rootfs remount on kppless is going to be even harder than I thought :\ (edited)
so some info for now
LightweightVolumeManager has it's internal array of "partitions"
actually pointers to partitions
it's located at offset 0x1A0
on offset 0x198 there's size of that array
and at index 0 is root partition obviously
and whatever partition is, it has some flag (let's call it isWriteProtected) at offset 0x28
if that flag is set, mapping would fail
so, we need to somehow locate LightweightVolumeManager instance
then do `(uint8_t*)(uint8_t* LightweightVolumeManagerInst)[0x1A0][0x28] = 0` (edited)
to unset that flag
however, there's another check which has to be bypassed

stek29 Yesterday at 7:03 AM
long story short -- to bypass that check you either have to make PE_i_can_has_kernel_configuration return 1, which is impossible under kpp/ktrr
or you have to add either `rd=md?` or `rootdev=md?` to bootargs
where ? can be any char
you have to add that to in-kernel bootargs, not to pass them from iboot I mean
upd: Partition is LwVMPartition object
for example
let's first leak LightweightVolumeManager

```
[1]+  Stopped(SIGTSTP)         ./memctl
Naomis-iPhone:/fox root# ./explorer -K 0xffffffff00d7b2050 -s LightweightVolumeManager
port  : 0xe03
inkern: 0xffffffff110d3abf8
kobj  : 0xffffffff10e767000
vtable: 0xffffffff00d120510
port  : 0xd07
inkern: 0xffffffff110bb96f8
kobj  : 0xffffffff110f93000
vtable: 0xffffffff00d11f8a0
Naomis-iPhone:/fox root#
```

here we can see that LightweightVolumeManager is at `0xffffffff10e767000`
reading at that address gives us pointer to vtable

```
memctl> r 0xffffffff10e767000
ffffffff10e767000:  ffffffff00d120510
memctl>
```

and as you can see vtable matches

```
~/Projects/memctl/bin master*
❯ ~/Projects/iometa/iometa -v ~/workdir/kc/iphonese10|grep Lightweight
vtab=0xffffffff006f20510 LightweightVolumeManager
```

so we know that it's actually LightweightVolumeManager instance
now the next step -- let's look into partitions

reading at offset 0x198 gives 3 -- as expected -- there are 3 partitions (root, var, baseband)

```
memctl> r 0xfffffff10e767198
fffffff10e767198:  000000000000003
```

so now let's read some partition and check it's vtable

```
memctl> r 0xfffffff10e7671a8
fffffff10e7671a8:  fffffff10e1b9bd0
memctl> r 0xfffffff10e1b9bd0
fffffff10e1b9bd0:  fffffff00d120da0
```

```
vtab=0xfffffff006f20c88 LwVMPartialIOPo
vtab=0xfffffff006f20da0 LwVMPartition
```

as we can see that's clearly LwVM Partition

now let's go further and check if isWriteProtected flag on all of them

```
memctl> r 0xfffffff10e7671a0
fffffff10e7671a0:  fffffff10e1b8870
memctl> r 0xfffffff10e1b8898
fffffff10e1b8898:  000000000000001
memctl> r 0xfffffff10e7671a8
fffffff10e7671a8:  fffffff10e1b9bd0
memctl> r 0xfffffff10e1b9bf8
fffffff10e1b9bf8:  000000000000000
memctl> r 0xfffffff10e7671b0
fffffff10e7671b0:  fffffff10e1b8500
memctl> r 0xfffffff10e1b8528
fffffff10e1b8528:  000000000000000
```

as expected -- root has that flag set, and data/baseband don't

so to pass that check:

```
io_service_t service =
IOServiceGetMatchingService(kIOMasterPortDefault,
IOServiceMatching("LightweightVolumeManager"));
uint64_t inkernel = find_port_address(service);
uint64_t lwvm_kaddr = rk64(inkernel + OFF_IPC_PORT__IP_KOBJECT);
uint64_t rootp_kaddr = rk64(lwvm_kaddr + OFF_LWVM__PARTITIONS);
uint64_t rootp_iswp_addr = rootp_kaddr + OFF_LWVMPART_ISWP;
// rk64(rootp_iswp_addr) should be 1
wk64(rootp_iswp_addr, 0);
```

