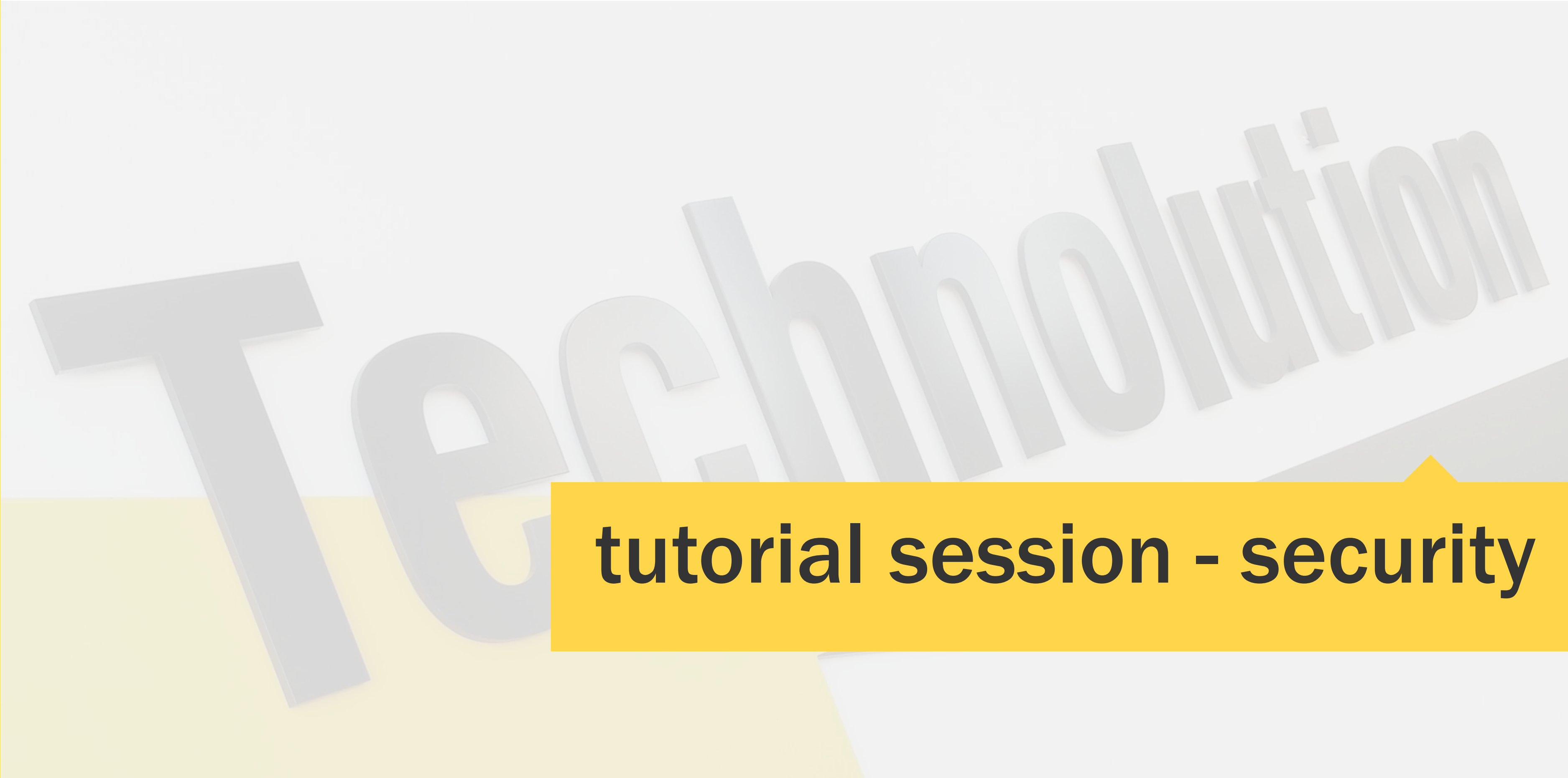


tutorial session - security

What we will do

- Do some hacking ☺
  - Why are embedded systems vulnerable?
  - A simple hack demo
  - How can RISC-V help

- FreeRTOS
  - A nice and good OS for embedded systems
  - We made RISC-V port available on FreeRTOS site
  - Give a practical starting point for your ‘next’ design

Set-up

Preconditions

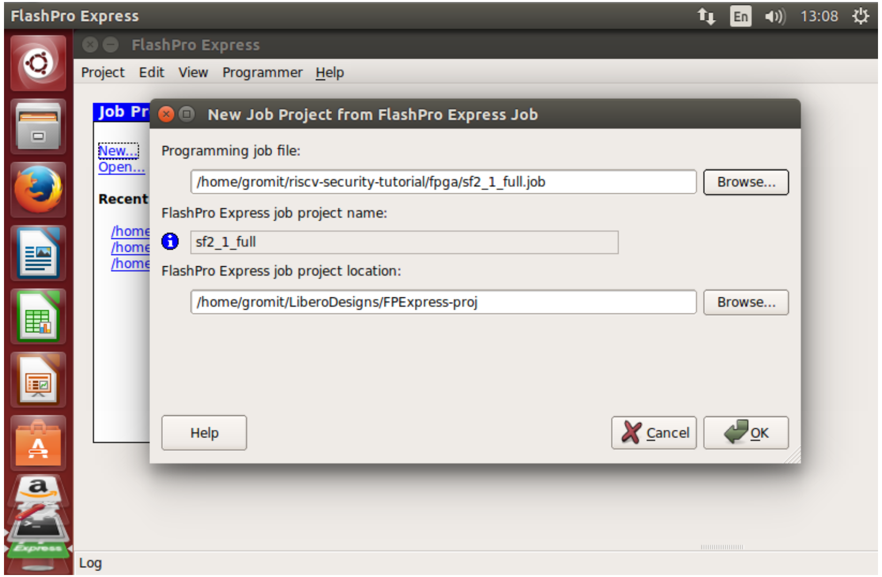
- running the tutorial VM
- cloned the tutorial archive in home dir:

```
git clone -detph=1 https://github.com/riscv/riscv-4th-workshop-tutorials.git
```

Program the FPGA

```
./start_FPExpress
```

- Select new…
- Select programming job file: fpga/sf2_1_full.job
- Press oke
- Click large RUN button



Programming job file: /home/gromit/riscv-security-tutorial/fpga/sf2_1_full.job

FlashPro Express job project name: sf2_1_full

FlashPro Express job project location: /home/gromit/LiberoDesigns/FPExpress-proj

About me

- Jonathan Hofman
  - 2003 – Master computer engineering
  - 2005 – VHDL programmer
  - 2009 – Project management
  - 2011 – Technology Manager Programmable Logic
  - 2016 – Domain architect Defense, Safety & Security

TU Delft Delft University of Technology

Technolution

- Interests
  - Mixed criticality systems
  - High assurance security systems
  - Computer architectures (RISC-V)
  - High performance real-time processing
  - Programmable logic, FPGAs, etc.

