Date: 04/04/24
Merve and Luciano have selected two schools to attack SMCAA, and TTU
Luciano wants to use TTU because he thinks the Buffer Overflow attack can be performed by modifying the python file called generate.py to send a malicious boot message to cause unexpected behavior.

Date: 03/28/24
Monday at 10:30 we will get the code released and we will be able to start working on it.
Post quantum side channel challenge
Test Vector Leakage Assessment
DPA was an older technique

Date: 03/14/24
Header file needs to be edited to use the wolfssl.

Date: 02/29/24
Two devices not connecting on Luciano computer
Could not figure out how to have two debuggers running at same time for component and ap.

possible solution:
replacing ID in configs to the boards connected to the computer and run openocd again with the new configs.

What we tried:
replacing all the IDs in the configs

Solution on merve computer:
component is port 4402, and ap is port 4401.

```
openocd -f ./debug/debug/comp1.cfg
```

Date: 02/21/24
Current task:
find buffer overflows on the devices.
We are in security phase
use bufferoverflow with gets function.

```
openocd -f device1.cfg
```

Daniel uses a different port for the component but i don’t know why he does because i used only one port and it worked.

Go to the MSSDK folder and look at libraries that could be useful in the code. The ECC folder could protect the code.

use get_s to specify the size of the buffer.
The project.mk file has constants that can be made in the whole code.

```
gcc bufferoverflow.c -o overflow -fno-stack-protector -z exestack -no-pie
```

```
poetry run arm-none-eabi-gdb
```

NOTES

Commands:

(1) git clone https://github.com/mitre-cyber-academy/2024-ectf-insecure-example

Installed extensions:
C/C++
Python
Cortex-Debug

Commands after running (1):

```
nix-shell
poetry shell
```

After first time run:

```
poetry install
```

- Nix shell is the isolation environment it allows you to seperate packages from the main system. Poetry shell is the dependency environment for the Python programming language; it allows dependencies installed from Python packages to be run after installation.

To update (1) if there is any more security updates run:

```
git pull origin release
```

- release is the name of the branch that the code from the insecure example is from. This command get the latest version of the example.

In the component folder we have 3 .c file:

- Board_link.c
- Component.c
- imple_i2c_peripheral.c

Top left button is the reset button

Bottom right button is the central buttons

Press both buttons at the same time to reset and release the right button to see blue light flash;

Left board does not have any light but has red blink when it is connected.

To see usb port name on MAC: `ls /dev/tty.*`

Board commands:
build the global secrets

```
ectf_build_depl -d .
```

build ap:

```
ectf_build_ap -d . -on ap -od build -p 123456 -t 1234567890abcdef -c 1 -ids "0x11111124" -b "Boot message"
```

Run ls build to see all the files you built already, such as:

```
ap.bin  ap.elf  ap.img
```

build the component:

```
ectf_build_comp -d . -on c11111124 -od build -id 0x11111124 -b “Component” -al “Florida” -ad “2/3/24” -ac “Luciano”
```

Run ls build to see all the files you built already, such as:

```
ap.bin  ap.elf  ap.img  c11111124.bin  c11111124.elf  c11111124.img
```

This method is how i got it working:
The side that has the AP written in the black pen is the actual AP only have the AP plugged into the computer at one time. I put the AP into update mode by doing the method also i used the SW0 in the blue pen to see if anything would show up, Next while it seemed in update mode flash the insecure.bin to the board. Once flashed it will now turn the blue LED off, next do the UPDATE MODE and ectf update with the newly created ap.img. do ectf_list -a /dev/ttyACM0 and you will see the P>0x11111124 and SUCCESS. Next unplug the AP from the computer and plug in the component board. Now put the component into UPDATE MODE, and flash the c11111124.img using the ectf_update tool. Next plug both the ap and the component into the computer at the same time and do the ectf_list tool on /dev/ttyACM1 to see the P and the F status in the application processor

UPDATE MODE the component board, press both buttons and release the button to the bottom right

UPDATE MODE on the ap board, press both button and release the button on the top left

BEFORE RUNNING ECTF UPDATE RUN THE LIST TOOL TO FIND THE AP

ectf_list -a /dev/ttyACM0 - check output, if output does not work i think its a component, do not write an ap to this if no output is detected.
ectf_list -a /dev/ttyACM1 - output detected

MAC OS
4 - ectf_update -infile build/ap.img –port /dev/tty.usbmodem1302
5 - ectf_update -infile build/comp.img –port /dev/tty.usbmodem1102

LINUX
According to Fritz he said that the device does not matter to flash it

```
sudo chmod 777 /dev/ttyACM*
ls /dev/ | grep ttyACM
```

This finds the application processor maybe ?

```
ectf_list -a /dev/ttyACM1
ectf_list -a /dev/ttyACM0
```

```
ectf_update --infile build/ap.img --port /dev/ttyACM1
```

