CSAW ESC Final - Report

Theo DUFOUR
Hackcess
IUT ROANNE
Roanne, France
theodufour@hackcess.org

Tom MALOSSE
Hackcess
IUT ROANNE
Roanne, France
tommalosse@hackcess.org

Naïl BIBIMOUNE
HACKCESS
IUT ROANNE
Roanne, France
mailbibimoune@hackcess.org

Mateo FERREIRA
Hackcess
IUT ROANNE
Roanne, France
mateoferreira@hackcess.org

I. Introduction

A. Context

This report follows the qualification of our team, Hackcess, at the CSAW ECS 2023 final. We had the chance to be announced as a finalist on October 3rd and subsequently received the Arduino board and these various components, thus signing the start of the competition.

B. Objectives

We will present in this report our achievements of the various challenges that have been proposed and spread over 3 different weeks, the difficulty of these being increasing.

C. Hardware and Tools

Here is a list of our equipment :

- *Elegoo UNO R3*
- *PCF8574 (GPIO Expander)*
- *Buzzer*
- *Keypad*
- *Haptic (Vibration motor)*
- *Microphone*
- *Relay*
- *ESC board*
- *8 Segment display*
- *Logic analyzer (24MHz, 8 channels)*

II. Challenges week 1

A. All White Party

The first challenge was called All White Party and here is its description:

“You and your friends have just arrived at the exclusive Hollywood "All-White Party", but you're missing invitations. You found a missing badge outside the entrance gate, but after scanning at the gate, the security system is asking you for a username and 10-digit password PIN credentials. Can you uncover the secret passcode, blend in with the glamorous crowd, and find a way inside the party to experience the glitz and glamor in a reasonable amount of *time*.”

What we notice quickly is the last word of the description, time, which is written in italics, as if to accentuate the latter...

First, we had to inject the .hex into the arduino, for that we used avrdude. It's the same procedure for all competition challenges.

```
avrdude -v -V -patmega328p -carduino -P COM5 -b115200 -D -Uflash:w:
C:\Users\User\Desktop\COURSIUT\HACKESS\CSAW\AllWhiteParty.hex:i
```

Fig. 1. Command to inject hex code

After injecting the hex code into the arduino, we connect via the serial interface and we arrive at a username request.

```
14:20:17.045 → /*****************************************************************************/
14:20:17.045 → Challenge: All White Party
14:20:17.045 → /*****************************************************************************/
14:20:17.078 → Welcome!! Based on our records, your account has been located.
14:20:17.078 → Enter account username (over serial):
14:20:33.241 → Invalid username. Please try again:
```

Fig. 2. Serial display

With this word, “time”, as our only clue, we have therefore reflected on attacks related to time. The one that came to mind is that of response time changing according to input.

The username would be processed letter by letter, and if the first letter is good, it checks the next one and so on, so if we send 2 letters, and the first one is good, the processing time will be longer, since the arduino will have analyzed 2 letters and not only one.

So we imagined a brute force attack to realize our hypothesis, testing all the letters of the alphabet in lower and upper case, starting with "Aq", "Bq", "Cq"... and once the letter that takes the most time to be processed was found. For example, if the first letter is "c", we continue, "cAq", "cBq", "cCq"...

```python
import serial, string
import time

def start_chronometer():
    start_time = time.time()
    return start_time

def stop_chronometer(start_time):
    end_time = time.time()
    elapsed_time = end_time - start_time
    return elapsed_time

board = serial.Serial("/dev/cu.usbmodem1101", 115200)
line = board.readline()

while line != b"Enter account username (over serial): \x0f\n":
    print(line)
    line = board.readline()

chars = list(string.ascii_lowercase + string.ascii_uppercase)

def find_usr(base=''):
    times = {}
    for c in chars:
        usr = bytes(base, 'utf-8') + bytes(c, 'utf-8') + b'q'
        print(f"[Attempt] username: {usr}")

        stime = start_chronometer()
        board.write(usr)
        resline = board.readline()
        while resline != b'Invalid username. Please try again:\x0f\n':
            resline = board.readline()
        etime = stop_chronometer(stime)
        times[c] = etime
        print(f'[{etime}] Response: {resline}')
    sorted_items = sorted(times.items(), key=lambda item: item[1], reverse=True)
    found_char, _ = sorted_items[0]
    current_usr = base + found_char
    print(f'[>] Current username: {current_usr}')
    return find_usr(base + found_char)

find_usr()
```

Fig. 3. Code to brute force username

After a few minutes, we got our username: **Barry**

```
20:50:25.627 → Welcome!! Based on our records, your account has been located.
20:50:25.627 → Enter account username (over serial):
20:50:51.723 → 10-digit MFA code sent to your phone. Enter 10-digit Pin on Keypad.
20:50:55.671 → 0000000000
20:50:57.842 → Password SHA does not match:
20:50:57.842 → 32 117 74 65 206 Please try again
```

Fig. 4. Serial output after finding the username

Now we need a password.

Our only new index is this number: 32 117 74 65 206

We tried to use a technique similar to the previous one, using the response time of the arduino to find 1 to 1 the code, without success.

We did various searches with this number, was it a number, a hidden message, we found nothing conclusive. So we passed the first stage of the challenge, but remain stuck here.

To eliminate this side channel attack, make sure that the username check is performed on the entire username entered directly, and not letter by letter.