About Technolution

- Located in Gouda, The Netherlands
- Technology integrator
- 180 employees
- Active in:
    - Mobility management
    - Defense, safety & security
    - High tech industry
    - Smart energy

Our relation with RISC-V

founding member

- **Fault tolerant & secure SoC system**
  - RISC-V core, caches, busses, …
  - configurable fault resistance (lock-step, TMR, …)
  - functionality focusing on security features

→ trusted execution platform for **mixed criticality** systems

aviation, space, big science

government & defense

medical

Acknowledgements

Microsemi

SiFive

freeRTOS

NOTE:

The example exploit is **not** caused by any weaknesses in the used technologies!

It is simply due to a programming error made by the application developer.

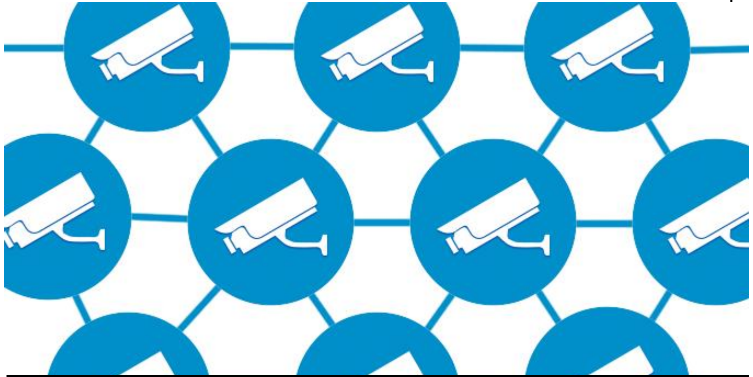
Microsemi provides FPGAs with great security features to realize high secure (embedded) applications.

Application

The downside of IoT

2016: WiFi hack disables alarm

source: theregister.co.ukv



2016: botnet powered by 25000 CCTV devices

source: threadpost.com

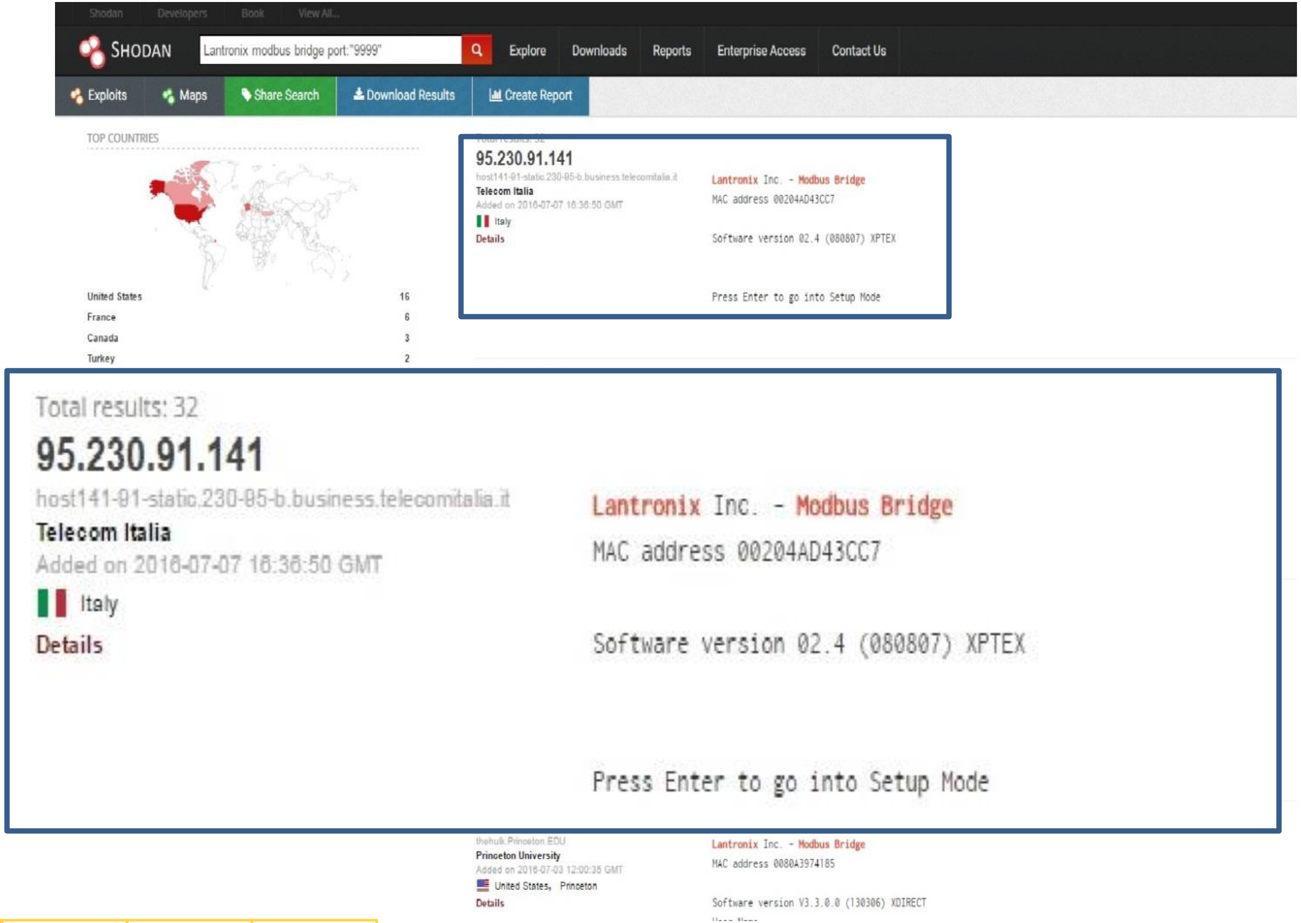
2015: kill a car remotely on the high-way

source: wired.com

2015: possible malware spread fitness tracker

source: theregister.co.uk

Search your next target



Lantronix modbus bridge port:"9999"

United States	16
France	6
Canada	3
Turkey	2

95.230.91.141
host141-91-static.230-95-b.business.telecomitalia.it
Telecom Italia
Added on 2016-07-07 16:36:50 GMT
Italy
Details

