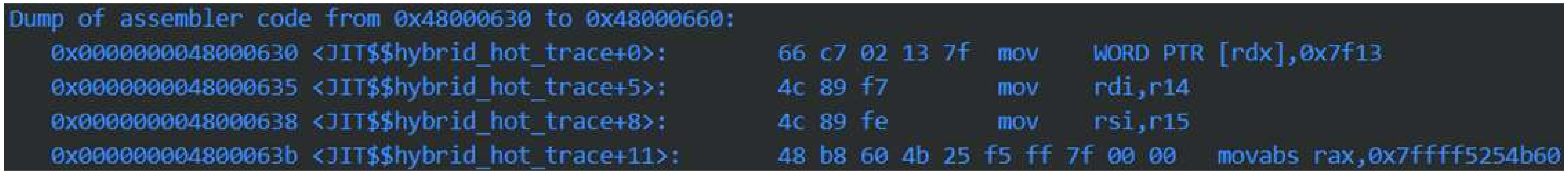
```
|->hybrid_loop_hot_counter:

|       ENDBR
return zend_jit_hybrid_hot_counter_stub
```

```
+   endbr32_0 = "F30F1EFB",
+   endbr64_0 =  "F30F1EFA",
```

Insert *endbr32/64* in JITTED code at any indirect CALL/JMP target

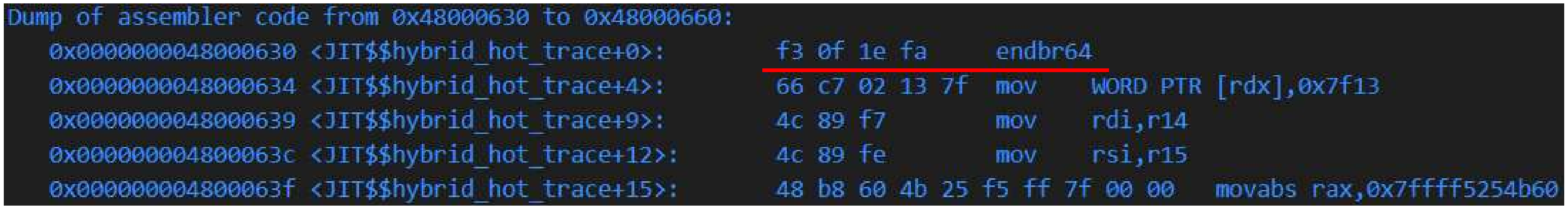
User Space CET Ref Design: PhP JIT

Php:



```
Dump of assembler code from 0x48000630 to 0x48000660:
   0x0000000048000630 <JIT$$hybrid_hot_trace+0>:        66 c7 02 13 7f  mov    WORD PTR [rdx],0x7f13
   0x0000000048000635 <JIT$$hybrid_hot_trace+5>:        4c 89 f7        mov    rdi,r14
   0x0000000048000638 <JIT$$hybrid_hot_trace+8>:        4c 89 fe        mov    rsi,r15
   0x000000004800063b <JIT$$hybrid_hot_trace+11>:       48 b8 60 4b 25 f5 ff 7f 00 00   movabs rax,0x7ffff5254b60
```

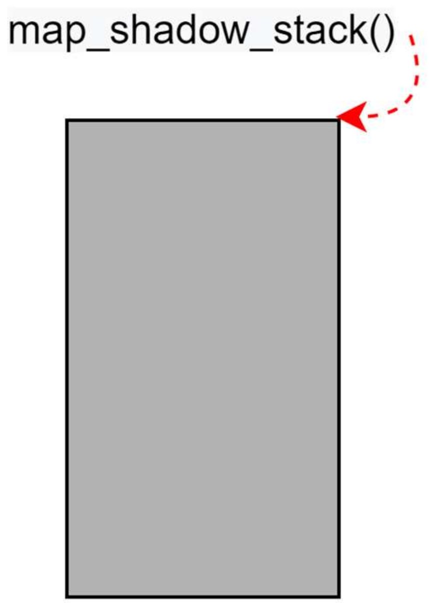
Php: Enable CET-IBT for JIT:



