

# A few notes on debugging the buggy 'veth' network driver using 'crash'


## Stack : traditional view

Typically, the lowest stack content are the parameters passed to the function! (CPU ABI).

View by increasing addresses upwards:

```
[...          <-- 'Bottom'; higher (virtual) addresses
PARAMS
...]
RET addr
[ER]BP/SFP  <-- EBP (base ptr) OR SFP (pointer to previous stack frame [optional])


[...
LOCALS
...]          <-- 'Top' of the stack (ESP); lower (virtual) addresses
```



On the x86\_64, the first 6 parameters to a function are passed in the registers %rdi, %rsi, %rdx, %rcx, %r8, %r9 and Not via the stack.

View it the way we're used to, by increasing addresses downwards:


```
[...          <-- 'Top' of the stack (ESP); lower (virtual) addresses
LOCALS
...]
[ER]BP/SFP  <-- EBP (base ptr) OR SFP (pointer to previous stack frame
[optional])
RET addr
[...
PARAMS
...]          <-- 'Bottom'; higher (virtual) addresses
```



## Stack:

Src: [Extracting kernel stack function arguments from Linux x86-64 kernel crash dumps, Calum Mackay, Sept 2020](#)

```
...
# lower memory addresses
[%rsp]          top of the stack; new items are pushed on top of <<below>> this
...
...
[%rbp]          base of the stack; fixed. Contains the address of the
calling function's stack frame base.
<return address> contains the address of where this function should
return to, in the calling function
# higher memory addresses
...
```



## Test rig for kernel crash

1. Boot into custom debug kernel (with *kdump* enabled), an Ubuntu OSBoxes VM

The bootloader MUST pass the kernel parameter 'crashkernel=<...>'

Verify:

```
$ cat /proc/cmdline
```

```
BOOT_IMAGE=/vmlinuz-5.10.153-kdbg1 root=UUID=b67... ro quiet splash  
crashkernel=256M
```

Ok.

```
$ uname -r
```

```
5.10.153-kdbg1
```

2. Perform the 'kexec' load of the dump kernel into reserved RAM:

Within the [L5 debug trg](#) repo:

```
cd kernel_debug/kdumpcrash/kexec_load
```

```
$ ./kexec_load
```

```
kexec_load: Loading the 5.10.153-kdbg1 dump-capture kernel into reserved RAM
```

```
[sudo] password for osboxes:
```

```
kexec_load: kexec success
```

Ok.

3. Load the (buggy) driver

```
cd kernel_debug/to_debug_assgn/netdrv_veth_buggy/netdriver
```

```
cat run
```

```
#!/bin/bash
```

```
DRV=veth_netdrv
```

```
make && {
```

```
    sudo rmmod ${DRV}
```

```
    sudo dmesg -C
```

```
    sudo insmod ${DRV}.ko
```

```
    sudo dmesg
```

```
    journalctl -f -k
```

```
}
```

```
$ ./run
```

```
...
```

```
LD [M] /home/osboxes/kaiwanTECH/L5_debug_trg/kernel_debug/to_debug_assgn/netdrv_veth_buggy/netdriver/veth_netdrv.ko
make[1]: Leaving directory '/home/osboxes/linux-5.10.153'
[sudo] password for osboxes:
rmmod: ERROR: Module veth_netdrv is not currently loaded
[ 701.781889] veth_netdrv:vnet_init(): vnet: Initializing network driver...
[ 701.781953] veth_netdrv:vnet_probe(): vnet_probe:246 :
[ 701.782309] veth_netdrv:vnet_init(): loaded.
Journal file /var/log/journal/99e625cfc17348078889e61898a6a36c/system@0005f67312b55fde-447ef64c62cf1eaa.journal~ is truncated, ignoring file.
Jul 08 08:04:30 osboxes kernel: vboxvideo: loading version 6.1.38 r153438
Jul 08 08:04:32 osboxes kernel: e1000: enp0s8 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
Jul 08 08:04:32 osboxes kernel: IPv6: ADDRCONF(NETDEV_CHANGE): enp0s8: link becomes ready
Jul 08 08:04:32 osboxes kernel: rfkill: input handler disabled
Jul 08 08:04:32 osboxes kernel: process 'VBoxClient' launched '/usr/bin/VBoxDRMClient' with NULL argv: empty string added
Jul 08 08:16:07 osboxes kernel: veth_netdrv:vnet_init(): vnet: Initializing network driver...
Jul 08 08:16:07 osboxes kernel: veth_netdrv:vnet_probe(): vnet_probe:246 :
Jul 08 08:16:07 osboxes kernel: veth_netdrv:vnet_init(): loaded.
Jul 08 08:16:07 osboxes kernel: veth_netdrv:vnet_open(): vnet_open:191 :
Jul 08 08:16:07 osboxes kernel: veth_netdrv:vnet_start_xmit(): UDP pkt::src=68 dest=67 len=9473
Jul 08 08:16:07 osboxes kernel: x
Jul 08 08:16:07 osboxes kernel: xx
Jul 08 08:16:07 osboxes kernel: x
Jul 08 08:16:08 osboxes kernel: x
Jul 08 08:16:08 osboxes kernel: x
Jul 08 08:16:08 osboxes kernel: xx
[ ... ]
```

It's waiting to receive packet(s) from the user space 'sender' app...

#### 4. Run the user space sender app (triggering the bug!)

```
cd ../userspc
./runapp
... <sets up the 'veth' network intf, assigns it an IP, ...>
...
<transmits packet(s)>
...
```

The entire system abruptly and immediately hangs, then warm boots into the dump kernel!  
Only console mode..