LINUX

On MAC
/dev/tty.usbmodem1302 - Application Processor
/dev/tty.usbmodem1102 - Component

TEAM PLAN
ChatGPT the code and try to figure out the vulnerabilities in the code:
https://chat.openai.com/share/b579020f-cd9e-4760-b083-4ca29e8c95a6
This example chat is a clue of what we need to secure.
Go to the component directory and try to secure the .c files.
Some ideas are to find insecure functions that can be replaced such as “memcpy” function because this is insecure and can lead to buffer overflow.
use TRNG true random number generator
use wolfssl or openssl

BOARD LINK CODE(board_link.c) NOTES
Question

Why is this code not secure? in the memcpy ? `uint8_t wait_and_receive_packet(uint8_t* packet)`
`{`

```
        while(!I2C_REGS[RECEIVE_DONE][0]);

        uint8_t len = I2C_REGS[RECEIVE_LEN][0];
        memcpy(packet, (void*)I2C_REGS[RECEIVE], len);

        return len;
}
```

Answer

The code you provided is not inherently secure because it lacks proper input validation and error handling. Here are a few reasons why this code may be considered insecure:

1. Lack of input validation: The code does not validate the `packet` pointer to ensure it is not `NULL` or that it has sufficient memory allocated to hold the received data. This can lead to buffer overflows or other memory-related vulnerabilities if an attacker provides a malicious or unexpected input.

2. No bounds checking: The code does not perform any bounds checking on the received data length (`len`) against the size of the `packet` buffer. If the received data length exceeds the size of the `packet` buffer, it can result in a buffer overflow, leading to memory corruption or other security vulnerabilities.

3. Inadequate error handling: The code does not handle errors that may occur during the I2C communication or the memory copy operation. If there are errors in the I2C communication or if the `memcpy` operation fails, the code may not provide appropriate error handling or recovery mechanisms, potentially leading to undefined behavior or security vulnerabilities.

To make this code more secure, you should consider implementing proper input validation, bounds checking, and error handling mechanisms. Additionally, it is important to follow best practices

Question

Why is this code not secure ?

```
void send_packet_and_ack(uint8_t len, uint8_t* packet) {
        I2C_REGS[TRANSMIT_LEN][0] = len;
        memcpy((void*)I2C_REGS[TRANSMIT], (void*)packet, len);
        I2C_REGS[TRANSMIT_DONE][0] = false;

        // Wait for ack from AP
        while(!I2C_REGS[TRANSMIT_DONE][0]);
        I2C_REGS[RECEIVE_DONE][0] = false;
}
```

Answer

The code you provided may be considered insecure due to the following reasons:

1. Lack of input validation: The code does not validate the `packet` pointer to ensure it is not `NULL`. If the `packet` pointer is `NULL`, attempting to copy the data using `memcpy` will result in undefined behavior, potentially leading to crashes or security vulnerabilities.

2. No bounds checking: The code does not perform any bounds checking on the length of the `packet` data (`len`) against the size of the destination buffer in `I2C_REGS[TRANSMIT]`. If the `len` exceeds the size of the destination buffer, it can lead to buffer overflows and memory corruption.

3. Incomplete error handling: The code does not handle errors that may occur during the `memcpy` operation. If the `memcpy` operation fails due to a memory access violation or other reasons, the code does not provide appropriate error handling or recovery mechanisms.

4. Lack of security measures: The code does not implement any security measures, such as data encryption or integrity checks. This can make the transmitted data susceptible to eavesdropping, tampering, or other malicious attacks.

To enhance the security of this code, it is advisable to add input validation, bounds checking

Question

Why is this code not secure?

```
i2c_addr_t component_id_to_i2c_addr(uint32_t component_id) {
        return (uint8_t) component_id & COMPONENT_ADDR_MASK;
}
```

Answer

The provided code appears to be a simple function that converts a `component_id` of type `uint32_t` to an `i2c_addr_t` value. However, based on the code snippet alone, it is difficult to definitively determine if the code is insecure or not.

To assess the security of this code, we need to consider the wider context and potential security requirements within the system. Here are a few reasons why this code might be considered insecure:

1. Lack of input validation: The code does not perform any input validation or sanitization on the `component_id` parameter. Depending on the specific security requirements, it may be necessary to validate the input to ensure it falls within an expected range or format. Without proper input validation, the code may be vulnerable to unexpected or malicious input values.

2. Possible information leakage: If the `component_id` is derived from user input or other external sources, returning the lower 8 bits of the `component_id` (`component_id & COMPONENT_ADDR_MASK`) could potentially leak information about the underlying structure or implementation details of the system. This leakage of information could be exploited by an attacker to gain insights or mount targeted attacks.