Lantronix Inc. - Modbus Bridge
MAC address 00204AD43CC7

Software version 02.4 (080807) XPTEX

Press Enter to go into Setup Mode

Total results: 32
95.230.91.141
host141-91-static.230-95-b.business.telecomitalia.it
Telecom Italia
Added on 2016-07-07 16:36:50 GMT
Italy
Details

Lantronix Inc. - Modbus Bridge
MAC address 00204AD43CC7

Software version 02.4 (080807) XPTEX

Press Enter to go into Setup Mode

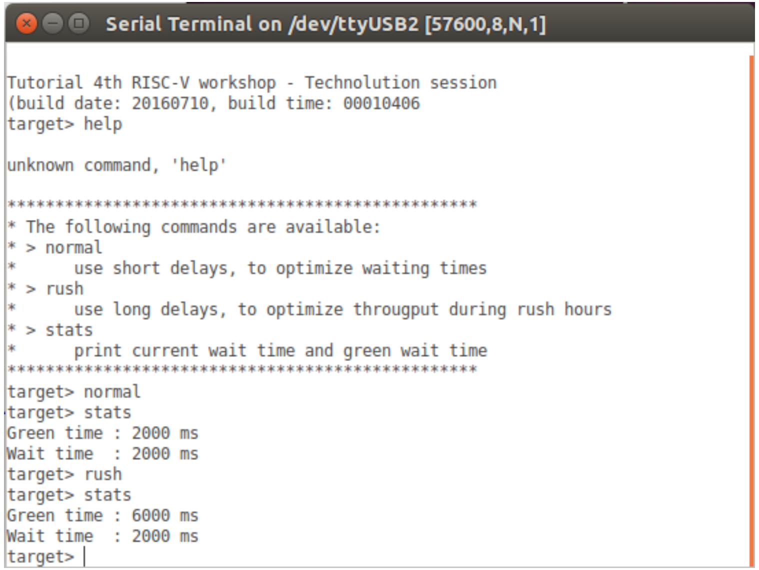
thehulk.Princeton.EDU
Princeton University
Added on 2016-07-03 12:00:35 GMT
United States, Princeton
Details

Lantronix Inc. - Modbus Bridge
MAC address 0080A3974185

Software version V3.3.0.0 (130306) XDIRECT

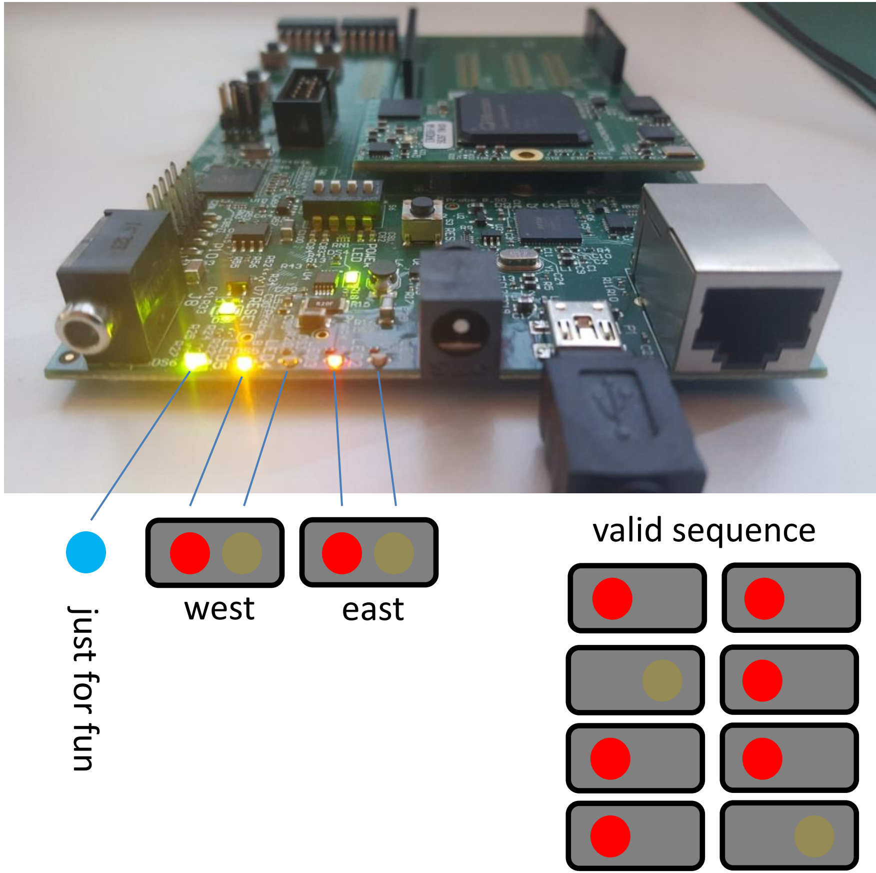
Back to our tunnel control

- Simple time based sequence
- Remote interface to control flow
    - normal
    - rush

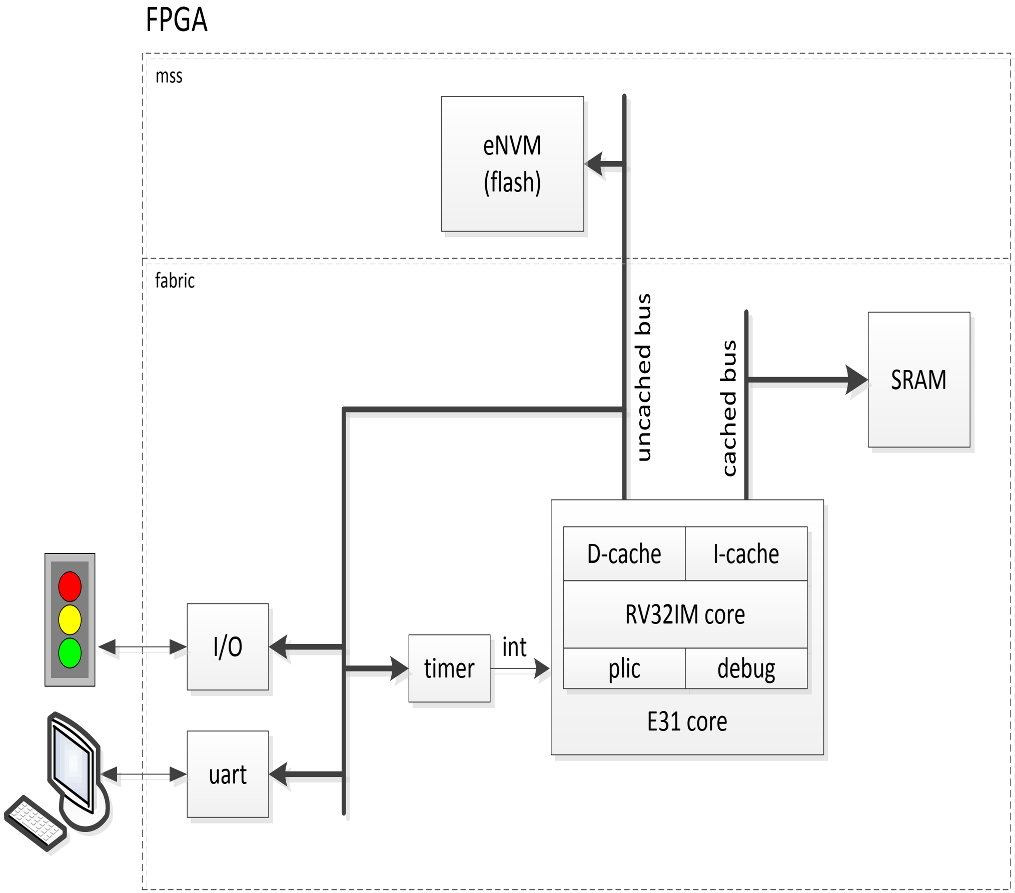


Serial Terminal on /dev/ttyUSB2 [57600,8,N,1]

```
Tutorial 4th RISC-V workshop - Technolution session
(build date: 20160710, build time: 00010406
target> help

unknown command, 'help'

**************************************************
* The following commands are available:
* > normal
*      use short delays, to optimize waiting times
* > rush
*      use long delays, to optimize througput during rush hours
* > stats
*      print current wait time and green wait time
**************************************************
target> normal
target> stats
Green time : 2000 ms
Wait time  : 2000 ms
target> rush
target> stats
Green time : 6000 ms
Wait time  : 2000 ms
target> |
```

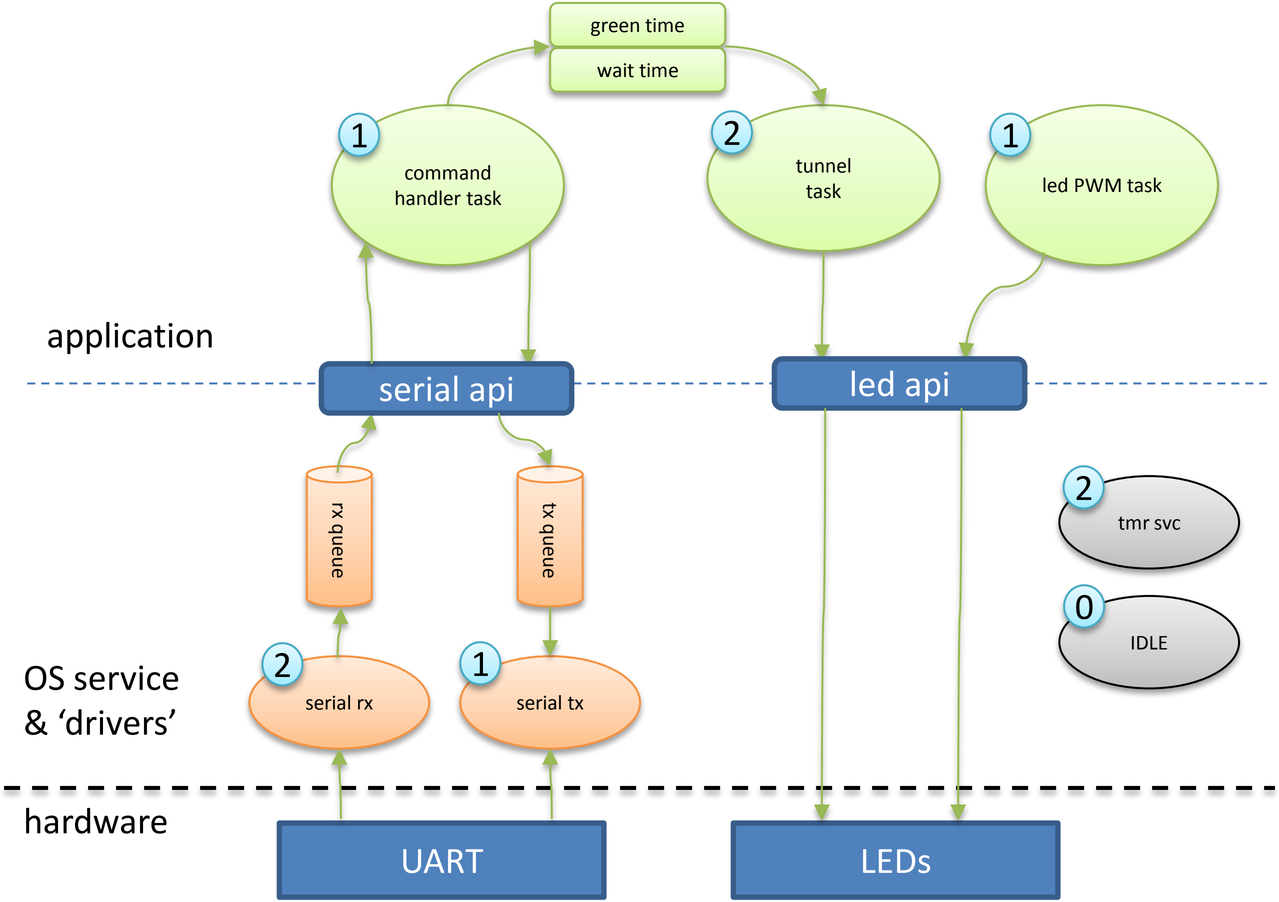


Hardware implementation



Technolution

Software implementation



1 = priority
(higher number = higer prio)

Background: FreeRTOS

Main goals

- Easy to use
- Small footprint
- Robust

Status

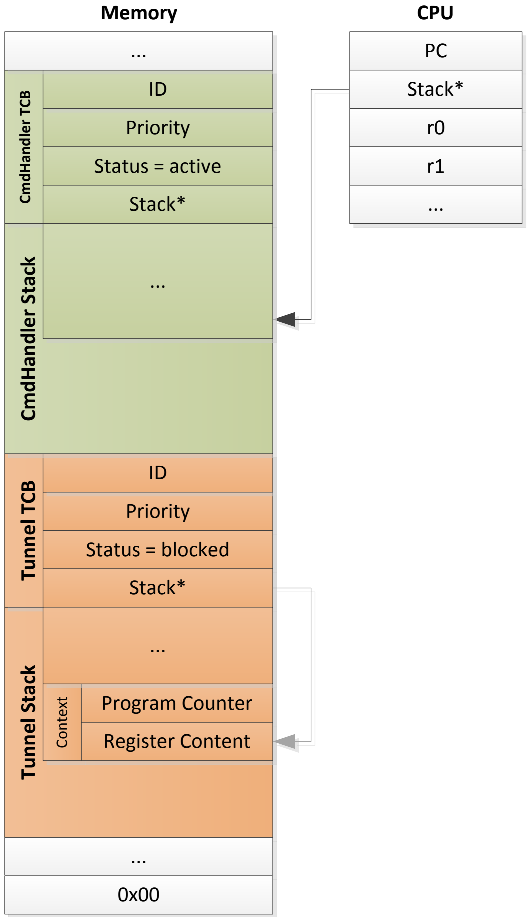
- High adaptation
- Many architectures supported
- Eco-system