Login.

```
ls -lh /proc/vmcore
-r----- 1 root root 1.8G Jul  8 08:27 /proc/vmcore
```

Copy in the kdump image (to disk or across n/w via scp):  
sudo cp /proc/vmcore kdump\_img

Reboot into regular kernel, retrieve the kdump image..

## Sample run with crash app on the kdump image obtained (after it crashed)

Key ref:

[Extracting kernel stack function arguments from Linux x86-64 kernel crash dumps, Calum Mackay, Sept 2020](#)

*Run the crash app on the kdump image file:*

A wrapper script runs

`sudo crash ./vmlinux-5.10.153-kdbg1 $1`

(so of course you MUST have the vmlinux with debug symbols matching version..).

```
$ ls -lh ~/kdump_img
-r----- 1 root root 1.8G Jul  8 08:29 /home/osboxes/kdump_img
$
$ ./crash_run
Usage: crash_run </path/to/kdump.img-OR-/proc/kcore>
$ ./crash_run ~/kdump_img

crash 8.0.0
Copyright (C) 2002-2021 Red Hat, Inc.
Copyright (C) 2004, 2005, 2006, 2010 IBM Corporation
Copyright (C) 1999-2006 Hewlett-Packard Co
Copyright (C) 2005, 2006, 2011, 2012 Fujitsu Limited
Copyright (C) 2006, 2007 VA Linux Systems Japan K.K.
Copyright (C) 2005, 2011, 2020-2021 NEC Corporation
Copyright (C) 1999, 2002, 2007 Silicon Graphics, Inc.
Copyright (C) 1999, 2000, 2001, 2002 Mission Critical Linux, Inc.
Copyright (C) 2015, 2021 VMware, Inc.
This program is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies of it under
certain conditions. Enter "help copying" to see the conditions.
This program has absolutely no warranty. Enter "help warranty" for details.

GNU gdb (GDB) 10.2
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...

      KERNEL: ./vmlinux-5.10.153-kdbg1 [TAINTED]
      DUMPFILE: /home/osboxes/kdump_img
      CPUS: 6
```

[ ... ]

TASKS: 563  
NODENAME: osboxes  
RELEASE: 5.10.153-kdbg1

VERSION: #1 SMP Tue Jun 13 15:51:43 IST 2023  
MACHINE: x86\_64 (2592 Mhz)  
MEMORY: 2 GB  
PANIC: "kernel BUG at mm/slab.c:305!"  
PID: 3610  
COMMAND: "talker\_dgram" << our user space 'sender' app! >>  
TASK: ffff8a61d098c740 [THREAD\_INFO: ffff8a61d098c740]  
CPU: 2  
STATE: TASK\_RUNNING (PANIC)

crash>

crash> bt

PID: 3610 TASK: ffff8a61d098c740 CPU: 2 COMMAND: "talker\_dgram"  
#0 [ffffa08ec21fb530] machine\_kexec at ffffffff8a5a80283  
#1 [ffffa08ec21fb590] \_\_crash\_kexec at ffffffff8a5b7f762  
#2 [ffffa08ec21fb660] crash\_kexec at ffffffff8a5b80f3d  
#3 [ffffa08ec21fb678] oops\_end at ffffffff8a5a3cb96  
#4 [ffffa08ec21fb6a0] die at ffffffff8a5a3ce43  
#5 [ffffa08ec21fb6d0] do\_trap at ffffffff8a5a38d39  
#6 [ffffa08ec21fb720] do\_error\_trap at ffffffff8a5a38e0f  
#7 [ffffa08ec21fb768] exc\_invalid\_op at ffffffff8a66204b3  
#8 [ffffa08ec21fb790] asm\_exc\_invalid\_op at ffffffff8a6800aa2  
[exception RIP: kfree+1129]  
RIP: ffffffff8a5d148e9 RSP: fffffa08ec21fb840 RFLAGS: 00010246  
RAX: ffff8a623b5f6c00 RBX: ffff8a623b5f6c00 RCX: ffff8a623b5f6d00  
RDX: 00000000000005915 RSI: ffff8a61c0000000 RDI: ffff8a61c1042600  
RBP: fffffa08ec21fb8b8 R8: ffff8a61c1042600 R9: ffffffff8cb200  
R10: 0000000000000000 R11: ffff8a623b5f6c00 R12: fffffea3181ed7d80  
R13: fffffea3181ed7d80 R14: ffff8a61c1042600 R15: ffffffff8a63c6612  
ORIG\_RAX: ffffffff8a63c6612 CS: 0010 SS: 0018  
#9 [ffffa08ec21fb8c0] skb\_release\_data at ffffffff8a63c6612  
#10 [ffffa08ec21fb8e8] consume\_skb at ffffffff8a63c6a0e  
#11 [ffffa08ec21fb900] vnet\_start\_xmit at ffffffff8a03682ad [veth\_netdrv]  
#12 [ffffa08ec21fb930] dev\_hard\_start\_xmit at ffffffff8a63e2b70  
#13 [ffffa08ec21fb980] sch\_direct\_xmit at ffffffff8a6433c62  
#14 [ffffa08ec21fb9d0] \_\_dev\_queue\_xmit at ffffffff8a63e33d9  
#15 [ffffa08ec21fba40] dev\_queue\_xmit at ffffffff8a63e35e0  
#16 [ffffa08ec21fba50] neigh\_resolve\_output at ffffffff8a63f0ff4  
#17 [ffffa08ec21fba88] ip\_finish\_output2 at ffffffff8a646dc7b  
#18 [ffffa08ec21fbae0] \_\_ip\_finish\_output at ffffffff8a646ea6c  
#19 [ffffa08ec21fbb20] ip\_finish\_output at ffffffff8a646eb31  
#20 [ffffa08ec21fbb58] ip\_output at ffffffff8a6470518  
#21 [ffffa08ec21fbbc0] ip\_send\_skb at ffffffff8a6470fb3  
#22 [ffffa08ec21fbbe8] udp\_send\_skb at ffffffff8a64a761e  
#23 [ffffa08ec21fbc28] udp\_sendmsg at ffffffff8a64aaff6  
#24 [ffffa08ec21fbdb8] inet\_sendmsg at ffffffff8a64ba0dd  
#25 [ffffa08ec21fbdc8] inet\_sendmsg at ffffffff8a64ba0dd  
#26 [ffffa08ec21fbdf0] \_\_sys\_sendto at ffffffff8a63b95fb << app sendto() >>  
#27 [ffffa08ec21fbea8] vfs\_write at ffffffff8a5d4f5b9  
#28 [ffffa08ec21fbeb0] vfs\_write at ffffffff8a5d4f5b9  
#29 [ffffa08ec21fbf28] \_\_x64\_sys\_sendto at ffffffff8a63b9654  
#30 [ffffa08ec21fbf38] do\_syscall\_64 at ffffffff8a661fe98  
#31 [ffffa08ec21fbf50] entry\_SYSCALL\_64\_after\_hwframe at ffffffff8a68000a9  
RIP: 00007f02f4293bba RSP: 00007ffd80dde498 RFLAGS: 00000246  
RAX: ffffffff8a63b9654 RBX: 0000000000000000 RCX: 00007f02f4293bba  
RDX: 0000000000000011 RSI: 00007ffd80de0855 RDI: 0000000000000003



```
RBP: 00007ffd80dde4f0 R8: 00007ffd80dde4d0 R9: 0000000000000010
R10: 0000000000000000 R11: 0000000000000246 R12: 00007ffd80dde638
R13: 0000556b356ba3f4 R14: 0000556b356bcd50 R15: 00007f02f43ee040
ORIG_RAX: 000000000000002c CS: 0033 SS: 002b
```

crash>

crash> log

...

...

```
[ 1280.033638] veth_netdrv:vnet_start_xmit(): UDP pkt::src=56894 dest=54295 len=6400
[ 1280.033639] veth_netdrv:vnet_start_xmit(): ah, a UDP packet Tx via our app (dest port
54295)
```

```
[ 1280.033640] veth_netdrv:vnet_start_xmit(): skb=ffff8a61cf4fab00 netdev=ffff8a61d090b000
pstCtx=ffff8a61d090b900
```

```
[ 1280.033642] veth_netdrv:vnet_start_xmit(): 002) talker_dgram :3610 | ...1 /*
vnet_start_xmit() */
```

```
[ 1280.033644] veth_netdrv:vnet_start_xmit(): ////////////////////////////////// SKB_PEEK
skb ptr: ffff8a61cf4fab00
len=59 truesize=768 users=1
Offsets: mac_header:2 network_header:16 transport_header:36
SKB packet pointers & offsets:
headroom : head:ffff8a623b5f6c00 - data:ffff8a623b5f6c02 [ 2 bytes]
pkt data : data - tail: 61 [ 59 bytes]
tailroom : tail - end:192 [ 131 bytes]
```

```
[ 1280.033645] veth_netdrv:vnet_start_xmit(): //////////////////////////////////
[ 1280.033651] 00000000: 00 00 48 0f 0e 0d 0a 02 48 0f 0e 0d 0a 02 08 00 ..H.....H.....
[ 1280.033652] 00000010: 45 00 00 2d a5 be 40 00 40 11 7c 20 0a 00 02 d2 E...-..@.@.| ....
[ 1280.033652] 00000020: 0a 00 02 10 de 3e d4 17 00 19 ff 7a 68 65 79 2c .....>.....zhey,
[ 1280.033653] 00000030: 20 76 65 74 68 2c 20 77 61 73 73 75 70 00 00 00 veth, wassup...
[ 1280.033653] 00000040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033654] 00000050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033655] 00000060: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033655] 00000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033656] 00000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033656] 00000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033657] 000000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033657] 000000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
[ 1280.033658] -----[ cut here ]-----
```

```
[ 1280.033658] refcount_t: underflow; use-after-free.
```

```
[ 1280.033670] WARNING: CPU: 2 PID: 3610 at lib/refcount.c:28
```

```
refcount_warn_saturate+0xf7/0x150
```

```
[ 1280.033671] Modules linked in: veth_netdrv(OE) vboxvideo(OE) binfmt_misc vboxsf(OE)
intel_rapl_msr vmwgfx intel_rapl_common crct10dif_pclmul snd_intel8x0 crc32_pclmul
snd_ac97_codec ghash_clmulni_intel aesni_intel ac97_bus glue_helper crypto_simd snd_pcm cryptd
drm_kms_helper rapl joydev snd_seq syscopyarea sysfillrect sysimgblt fb_sys_fops cec snd_timer
rc_core snd_seq_device ttm snd_input_leds serio_raw soundcore vboxguest(OE) video sch_fq_codel
drm msr parport_pc ppdev lp parport ip_tables x_tables autofs4 hid_generic usbhid hid psmouse
ahci libahci e1000 i2c_piix4 pata_acpi
```

```
[ 1280.033702] CPU: 2 PID: 3610 Comm: talker_dgram Kdump: loaded Tainted: G OE
5.10.153-kdbg1 #1
```

```
[ 1280.033703] Hardware name: innotek GmbH VirtualBox/VirtualBox, BIOS VirtualBox 12/01/2006
```

```
[ 1280.033705] RIP: 0010:refcount_warn_saturate+0xf7/0x150
```

```
[ 1280.033706] Code: eb 9e 0f b6 1d 84 0d a4 01 80 fb 01 0f 87 e7 d6 63 00 83 e3 01 75 89 48
c7 c7 c8 b5 fa a6 c6 05 68 0d a4 01 01 e8 38 8f 60 00 <0f> 0b e9 6f ff ff ff 0f b6 1d 53 0d a4
01 80 fb 01 0f 87 a4 d6 63
```

```
[ 1280.033706] RSP: 0018:ffffa08ec21fb8a0 EFLAGS: 00010286
```

```
[ 1280.033707] RAX: 0000000000000000 RBX: 0000000000000000 RCX: 0000000000000027
```

```
[ 1280.033708] RDX: ffff8a623dca0a48 RSI: 0000000000000001 RDI: ffff8a623dca0a40
```

```
[ 1280.033708] RBP: ffffa08ec21fb8a8 R08: 0000000000000003 R09: ffffffffcb200
```

```
[ 1280.033709] R10: 0000000000000001 R11: 0000000000000001 R12: ffff8a61c5d5b300
```

```
[ 1280.033710] R13: ffff8a61c5d5b43c R14: ffff8a61d090b91c R15: ffff8a61d090b000
```

```
[ 1280.033711] FS: 00007f02f4169740(0000) GS: ffff8a623dc80000(0000) knlGS:0000000000000000
```

```
[ 1280.033711] CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
```

```
[ 1280.033712] CR2: 00007f02f432b423 CR3: 000000000ffe6006 CR4: 00000000000706e0
```

```
[ 1280.033714] Call Trace:
[ 1280.033720] sock_wfree+0xc0/0xd0
[ 1280.033722] skb_release_head_state+0x36/0x80
[ 1280.033723] consume_skb+0x2b/0xb0
[ 1280.033726] vnet_start_xmit+0x9d/0xb0 [veth_netdrv]
[ 1280.033727] dev_hard_start_xmit+0xd0/0x1e0
[ 1280.033729] sch_direct_xmit+0xe2/0x260
[ 1280.033730] __dev_queue_xmit+0x6a9/0x8a0
...
```

crash> log << look up kernel log (dmesg) >>

```
...
crash> log |grep "netdev="
veth_netdrv:vnet_start_xmit(): skb=ffff950a510d8500 netdev=ffff950a8889d000
pstCtx=ffff950a8889d900
...
```

### Look up the stack

Looks like the KVA's (kernel va's) in green are linkage to called frame (function).

crash> bt -FF

```
...
...
#9 [fffffae4543f4b8c0] skb_release_data at ffffffff81dc6612
fffffae4543f4b8c8: [ffff950a85450424:kmalloc-512]
[ffff950a510d8500:skbuff_head_cache]
fffffae4543f4b8d8: [ffff950a8889d000:kmalloc-4k] fffffae4543f4b8f8
fffffae4543f4b8e8: consume_skb+62

#10 [fffffae4543f4b8e8] consume_skb at ffffffff81dc6a0e
fffffae4543f4b8f0: [ffff950a510d8500:skbuff_head_cache] fffffae4543f4b928
fffffae4543f4b900: vnet_start_xmit+157
```

```
<<
static int vnet_start_xmit(struct sk_buff *skb, struct net_device *ndev)
{
    struct iphdr ip = NULL;      << locals seen in RTL (right-to-left) order >>
    struct udphdr *udph = NULL;
    struct stVnetIntfCtx *pstCtx = netdev_priv(ndev);
    ...
    u64 ts1, ts2; << unused, optimized away >>
>>
#11 [fffffae4543f4b900] vnet_start_xmit at ffffffff807b32ad [veth_netdrv]
fffffae4543f4b908: [ffff950a5016ec00:kmalloc-512] 0000000000000000
fffffae4543f4b918: 00000000000000003b [ffff950a510d8500:skbuff_head_cache]
fffffae4543f4b928: fffffae4543f4b978 dev_hard_start_xmit+208 <<RET addr>>
```

<<-- So, very important: Call frame layout on x86\_64:

LOCALS [...3,2,1]

<< PARAMS\* may or may not be pushed onto stack >>

[SFP?]RBP\*

RET addr

-->>

<<

\* See the disassembly (you might find a **push**)

\* See notes below for inferring RBP

va

>>

```
#12 [fffffae4543f4b930] dev_hard_start_xmit at ffffffff81de2b70
fffffae4543f4b938: fffffae4543f4b994 [fffff950a8889d088:kmalloc-4k]
fffffae4543f4b948: fffffae4543f4b978 [fffff950a5016ec00:kmalloc-512]
fffffae4543f4b958: [fffff950a88b57e00:kmalloc-512]
                    [fffff950a510d8500:skbuff_head_cache]
fffffae4543f4b968: [fffff950a88b57eac:kmalloc-512] [fffff950a8889d000:kmalloc-4k]
fffffae4543f4b978: fffffae4543f4b9c8 sch_direct_xmit+226

#13 [fffffae4543f4b980] sch_direct_xmit at ffffffff81e33c62
fffffae4543f4b988: 00000000581ec13f1 0000000100089d000
fffffae4543f4b998: 277ccc06ec916700 0000000000000000
fffffae4543f4b9a8: [fffff950a510d8500:skbuff_head_cache]
                    [fffff950a88b57e00:kmalloc-512]
fffffae4543f4b9b8: [fffff950a8889d000:kmalloc-4k] [fffff950a5016ec00:kmalloc-512]
fffffae4543f4b9c8: fffffae4543f4ba38 __dev_queue_xmit+1705

#14 [fffffae4543f4b9d0] __dev_queue_xmit at ffffffff81de33d9
fffffae4543f4b9d8: fffffae4543f4b9e8 [fffff950a88b57eac:kmalloc-512]
fffffae4543f4b9e8: ffffffff443f4ba70 0000000000000000
fffffae4543f4b9f8: fffffae4543f4bb44 [fffff950a412f0000:task_struct]
fffffae4543f4ba08: 277ccc06ec916700 [fffff950a85451000:kmalloc-512]
fffffae4543f4ba18: 0000000000000000 [fffff950a510d8500:skbuff_head_cache]
fffffae4543f4ba28: [fffff950a8889d000:kmalloc-4k] 00000000000000008
fffffae4543f4ba38: fffffae4543f4ba48 dev_queue_xmit+16

#15 [fffffae4543f4ba40] dev_queue_xmit at ffffffff81de35e0
fffffae4543f4ba48: fffffae4543f4ba80 neigh_resolve_output+276
...
```

[fffff950a510d8500:skbuff\_head\_cache]

# the SKB that's freed up too soon, resulting in a UAF; the root cause of the bug.

### Useful

```
crash> bt -FF |grep -C3 --color=always fffff8a61f6a55600
```

...

### Original un-annotated stack frames

```
crash> bt -F
```

...

...

```
#9 [fffffae4543f4b8c0] skb_release_data at ffffffff81dc6612
fffffae4543f4b8c8: [kmalloc-512] [skbuff_head_cache]
fffffae4543f4b8d8: [kmalloc-4k] fffffae4543f4b8f8
fffffae4543f4b8e8: consume_skb+62
#10 [fffffae4543f4b8e8] consume_skb at ffffffff81dc6a0e
fffffae4543f4b8f0: [skbuff_head_cache] fffffae4543f4b928
fffffae4543f4b900: vnet_start_xmit+157
#11 [fffffae4543f4b900] vnet_start_xmit at ffffffff807b32ad [veth_netdrv]
fffffae4543f4b908: [kmalloc-512] 0000000000000000
fffffae4543f4b918: 0000000000000003b [skbuff_head_cache]
```



```
fffffae4543f4b928: fffffae4543f4b978 dev_hard_start_xmit+208
#12 [fffffae4543f4b930] dev_hard_start_xmit at ffffffff81de2b70
fffffae4543f4b938: fffffae4543f4b994 [kmallo-4k]
fffffae4543f4b948: fffffae4543f4b978 [kmallo-512]
fffffae4543f4b958: [kmallo-512] [skbuff_head_cache]
fffffae4543f4b968: [kmallo-512] [kmallo-4k]
fffffae4543f4b978: fffffae4543f4b9c8 sch_direct_xmit+226
#13 [fffffae4543f4b980] sch_direct_xmit at ffffffff81e33c62
fffffae4543f4b988: 00000000581ec13f1 00000000100089d000
fffffae4543f4b998: 277ccc06ec916700 0000000000000000
fffffae4543f4b9a8: [skbuff_head_cache] [kmallo-512]
fffffae4543f4b9b8: [kmallo-4k] [kmallo-512]
fffffae4543f4b9c8: fffffae4543f4ba38 __dev_queue_xmit+1705
#14 [fffffae4543f4b9d0] __dev_queue_xmit at ffffffff81de33d9
fffffae4543f4b9d8: fffffae4543f4b9e8 [kmallo-512]
fffffae4543f4b9e8: ffffffff443f4ba70 0000000000000000
fffffae4543f4b9f8: fffffae4543f4bb44 [task_struct]
fffffae4543f4ba08: 277ccc06ec916700 [kmallo-512]
fffffae4543f4ba18: 0000000000000000 [skbuff_head_cache]
fffffae4543f4ba28: [kmallo-4k] 00000000000000008
fffffae4543f4ba38: fffffae4543f4ba48 dev_queue_xmit+16
#15 [fffffae4543f4ba40] dev_queue_xmit at ffffffff81de35e0
fffffae4543f4ba48: fffffae4543f4ba80 neigh_resolve_output+276
...
=====
```

**Excellent article!**

**Src:** [Extracting kernel stack function arguments from Linux x86-64 kernel crash dumps, Calum Mackay, Sept 2020](#)

...

The **stack frame base pointer**, (%RBP), for a function, may be found:

- As the **second-last value** in the stack frame above the function (i.e. above in the **bt** output)
- As the **location of the second-last value** in the stack frame for the function

...

<<

Eg. find RBP for the function consume\_skb():

**crash> bt -FF**

```
...
#9 [ffffa08ec21fb8c0] skb_release_data at ffffffff63c6612
ffffa08ec21fb8c8: [ffff8a623b5f6c24:kmalloc-512] [ffff8a61cf4fab00:skbuff_head_cache]
ffffa08ec21fb8d8: [ffff8a61d090b000:kmalloc-4k] fffffa08ec21fb8f8
ffffa08ec21fb8e8: consume_skb+62
#10 [ffffa08ec21fb8e8] consume_skb at ffffffff63c6a0e
ffffa08ec21fb8f0: [ffff8a61cf4fab00:skbuff_head_cache] fffffa08ec21fb928
ffffa08ec21fb900: vnet_start_xmit+157
#11 [ffffa08ec21fb900] vnet_start_xmit at ffffffff6c03682ad [veth_netdrv]
ffffa08ec21fb908: [ffff8a61f6a55600:kmalloc-512] 0000000000000000
ffffa08ec21fb918: 0000000000000003b [ffff8a61cf4fab00:skbuff_head_cache]
ffffa08ec21fb928: fffffa08ec21fb978 dev_hard_start_xmit+208
...
...
```

First method: RBP is the value **ffffa08ec21fb8f8**

Second method: 2<sup>nd</sup> last value in stack frame of consume\_skb() is **ffffa08ec21fb928**. Its location is **ffffa08ec21fb8f0+0x8 = fffffa08ec21fb8f8 !**

>>

...

**<< Very Important! >>**

...

## Summary of steps

1. Note which registers you need, corresponding to the position of the called function's arguments you need << Recall: on x86\_64, first 6 parameters are passed via the registers %rdi, %rsi, %rdx, %rcx, %r8, %r9, not via the stack >>
  1. Refer to the register-naming table above, **in case the quantities passed are smaller than 64-bit**, e.g. integers, other non-pointer types. The 1st argument will be passed in %rdi, %edi, %di or %dil. Note that all the names contain "di".
2. **Disassemble the calling function**, and inspect the instructions leading up to where it calls the function you're interested in. **Note from where the compiler gets the values it places in those registers**
  1. If from the stack, find the caller's stack frame base pointer, and from there find the value in the stack frame

2. If from memory, can you calculate the memory address used? If so, read the value from memory
3. If from another register, from where was *that* register's contents obtained? And see case 3.3 below. << **Very often, the parameter's in a register (say, %r12); so where/how did r12 get set?** In effect, you have to keep following the disassembly... again, often/hopefully, an earlier function's assembly code PUSHed that register on the stack somewhere! Now you can find what the value in the register is by looking up that stack location! >>
3. **Disassemble the first part of the called function.** Note where it stores the values passed in the registers you need
  1. If onto the stack, find the called function's stack frame base pointer, and find the value in the stack frame
  2. If from memory, can you calculate the memory address used? If so, read the value from memory
  3. If the calling function obtained the value from another register (case 2.3 above) does the called function save *that* register to stack/memory?
4. If none of the above gave a usable result, see if the values you need are passed to another function call **further up or down the stack**, or may be derived from a different value.
  1. For example the structure you want is referenced from another structure that is passed to a function elsewhere in the stack trace
5. Once you've obtained answers, perform a sanity check
  1. Is the value obtained on a slab cache? If so, is the cache of the expected type?
  2. Is the value, or what it points to, of the expected type?
  3. If the value is a pointer to a structure, does the structure content look correct? e.g. pointers where pointers are expected, function op pointers pointing to real functions, etc
6. Read the *Caveats* section, to understand whether you can rely on the answer you've found

At this point, you may either skip directly to the *Worked Examples* , or read on for more detail.

...

---

## Investigation: find how & which params are passed by the caller of `vnet_start_xmit()`, i.e., by `dev_hard_start_xmit()`:

`crash>bt -FF`

```
...
#9 [ffffa543410ab8c0] skb_release_data at ffffffff8a3c6612
ffffa543410ab8c8: [ffff955112efb424:kmalloc-512] [ffff955111e5ab00:skbuff_head_cache]
ffffa543410ab8d8: [ffff95511c085000:kmalloc-4k] fffffa543410ab8f8
ffffa543410ab8e8: consume_skb+62
#10 [ffffa543410ab8e8] consume_skb at ffffffff8a3c6a0e
ffffa543410ab8f0: [ffff955111e5ab00:skbuff_head_cache] fffffa543410ab928
ffffa543410ab900: ()+157
#11 [ffffa543410ab900] vnet_start_xmit at ffffffff8c083b2ad [veth_netdrv]
ffffa543410ab908: [ffff95514bc29200:kmalloc-512] 0000000000000000
ffffa543410ab918: 0000000000000003b [ffff955111e5ab00:skbuff_head_cache]
```

```

fffffa543410ab928: fffffa543410ab978 dev_hard_start_xmit+208
#12 [fffffa543410ab930] dev_hard_start_xmit at ffffffff8a3e2b70
fffffa543410ab938: fffffa543410ab994 [ffff95511c085088:kmalloc-4k]
fffffa543410ab948: fffffa543410ab978 [ffff95514bc29200:kmalloc-512]
fffffa543410ab958: [ffff95514bc28600:kmalloc-512] [ffff95511e5ab00:skbuff_head_cache]
fffffa543410ab968: [ffff95514bc286ac:kmalloc-512] [ffff95511c085000:kmalloc-4k]
fffffa543410ab978: fffffa543410ab9c8 sch_direct_xmit+226
#13 [fffffa543410ab980] sch_direct_xmit at ffffffff8a433c62
fffffa543410ab988: 0000000058a4c13f1 00000001000085000
fffffa543410ab998: 5979a8f4c1114800 0000000000000000
fffffa543410ab9a8: [ffff95511e5ab00:skbuff_head_cache] [ffff95514bc28600:kmalloc-512]
fffffa543410ab9b8: [ffff95511c085000:kmalloc-4k] [ffff95514bc29200:kmalloc-512]
fffffa543410ab9c8: fffffa543410aba38 __dev_queue_xmit+1705
#14 [fffffa543410ab9d0] __dev_queue_xmit at ffffffff8a3e33d9
fffffa543410ab9d8: fffffa543410ab9e8 [ffff95514bc286ac:kmalloc-512]
fffffa543410ab9e8: ffffffff4410aba70 0000000000000000
fffffa543410ab9f8: fffffa543410abb44 [ffff955103238000:task_struct]
fffffa543410aba08: 5979a8f4c1114800 [ffff955112efa000:kmalloc-512]
fffffa543410aba18: 0000000000000000 [ffff95511e5ab00:skbuff_head_cache]
fffffa543410aba28: [ffff95511c085000:kmalloc-4k] 0000000000000008
fffffa543410aba38: fffffa543410aba48 dev_queue_xmit+16
...

```

```
static int vnet_start_xmit(struct sk_buff *skb, struct net_device *ndev);
```

Can see that dev\_hard\_start\_xmit() calls vnet\_start\_xmit(). But where in the disassembly?? not apparent as it's a virtual func call (a call to a struct ptr)...

Look at this:

```

#11 [fffffa543410ab900] vnet_start_xmit at ffffffff8c083b2ad [veth_netdrv]
...
fffffa543410ab928: fffffa543410ab978 dev_hard_start_xmit+208 <---

```

The RET addr is 208 bytes after the start of dev\_hard\_start\_xmit()!

So, now we can easily gauge the call point and thus the prologue - the setting up of (param) registers!

crash> dis -l dev\_hard\_start\_xmit

```

/home/osboxes/linux-5.10.153/net/core/dev.c: 3596
0xffffffff8a3e2aa0 <dev_hard_start_xmit>:      nopl      0x0(%rax,%rax,1) [FTRACE NOP]
/home/osboxes/linux-5.10.153/net/core/dev.c: 3597
0xffffffff8a3e2aa5 <dev_hard_start_xmit+5>:      push      %rbp
/home/osboxes/linux-5.10.153/net/core/dev.c: 2308
0xffffffff8a3e2aa6 <dev_hard_start_xmit+6>:      lea       0x88(%rsi),%rax
/home/osboxes/linux-5.10.153/net/core/dev.c: 3596 <---
0xffffffff8a3e2aad <dev_hard_start_xmit+13>:      mov       %rsp,%rbp
0xffffffff8a3e2ab0 <dev_hard_start_xmit+16>:      push     %r15
0xffffffff8a3e2ab2 <dev_hard_start_xmit+18>:      mov      %rsi,%r15
0xffffffff8a3e2ab5 <dev_hard_start_xmit+21>:      push     %r14
0xffffffff8a3e2ab7 <dev_hard_start_xmit+23>:      mov      %rdi,%r14

```

<<

net/core/dev.c

```

3594 struct sk_buff *dev_hard_start_xmit(struct sk_buff *first, struct net_device *dev,
3595                                     struct netdev_queue *txq, int *ret)
3596 {      <--- [A]          << see below >>
3597     struct sk_buff *skb = first;
>>

```

```

/home/osboxes/linux-5.10.153/net/core/dev.c: 3600
0xffffffff8a3e2aba <dev_hard_start_xmit+26>:      push     %r13
0xffffffff8a3e2abc <dev_hard_start_xmit+28>:      push     %r12
0xffffffff8a3e2abe <dev_hard_start_xmit+30>:      push     %rbx