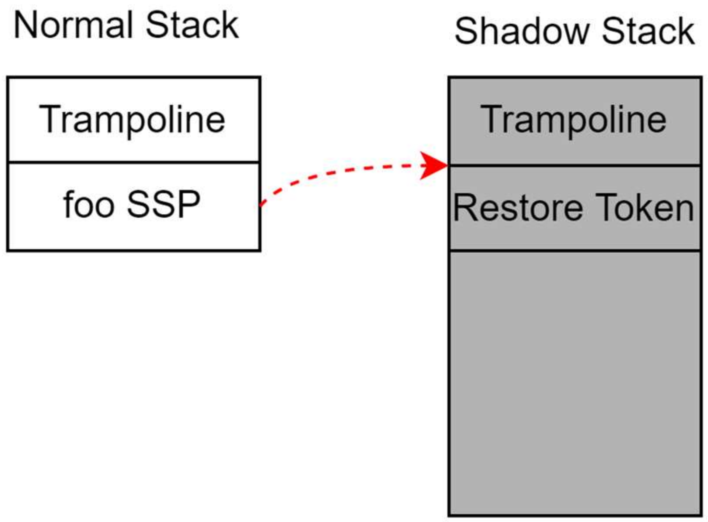
```
Dump of assembler code from 0x48000630 to 0x48000660:
   0x0000000048000630 <JIT$$hybrid_hot_trace+0>:        f3 0f 1e fa     endbr64
   0x0000000048000634 <JIT$$hybrid_hot_trace+4>:        66 c7 02 13 7f  mov    WORD PTR [rdx],0x7f13
   0x0000000048000639 <JIT$$hybrid_hot_trace+9>:        4c 89 f7        mov    rdi,r14
   0x000000004800063c <JIT$$hybrid_hot_trace+12>:       4c 89 fe        mov    rsi,r15
   0x000000004800063f <JIT$$hybrid_hot_trace+15>:       48 b8 60 4b 25 f5 ff 7f 00 00   movabs rax,0x7ffff5254b60
```

https://github.com/php/php-src/pull/8774

User Space CET Ref Design: CoRoutine



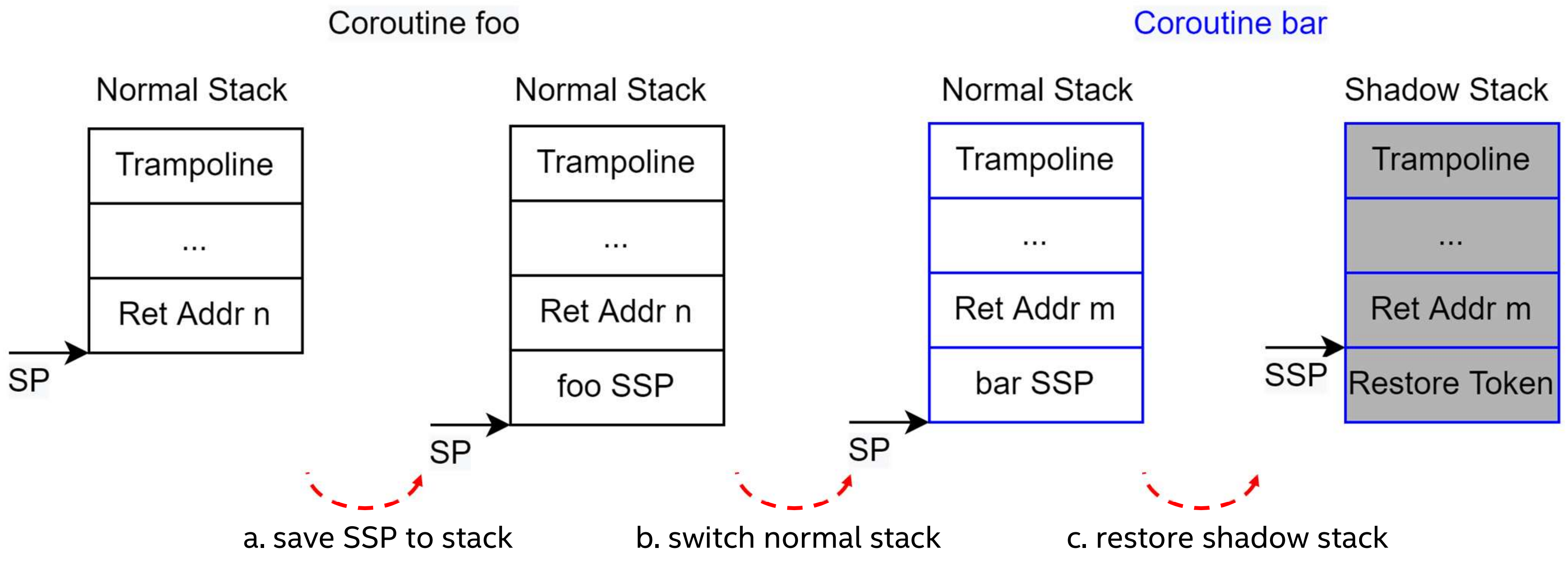
1. Allocate shadow stack for new coroutine.



2. Populate normal / shadow stack when create fcontext

https://github.com/boostorg/context/pull/207/

User Space CET Ref Design: CoRoutine



3. Context switch from coroutine foo to bar

CET Perf Impact

- IBT
  - More uops used
  - More instructions (endbr) in binary.
  - If CPU doesn’t support IBT or doesn’t enable IBT, endbr is nop.
- Shadow stack
  - More uops used
  - Two additional mem access (push/pop shadow stack) for each function call
  - Compare normal / shadow stack during return
- Speculation behavior (Refer to Intel SDM)

Acknowledgement

Srinivas Suresh, H.J. Lu, Rick Edgecombe, Yu-cheng Yu, Yuhan Yang

Q & A

intel®