where OFF_LWVM__PARTITIONS is 0x1a0, OFF_LWVMPART_ISWP is 0x28

now to boot args
so, bootargs in kernel are accessed/stored/etc like this
there's PE_state structure
here's it's definition (form pexpert/pexpert/pexpert.h)

```
typedef struct PE_state {
    boolean_t    initialized;
    PE_Video    video;
    void        *deviceTreeHead;
    void        *bootArgs;
} PE_state_t;
```

there's one global variable with type PE_state_t
bootArgs points to struct boot_args which is platform dependent
on arm64 (pexpert/pexpert/arm64/boot.h):

```
typedef struct boot_args {
    uint16_t        Revision;            /* Revision of boot_args
structure */
    uint16_t        Version;            /* Version of boot_args
structure */
    uint64_t        virtBase;            /* Virtual base of memory */
    uint64_t        physBase;            /* Physical base of memory */
    uint64_t        memSize;            /* Size of memory */
    uint64_t        topOfKernelData;    /* Highest physical address
used in kernel data area */
    Boot_Video        Video;                /* Video Information */
    uint32_t        machineType;        /* Machine Type */
    void            *deviceTreeP;        /* Base of flattened device
tree */
    uint32_t        deviceTreeLength;    /* Length of flattened tree */
    char            CommandLine[BOOT_LINE_LENGTH];    /* Passed in
command line */
    uint64_t        bootFlags;        /* Additional flags specified by
the bootloader */
    uint64_t        memSizeActual;        /* Actual size of memory */
} boot_args;
```

notice `char CommandLine[BOOT_LINE_LENGTH];`
now, when some code wants to parse some boot arg, it calls `PE_parse_boot_argn` function

```
boolean_t
PE_parse_boot_argn(
    const char    *arg_string,
    void          *arg_ptr,
    int            max_len)
{
    return PE_parse_boot_argn_internal(arg_string, arg_ptr, max_len,
FALSE);
}
```

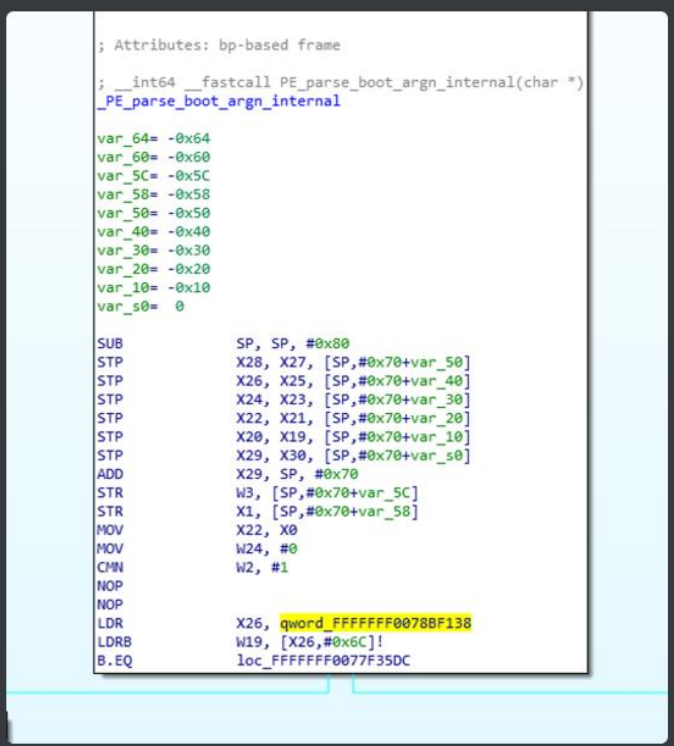
PE_parse_boot_argn_internal performs parsing

to get boot args string it does `args = PE_boot_args();`

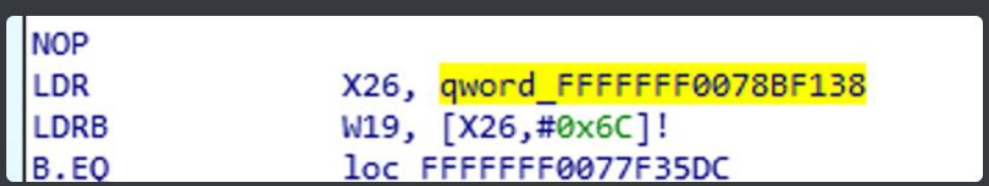
`PE_boot_args` is once again platform specific, on arm64:

```
char *
PE_boot_args(
    void)
{
    return (char *)((boot_args *)PE_state.bootArgs)->CommandLine;
}
```

now knowing that, let's look into PE_parse_boot_argn_internal disassembly



notice these lines



if we go to that addr and scroll 0xa0 up...

if we go to that addr and scroll 0xa0 up...

```
:FFFFFFF0078BF090                                       ; _do_kern_dump+44↑r ...
:FFFFFFF0078BF098                       EXPORT _PE_state
:FFFFFFF0078BF098 _PE_state             % 4             ; DATA XREF: _kdp_stackshot_kcdata_format:l
:FFFFFFF0078BF098                                       ; _kdp_stackshot_kcdata_format+2D64↑o ...
:FFFFFFF0078BF09C                       ALIGN 0x20
:FFFFFFF0078BF0A0 qword_FFFFFFF0078BF0A0 % 8            ; DATA XREF: _PE_init_platform+48↑w
:FFFFFFF0078BF0A8 qword_FFFFFFF0078BF0A8 % 8            ; DATA XREF: _PE_init_platform+50↑w
:FFFFFFF0078BF0B0 qword_FFFFFFF0078BF0B0 % 8            ; DATA XREF: _PE_init_iokit:loc_FFFFFFF0077
:FFFFFFF0078BF0B0                                       ; _PE_init_platform+58↑w
:FFFFFFF0078BF0B8 qword_FFFFFFF0078BF0B8 % 8            ; DATA XREF: _PE_init_iokit:loc_FFFFFFF0077
:FFFFFFF0078BF0B8                                       ; _PE_init_platform+60↑w
:FFFFFFF0078BF0C0 qword_FFFFFFF0078BF0C0 % 8            ; DATA XREF: _PE_init_platform+6C↑w
:FFFFFFF0078BF0C8 qword_FFFFFFF0078BF0C8 % 8            ; DATA XREF: _PE_init_platform+88↑w
:FFFFFFF0078BF0C8                                       ; _PE_create_console+11C↑r
:FFFFFFF0078BF0D0                       % 1
```

so, _PE_state.bootArgs is at offset 0xa0, but it's easier to find it directly (edited)

and `boot_args.CommandLine` is at offset 0x6c

let me show those in kernel I currently have on device

here IDA even got the right base and offset :)

```
NOP
LDR             X26, [X8,#(qword_FFFFFFF0075B15A8 - 0xFFFFFFF0075B1508)]
LDRB            W21, [X26,#0x6C]!
CMN             W2, #1
```

so now let's read them in booted kernel

```
memctl> r 0xffffffff00d7b15a8
ffffffff00d7b15a8:  ffffffff00e084000
```

so, `PE_state.bootArgs` points to 0xffffffff00e084000 (edited)

let's dump it

```
Naomis-iPhone:/fox root# ./ios-kern-utils/kmem 0xffffffff00e084000 0x100
[*] Reading 256 bytes from 0xffffffff00e084000
02 00 02 00 00 00 00 00  00 00 00 0C F0 FF FF FF  |................|
00 00 00 00 08 00 00 00  00 00 98 7D 00 00 00 00  |...........}....|
00 80 0B 02 08 00 00 00  00 00 60 7E 08 00 00 00  |..........`~....|
01 00 00 00 00 00 00 00  00 0A 00 00 00 00 00 00  |................|
80 02 00 00 00 00 00 00  70 04 00 00 00 00 00 00  |........p.......|
20 00 01 00 00 00 00 00  00 00 00 00 00 00 00 00  | ...............|
00 80 08 0E F0 FF FF FF  5C E7 02 00 20 00 00 00  |........\... ...|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

[redacted] offset 0x6c underlined

so from what we see boot args should be just space

" "

let's check that via sysctl

however, we can't really see space in terminal, so let's replace spaces with underscores too notice it

```
Naomis-iPhone:/fox root# sysctl kern.bootargs | sed 's/ /_/g'
kern.bootargs:__
```

confirmed, bootargs is indeed just " "

now as another test, let's try to write to those bootargs

```
memctl> ws 0xffffffff00e08406c helloworld
memctl>
[1]+  Stopped(SIGTSTP)        ./memctl
Naomis-iPhone:/fox root# sysctl kern.bootargs
kern.bootargs: helloworld
```