```

```

0xffffffff8a3e2abf <dev_hard_start_xmit+31>: mov    %rdx,%rbx
/home/osboxes/linux-5.10.153/net/core/dev.c: 3598
0xffffffff8a3e2ac2 <dev_hard_start_xmit+34>: xor     %edx,%edx
/home/osboxes/linux-5.10.153/net/core/dev.c: 3596
0xffffffff8a3e2ac4 <dev_hard_start_xmit+36>: sub     $0x18,%rsp
0xffffffff8a3e2ac8 <dev_hard_start_xmit+40>: mov     %rcx,-0x40(%rbp)
/home/osboxes/linux-5.10.153/net/core/dev.c: 2308
0xffffffff8a3e2acc <dev_hard_start_xmit+44>: mov     %rax,-0x38(%rbp)
/home/osboxes/linux-5.10.153/net/core/dev.c: 3600
0xffffffff8a3e2ad0 <dev_hard_start_xmit+48>: test    %r14,%r14
0xffffffff8a3e2ad3 <dev_hard_start_xmit+51>: je      0xffffffff8a3e2bb6
<dev_hard_start_xmit+278>
/home/osboxes/linux-5.10.153/net/core/dev.c: 3601
0xffffffff8a3e2ad9 <dev_hard_start_xmit+57>: mov     (%r14),%r12
/home/osboxes/linux-5.10.153/.include/linux/skbuff.h: 1503
0xffffffff8a3e2adc <dev_hard_start_xmit+60>: movq    $0x0,(%r14)
/home/osboxes/linux-5.10.153/net/core/dev.c: 3604
0xffffffff8a3e2ae3 <dev_hard_start_xmit+67>: mov     0x1626806(%rip),%rax      #
0xffffffff8ba092f0 <ptype_all>
0xffffffff8a3e2aea <dev_hard_start_xmit+74>: test    %r12,%r12
0xffffffff8a3e2aed <dev_hard_start_xmit+77>: setne   %cl
/home/osboxes/linux-5.10.153/.include/linux/list.h: 282
0xffffffff8a3e2af0 <dev_hard_start_xmit+80>: cmp     $0xffffffff8ba092f0,%rax
0xffffffff8a3e2af6 <dev_hard_start_xmit+86>: je      0xffffffff8a3e2c67
<dev_hard_start_xmit+455>
/home/osboxes/linux-5.10.153/net/core/dev.c: 3583
0xffffffff8a3e2afc <dev_hard_start_xmit+92>: mov     %r15,%rsi
0xffffffff8a3e2aff <dev_hard_start_xmit+95>: mov     %r14,%rdi
0xffffffff8a3e2b02 <dev_hard_start_xmit+98>: mov     %cl,-0x2c(%rbp)
0xffffffff8a3e2b05 <dev_hard_start_xmit+101>: call    0xffffffff8a3dcba0 <dev_queue_xmit_nit>
0xffffffff8a3e2b0a <dev_hard_start_xmit+106>: movzbl  -0x2c(%rbp),%ecx
/home/osboxes/linux-5.10.153/net/core/dev.c: 3585
0xffffffff8a3e2b0e <dev_hard_start_xmit+110>: mov     0x68(%r14),%esi
/home/osboxes/linux-5.10.153/net/core/dev.c: 3586
0xffffffff8a3e2b12 <dev_hard_start_xmit+114>: mov     0x1224e67(%rip),%rax      #
0xffffffff8b607980 <jiffies>
/home/osboxes/linux-5.10.153/net/core/dev.c: 3585
0xffffffff8a3e2b19 <dev_hard_start_xmit+121>: mov     %rsi,%r13
/home/osboxes/linux-5.10.153/net/core/dev.c: 3586
0xffffffff8a3e2b1c <dev_hard_start_xmit+124>: add     %rax,%rsi
/home/osboxes/linux-5.10.153/.include/linux/prandom.h: 57
0xffffffff8a3e2b1f <dev_hard_start_xmit+127>: mov     %gs:0x75c4efb9(%rip),%rax      #
0x31ae0
0xffffffff8a3e2b27 <dev_hard_start_xmit+135>: xor     %r14,%rax
/home/osboxes/linux-5.10.153/.include/linux/bitops.h: 85
0xffffffff8a3e2b2a <dev_hard_start_xmit+138>: lea     (%rbx,%rsi,1),%rdx
0xffffffff8a3e2b2e <dev_hard_start_xmit+142>: rol     $0x10,%rsi
/home/osboxes/linux-5.10.153/.include/linux/prandom.h: 58
0xffffffff8a3e2b32 <dev_hard_start_xmit+146>: add     %r15,%rax
0xffffffff8a3e2b35 <dev_hard_start_xmit+149>: xor     %rsi,%rdx
/home/osboxes/linux-5.10.153/.include/linux/bitops.h: 85
0xffffffff8a3e2b38 <dev_hard_start_xmit+152>: rol     $0x20,%rax
0xffffffff8a3e2b3c <dev_hard_start_xmit+156>: add     %rdx,%rax
0xffffffff8a3e2b3f <dev_hard_start_xmit+159>: rol     $0x15,%rdx
/home/osboxes/linux-5.10.153/.include/linux/prandom.h: 59
0xffffffff8a3e2b43 <dev_hard_start_xmit+163>: xor     %rdx,%rax
0xffffffff8a3e2b46 <dev_hard_start_xmit+166>: mov     %rax,%gs:0x75c4ef92(%rip)      #
0x31ae0
/home/osboxes/linux-5.10.153/.arch/x86/include/asm/jump_label.h: 25
0xffffffff8a3e2b4e <dev_hard_start_xmit+174>: nopl    0x0(%rax,%rax,1)
/home/osboxes/linux-5.10.153/.include/linux/netdevice.h: 4803
0xffffffff8a3e2b53 <dev_hard_start_xmit+179>: mov     0x1e0(%r15),%rax
/home/osboxes/linux-5.10.153/.include/linux/netdevice.h: 4791
0xffffffff8a3e2b5a <dev_hard_start_xmit+186>: mov     %r15,%rsi      ← 2nd parameter

```

```

0xffffffff8a3e2b5d <dev_hard_start_xmit+189>:  mov    %r14,%rdi    ← 1st parameter
0xffffffff8a3e2b60 <dev_hard_start_xmit+192>:  mov    %cl,%gs:0x75c522fb(%rip)    #
0x34e62
/home/osboxes/linux-5.10.153/./include/linux/netdevice.h: 4792
0xffffffff8a3e2b67 <(+199>:  mov    0x20(%rax),%rax
0xffffffff8a3e2b6b <dev_hard_start_xmit+203>:  call   0xffffffff8aa02100
<__x86_indirect_thunk_array>
0xffffffff8a3e2b70 <dev_hard_start_xmit+208>:  mov    %eax,%edx

```

...

See the 'call' instr (2nd last line)!

Now the assembly which first touches `%r14` 's here:

```

0xffffffff8a3e2ab5 <dev_hard_start_xmit+21>:  push    %r14

```

So it corr to src code line 3596; here:

See pt [A] :

```

3594 struct sk_buff *dev_hard_start_xmit(struct sk_buff *first, struct net_device *dev,
3595                                     struct netdev_queue *txq, int *ret)
3596 {    <--- [A]
3597     struct sk_buff *skb = first;

```

Aha! the compiler's setting the local variable - **using register r14 for optimization!** - the SKB to the first param (rdi val);

So we confirm that r14, and thus rdi, and thus the first parameter, is the SKB ptr !!!

## 2nd param: rsi:

```

...
/home/osboxes/linux-5.10.153/net/core/dev.c: 3583
0xffffffff8a3e2afc <dev_hard_start_xmit+92>:  mov    %r15,%rsi
...

```

It's set to the register r15's value. We need to find what that is...

See the disassembly (search for the needed register (`%r15` here) being set):

This line sets rsi to r15; rsi is the 2nd param passed to this func - `dev_hard_start_xmit()`; thus it's `struct net_device *dev` ! Now setting this to r15

```

0xffffffff8a3e2ab2 <dev_hard_start_xmit+18>:  mov    %rsi,%r15

```

and then r15 gets set to rsi before calling our `vnet_start_xmit()` func ; thus, the 2nd param passed is indeed the ptr to `struct net_device`.

Done.

---