To enhance the security of this code, it is recommended to perform proper input validation and

Question

Why is this code not secure?
```
int board_link_init(i2c_addr_t addr) {
        return i2c_simple_peripheral_init(addr);
}
```

Answer

The provided code snippet appears to be a function called `board_link_init` that initializes a board link using the `i2c_simple_peripheral_init` function with the `addr` parameter. From this code snippet alone, it is difficult to identify any specific security vulnerabilities.

However, to assess the security of this code, it is important to consider the implementation details of the `i2c_simple_peripheral_init` function and the wider context in which this code is used. Here are a few potential security concerns to consider:

1. Insecure implementation of `i2c_simple_peripheral_init`: If the `i2c_simple_peripheral_init` function itself has security vulnerabilities, such as improper handling of user input or insufficient data validation, then calling this function could introduce security risks.

2. Lack of input validation: The code snippet does not perform any input validation or sanitization on the `addr` parameter. Depending on the specific security requirements, it may be necessary to validate the input to ensure it falls within an expected range or format.

3. Missing error handling: The code snippet does not include error handling or error-checking mechanisms. If the `i2c_simple_peripheral_init` function encounters an error, the code does not provide any means to handle

Question

Should the board_link.c be secured? Why or why not?

Answer

Whether the `board_link.c` file should be secured or not depends on the specific context and requirements of your application. However, in general, it is good practice to consider security measures for any code that interacts with external devices or networks.

Here are a few reasons why you might consider securing the `board_link.c` code:

1. Protection against unauthorized access: If the `board_link.c` code deals with sensitive data or provides access to critical functionality, securing it can help prevent unauthorized access or manipulation. This may involve implementing authentication mechanisms, access controls, or encryption to ensure that only authorized entities can interact with the board link.

2. Mitigating potential vulnerabilities: Even if the `board_link.c` code itself does not contain obvious security flaws, it is important to consider potential vulnerabilities that could be exploited. This may include input validation, sanitization, and bounds checking to prevent buffer overflows, injection attacks, or other common security issues.

3. Data integrity and confidentiality: Depending on the nature of the data being transmitted over the board link, it may be important to ensure data integrity (preventing tampering) and data confidentiality (preventing unauthorized access to sensitive information). Implementing encryption or other cryptographic measures can help protect the privacy and integrity of the data.

HOW TO RESEARCH VULNERABILITIES:
Go to cplusplus reference and search the vulnerabilities
show which functions can be replaced with the other functions.

WOLFSSL PART OF DESIGN

- make sure communications are secure against man-in-the middle attack.
- i2c needs to be secure over the connections.
- encrypt the communications
- ensure all devices are connected, create Root CA Certificate.
  Only accept encrypted communications from trust entity.

Pull header files from the source thats what Daniel Owens says.
i2C example changes from wolfSSL
02/15/24 notes
In project.mk DEBUG=1 to connect to AP board to debug the AP board. Only have one board connected at once.

After setting DEBUG to 1 you need to run:

```
ectf_build_ap -d ./ -on ap --p 123456 -c 1 -ids "0x11111124" -b "Test boot message" -t 0123456789abcdef -od build
```

After that you need to run in a **new terminal:**

```
openocd -f interface/cmsis-dap.cfg -f target/max78000.cfg -c "init"
```

```
brew install arm-none-eabi-gdb
```

You need to install arm-none-eabi to your computer and, open the .vscode folder and find launch.json on JSON file edit these parts:
Add “gdbTarget”: “localhost:4444” to json file in launch.json.
Add “executable” to ap.elf
Add “servertype” to external not jlink

You can start run and debug by clicking the run and debug in vscode

Reminder - Luciano needs to instal arm-none-eabi to debug on his computer.
To run the shell:

```
nix-shell -p nix -I nixpkgs=channel:nixpkgs-unstable --run "nix --version"
```

Questions: 02.07.2024

1. We don’t know anything about the registry . like what are r0 , r1, …., r12? Assembly ?
   (gdb) info registrar
   Slack video: announcement : title: debuggers

2. How much of the debugger we need to know, like stepping thru the code , sending the break points, do we need to use these info in the class or all are just theory
   Slack video: announcement : title: debuggers

HOW TO USE DEBUGGER:
create a .vscode folder, and a launch.json file inside with:

```
{
   "version": "0.2.0",
   "configurations": [
       {
           "name": "Component Board",
           "cwd": "${workspaceFolder}",
           "executable": "./build/c11111124.elf",
           "request": "launch",
           "type": "cortex-debug",
           "gdbTarget": "localhost:3333",
           "runToEntryPoint": "main",
           "servertype": "external"
       },

   {
       "name": "Application Processor",
       "cwd": "${workspaceFolder}",
       "executable": "./build/ap.elf",
       "request": "launch",
       "type": "cortex-debug",
       "gdbTarget": "localhost:3333",
       "runToEntryPoint": "main",
       "servertype": "external"
   }
]
```

Now to use the debugger, just set a breakpoint in any of the files by clicking on the red dot on the side,in board_link.c or component.c.

GUESSING INFORMATION: