Experiment of page table walking.

===

1. 4leval paging graph.

2. two flags (PSE,NX), (PSE means the page is huge page, NX means date don't have execution permission)

3. x86\_64 virtual address space, and kaslr

3. 2M huge page walking.

4. 4k page walking.

5. 4k kernel vmalloc walking.

6. page\_table\_offset is randomised by kaslr.

===

1. Page table structure for x86\_64 (4level)

http://developer.amd.com/wordpress/media/2012/10/24593\_APM\_v21.pdf

================== walk 4k page ================================

1. compile and start user prog that mmap memory and write data with

'0xbeafbeaf'. (pid and mmap returned address would be showed).

[.]# make

[.]# ./userpage

mmap: Success

debug: pid = 6799, user\_addr = 0x7f04abe76000, (\*user\_addr): beafbeafbeafbeaf

2. Use crash to get user task memory descriptor.

> task [your\_pid\_in\_step2] | grep mm -w

3. Get user global page table virtual address (pgd address)

> mm\_struct.pgd [address\_from\_step2]

4. Get page table offsets.

Use python >>

hex(address >> 39 & 0x1ff) --> offset in pgd

hex(address >> 30 & 0x1ff) --> offset in pud

hex(address >> 21 & 0x1ff) --> offset in pmd

hex(address >> 12 & 0x1ff) --> offset in pte

hex(address 0xfff) --> offset in page

5. Use crash and the user virtual address to walk the page table.

# get pud address

crash > p/x \*((long\*)pgd\_address + offset\_in\_pgd)

crash > pte <value from previous step>

crash > ptov <value from previous step>

# get pmd address

crash > p/x \*((long\*)pud\_address + offset\_in\_pud)

crash > pte <value from previous step>

crash > ptov <value from previous step>

# get pte address

crash > p/x \*((long\*)pmd\_address + offset\_in\_pmd)

crash > pte <value from previous step>

crash > ptov <value from previous step>

# get page address

crash > p/x \*((long\*)pte\_address + offset\_in\_pte)

crash > pte <value from previous step>

crash > ptov <value from previous step>

# get final address

crash > p/x page\_address + offset\_in\_page.

# print the value, should show 0xbeafbeafbeafbeaf

crash > rd final\_address

==== Walk 2M huge page ===============

1. reserve huge page of 12 pages.

# sysctl vm.nr\_hugepages=12

2. run the userhugepage programme, which map 12 pages of 2M and write with

0xbeafbeafbeafbeaf.

# ./userhugepage

3. Walk the returned virtual address from previoius step, with the printed pid.

4. when getting the entry in pmd, A 'pte' command should show PSE flag.

then you get the physycal address of the final user page.

5. get the virtual address of the page.

crash > ptov <address\_from\_step 4>

6. print the content ,should show 0xbeafbeafbeafbeaf

crash > rd address\_from\_step\_5

===== Use module to walk kernel/user address =====

1. insmod walk\_pg\_table.ko vaddr=0xyour\_address\_from\_mmap pid=your\_user\_pid

2. check dmesg

[ 1021.645983] debug: Try to walk page table for virtual address ffff9aa5809f8000

[ 1021.647293] debug: Find pgd table address for pid 1: ffff89d6f7c03000

[ 1021.647979] debug: \_\_PAGE\_OFFSET(physical addr direct mapping start): ffff89d5c0000000

[ 1021.648628] debug: pgd\_table = ffff89d6f7c03000

[ 1021.649094] debug: pgd\_index = 135

[ 1021.649294] debug: pud\_table = ffff89d6fb060000

[ 1021.649786] debug: pud\_index = 96

[ 1021.649977] debug: pmd\_table = ffff89d6fb061000

[ 1021.650446] debug: pmd\_index = 4

[ 1021.650655] debug: pte\_table = ffff89d6f8f96000

[ 1021.651119] debug: pte\_index = 1f8

[ 1021.651324] debug: page\_address\_pfn = 0000000000036846

[ 1021.651848] debug: page\_address = ffff89d5f6846000

[ 1021.652205] debug: offset = 0

[ 1021.652519] debug: content = faebdab