Supports basic OS concepts

- Tasks & Co-routines
- Queues & Queue sets
- Semaphores, Mutexes
- Software timers
- MPU support
- Pre-emptive scheduling
- …

Background: FreeRTOS tasks memory

- Task Control Block (TCB) above stack
  - TCB on top prevents stack overflow issues
- Stack is allocated by `xTaskCreate`
- Stack overflow detection option
  → No guarantee



Hack

What makes embedded vulnerable

- Good chance we have physical access
  - use a debugger to obtain address locations
- Sacrifice one to take over the whole
- Code size is small
- Mixed open/closed source
- Security is not top of mind

Be warned:
these embedded system have access to our physical world!!

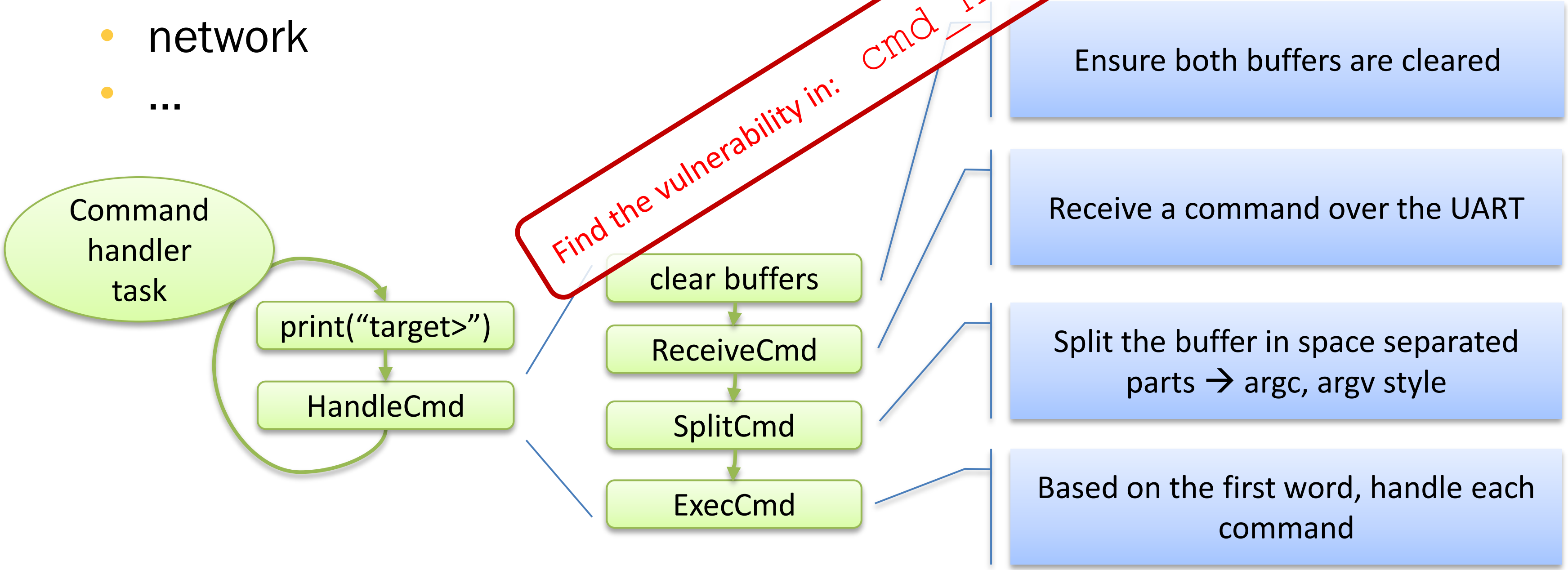
- Build the application

```
$ cd appl
$ make
```

- Open `src/cmd_hanler.c`
- Open `main.dump`

Find the vulnerability

- System vulnerabilities are typically in interaction
  - user IO
  - network
  - …



Ensure both buffers are cleared

Receive a command over the UART

Split the buffer in space separated parts → argc, argv style

Based on the first word, handle each command

Find the vulnerability in: `cmd_handler.c`

The vulnerability

```
static int ReceiveCmd(char* buf)
…
    do {
        /* increment index pointer for each character increment */
        idx++;
        if (xSerialGetChar(xComPort, (signed char *) &buf[idx], portMAX_DELAY) == pdFALSE) {
            continue;
        }
        /* echo the character back to the terminal */
        xSerialPutChar(xComPort, buf[idx], 0);
        /* handle the hit of an backspace by shifting the idx back */
        if (buf[idx] == '\b'){
            idx -= 2;
        }
        /* add some verbosity for the demo */
        if (verbose >= 2) {
            printf("buf[%d] = 0x%02x\n", idx, buf[idx]);
        }
    } while ((buf[idx] != '\n') && (buf[idx] != '\r'));
…
}
```

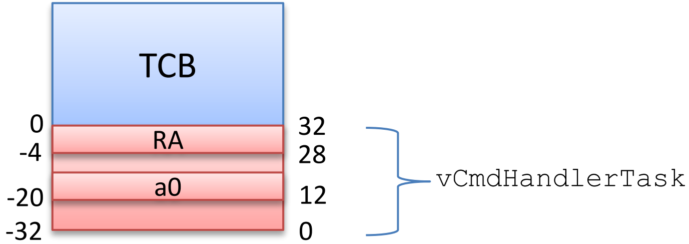
The character is always written at idx position.

We can make index negative, creating a buffer underflow

End position is not depended on buffer size, but on enter.

Stack layout

```
static void vCmdHandlerTask(void *pvParameters)
{
    (void) pvParameters;

    for (;;) {
        printf("target> ");
        HandleCmd();
    }
}
```

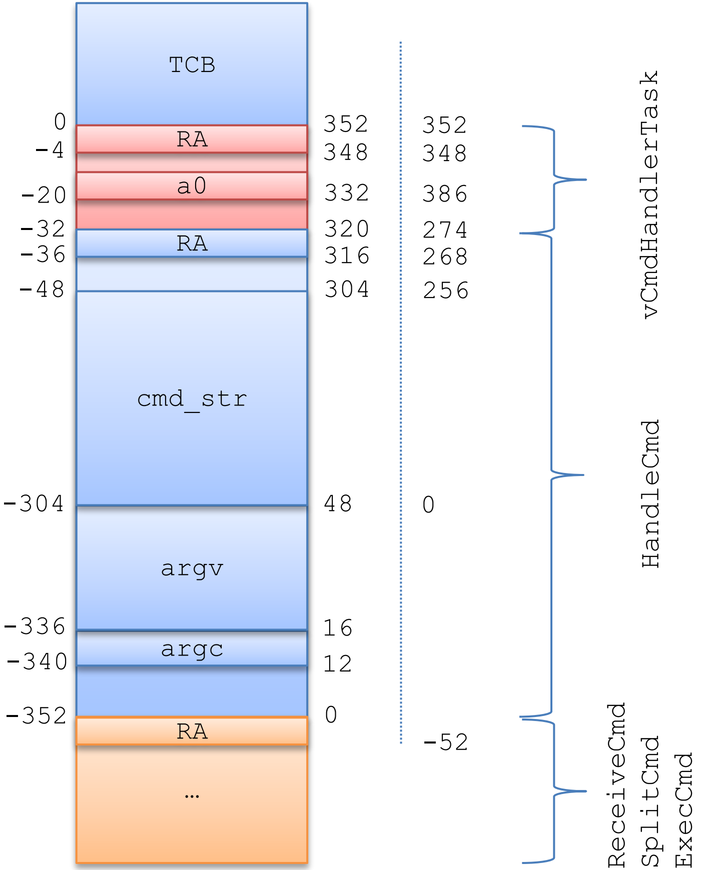


```
60026bfc <vCmdHandlerTask>:
60026bfc:    fe010113             addi      sp,sp,-32
60026c00:    00112e23             sw        ra,28(sp)
60026c04:    00a12623             sw        a0,12(sp)
60026c08:    6002b7b7             lui       a5,0x6002b
60026c0c:    15478513             addi      a0,a5,340 # 6002b154 <__rodata_start+0x2b8>
60026c10:    1a8010ef             jal       60027db8 <printf>
60026c14:    f69ff0ef             jal       60026b7c <HandleCmd>
60026c18:    ff1ff06f             j         60026c08 <vCmdHandlerTask+0xc>
```

Stack layout

```
static void HandleCmd(void)
{
    char cmd_str[256];
    char* argv[MAX_ARGS];
    int argc = 0;

    memset(argv, 0, sizeof(argv));
    memset(cmd_str, 0, sizeof(cmd_str));
    ReceiveCmd(cmd_str);
    SplitCmd(cmd_str, &argc, argv);
    ExecCmd(argc, (const char**) argv);
}
```

```
60026b7c <HandleCmd>:
60026b7c:   ec010113        addi    sp,sp,-320
60026b80:   12112e23        sw      ra,316(sp)
60026b84:   00012623        sw      zero,12(sp)
60026b88:   01010793        addi    a5,sp,16
60026b8c:   02000613        li      a2,32
60026b90:   00000593        li      a1,0
60026b94:   00078513        mv      a0,a5
60026b98:   30c030ef        jal     60029ea4 <memset>
60026b9c:   03010793        addi    a5,sp,48
60026ba0:   10000613        li      a2,256
60026ba4:   00000593        li      a1,0
60026ba8:   00078513        mv      a0,a5
60026bac:   2f8030ef        jal     60029ea4 <memset>
60026bb0:   03010793        addi    a5,sp,48
60026bb4:   00078513        mv      a0,a5
60026bb8:   a55ff0ef        jal     6002660c <ReceiveCmd>
```



Before hacking, solve this

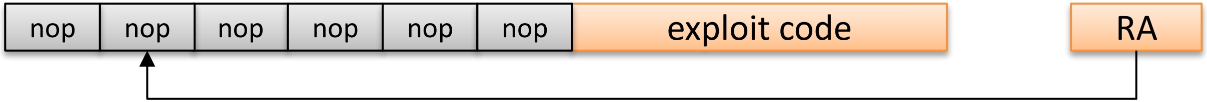
- the location of the buffer

  →using a debugger, we can dump memory and locate the stack

  dump memory

  ```
  mem 0x80003000 1024
  ```

  →we can ‘catch’ the return by a NOP train



- some characters are treated special (‘\n’,’\r’,‘\b’)

  →use escaping, de-escape at the start of the exploit code

  see exploit_init.s

- limited code size

  →use functions (like printf) in existing app

  →bootstrap

Put it all together

- Build exploit

```
$ cd exploit
$ make
```

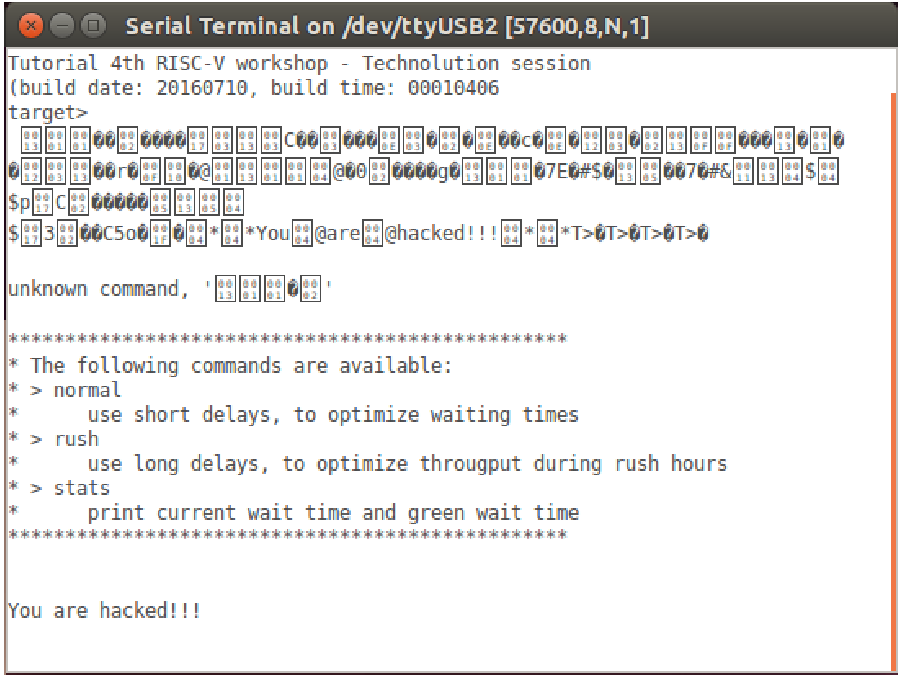
- generate the link symbols
- compile and link the exploit
- escape the binary

Lets take a look in:

- `exploit_init.S`
- `c_exploit.c`
- `link.ld`
- `exploit.ld`

- Download the exploit

Tools > Upload blob..
select “exploit/build/exploit.raw”



```
Tutorial 4th RISC-V workshop - Technolution session
(build date: 20160710, build time: 00010406
target>
 □□□��□����□□□□C��□���□□�□�□��c�□�□□�□□□□���□�□�
�□□□��r�□□�@□□□□□@�0□����g�□□□�7E�#$�□□��7�#&□□□$□
$p□C□�����□□□□
$□3□��C5o�□�□*□*You□@are□@hacked!!!□*□*T>�T>�T>�T>�

unknown command, '□□□�□'

*************************************************
* The following commands are available:
* > normal
*      use short delays, to optimize waiting times
* > rush
*      use long delays, to optimize througput during rush hours
* > stats
*      print current wait time and green wait time
*************************************************



You are hacked!!!
```

Lets bootstrap

- Build exploit_downloader

```
$ cd exploit_downloader
$ make
```

- Build the payload app

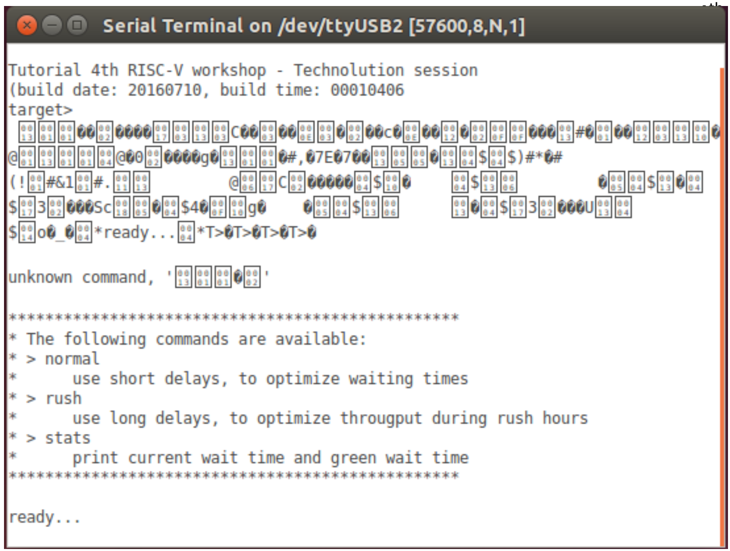
```
$ cd payload
$ make
```

- Download the exploit

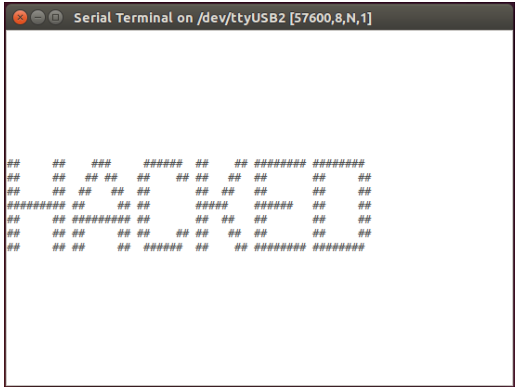
Tools > Upload blob..
select “exploit_downloader/build/exploit.raw”

- Download the payload

Tools > Upload blob..
select “payload/build/payload.bin”



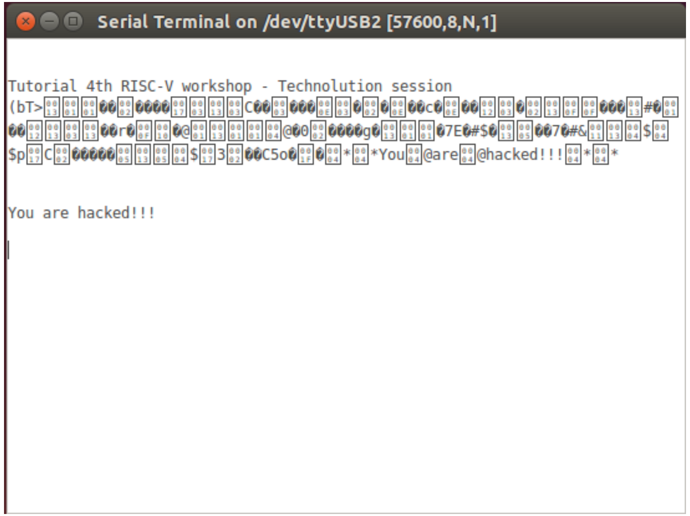
```
Tutorial 4th RISC-V workshop - Technolution session
(build date: 20160710, build time: 00010406
target>
 ▯▯▯��▯����▯▯▯▯C��▯��▯▯�▯��c�▯��▯�▯▯▯���▯#�▯��▯▯▯▯�
@▯▯▯▯▯@�0▯����g�▯▯▯�#,�7E�7��▯▯▯�▯▯$▯$)#*�#
(!▯#&1▯#.▯▯        @▯▯C▯�����▯$▯�    ▯$▯▯        �▯▯$▯�▯
$▯3▯���Sc▯▯�▯$4�▯▯g�    �▯▯$▯▯      ▯�▯$▯3▯���U▯▯
$▯o�_�▯*ready...▯*T>�T>�T>�T>�

unknown command, '▯▯▯�▯'

*************************************************
* The following commands are available:
* > normal
*      use short delays, to optimize waiting times
* > rush
*      use long delays, to optimize througput during rush hours
* > stats
*      print current wait time and green wait time
*************************************************

ready...
```



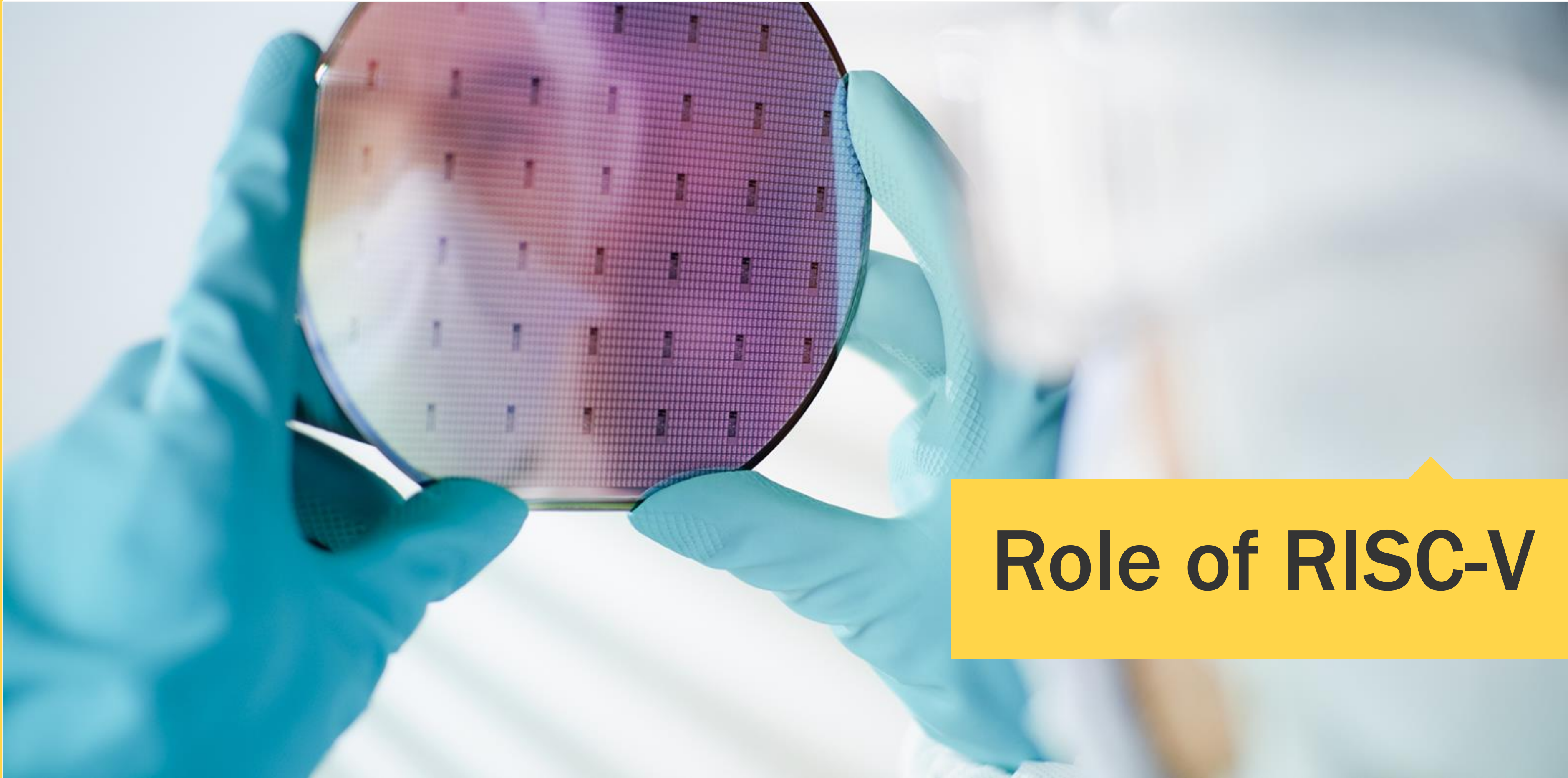
```
##     ##    ###     ######  ##    ## ######## ########
##     ##   ## ##   ##    ## ##   ##  ##       ##     ##
##     ##  ##   ##  ##       ##  ##   ##       ##     ##
######### ##     ## ##       #####    ######   ##     ##
##     ## ######### ##       ##  ##   ##       ##     ##
##     ## ##     ## ##    ## ##   ##  ##       ##     ##
##     ## ##     ##  ######  ##    ## ######## ########
```

Do it your self

- Modify your exploit builder to ‘mis-use’ exploit 2 (backspace)
- Wat would we do with this exploit?
- How can you see you have implemented it right?



```
Tutorial 4th RISC-V workshop - Technolution session
(bT>□□□��□����□□□□C��□���□□�□�□��c�□��□□�□□□□���□#�□
��□□□□��r�□□�@□□□□□@�0□����g�□□□�7E�#$�□□��7�#&□□□$□
$p□C□�����□□□□$□3□��C5o�□�□*□*You□@are□@hacked!!!□*□*


You are hacked!!!
```



Role of RISC-V

Known options

- Memory execute protection
  - does not prevent ‘return to C-lib’attacks
  - does not prevent data only attacks
- How can RISC-V help?
  - allows to create ‘low-end’ micro with security focus (e.g. MPU)
  - micro architecture features
    - e.g. labeling (lowRISC, draper)
  - special instructions?
    - ‘hardware’ stack canaries

Another approach

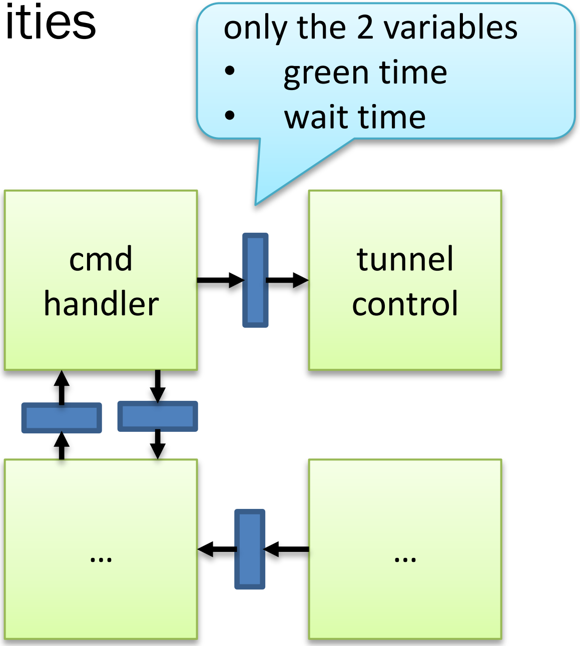
- mixed criticality
  - ensure non-interference between functionalities

→ **loosely** coupled cores

  - ease security or safety evaluation
  - simple interfaces with low attack potential

- requires flexible system configuration
  - FPGAs
  - create virtual processors (pipeline slots)

  → notion of multiple hw-threads in spec


only the 2 variables
- green time
- wait time

Wrap-up

Conclusion

- Security in embedded system is important
    - hacks can have significant effect in physical world!!

- RISC-V can help making embedded system more secure
    - micro architecture features
    - instruction set extensions
    - SoC system level features

- Technolution creates a RISC-V environment for security & safety
- We want to stimulate a secure RISC-V eco-system

→ Any questions, suggestions or ideas? Contact us!

Technolution

/ the right development

Jonathan Hofman

jonathan.hofman@technolution.nl

+31 6 10 782 782

in linkedin.com/in/jonathanhofman