B. Bluebox

This week's second challenge is called: Bluebox and here's its description :

“In a secret underground lair, you and your team uncover the hardware for the legendary blue box hack. To unlock the secrets of the lair, you must decode telephone frequencies and recreate the iconic audio tones in order to reveal the flag. It's time to unearth the blue box technological history before the authorities arrive to shutdown your operation”

First, we did some blind tests to find out how the challenge worked.

We noticed that when we pressed the keys to enter the right combination, each key emitted a certain audible frequency.

We then realized that the original message being played was simply the reproduction of the keys being played.

From then on, we recorded each sound of each key (we had 3 attempts to record the different sounds, the 4th playing a new sound, it was not possible to record the sound).

Next, we recorded the sequence, then at each intonation we compared our recordings of each key to note the associated number.

From there, we sent off our sequence, but we didn't expect to receive a second, longer sequence of 8 intonations (which was originally 4).

We repeated the same process after a few unsuccessful attempts, then arrived at the flag.

```
17:07:38.132 → /********************  YOU BEAT THE CHALLENGE!!!   ********************/
17:07:38.132 → Congrats! Please add the flag to the report...
17:07:38.165 → Flag: B339B009
17:07:38.165 → /********************  Congrats!!! ********************/
```

Fig. 5. Serial output of the flag

Challenge Flag: "**B339B009**"

III. Challenges week 2

A. Operation SPItFire

For the first of the 2nd week we have a challenge called "Operation SPItFire" :

“Amid the digital battleground, you, an accomplished spy, are assigned the mission of deciphering an intricate maze of wire traffic acquired from the mysterious hacker collective, SPItFire. To assist your efforts, your remote team has gained access to a device linked to one of SPItFire's surveillance cameras, allowing you to clandestinely exchange messages. Your objective: find out how to communicate with the security camera, and acquire the coveted password flag that can be used to infiltrate their security footage.”

When we started the challenge, we observed the emission of sounds and the illumination of the LED on the Elegoo UNO R3 relay board. Initially , we thought about the possibility that it might be Morse code, but we ultimately opted to use a frame analyzer because it could be a binary code or something similar so use it appeared to be a more straightforward and efficient method.

Here’s how we proceeded :

Step 1 : Frame Analysis with a Logic Analyzer

For Operation Spitfire , we initiated by analyzing the frames using a logic analyzer. Here’s a picture of the wiring :

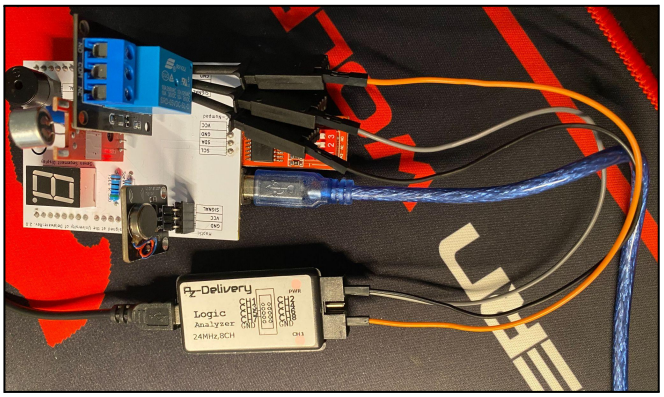

Fig. 6. Measurement wiring with logic analyzer of relay signal

This step provided us a detailed view of the communication between the components of the system under investigation.

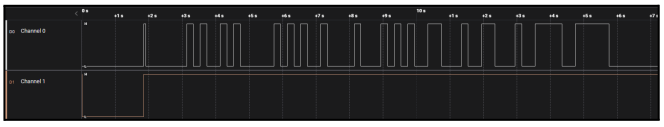

Fig. 7. Logic analyzer output on relay signal pin

Step 2 : Decomposing the “Hello” Frame into 8-Bit Packets

After capturing the frames, we extracted the “Hello” frame for a more in-depth analysis. This frame was represented as binary sequence, as follows :

10100101 00000101 01001000 01000101 01001100 01001100 01001111 00111110

We segmented this binary sequence into 8-bit packets,which allowed us to identify each part of the “Hello” frames as follows:

- Header : 10100101
- Length : 00000101 (5 characters)
- H : 01001000
- E : 01000101
- L : 01001100
- L : 01001100
- O : 01001111
- CRC-8: 00111110

This decomposition enabled a more detailed analysis of the frame, helping us understand its structure and the date it conveyed .

Subsequently, we realized that we needed to send the word ‘FLAG’. We decomposed it into binary, but we initially faced uncertainty about how to calculate the checksum. To address this, we began by developing a Python program for a brute-force approach.

```python
import serial
from chepy import Chepy

board = serial.Serial("/dev/cu.usbmodem1101", 115200)

base = "10100101000001000100011001001100010000101000111"
count = 0

for i in range(256):
    count += 1
    binvar = format(i, "08b")
    bin = base + binvar
    mess = Chepy(Chepy(bin).from_binary().o).to_hex().o
    print(f"[Attempt {count}] {bin} | {mess}")

    resline = board.readline()
    while resline != b'Please send the message "FLAG" in hex (over serial):\r\n':
        print(resline)
        resline = board.readline()

    board.write(mess)
    fline = board.readline()

    if "Sending" in str(fline):
        print(fline)
        break
    elif "Error" in str(fline):
        print(fline)
    else:
        print("[Exception] An unexpected output was detected!")
        print(fline)
        print(board.readline())
```

Fig. 8. CRC-8 brute force program

Later, through a more efficient second method, we learned how to calculate the CRC-8 checksum using the hexadecimal representation of the “FLAG” with the website https://crccalc.com/ :

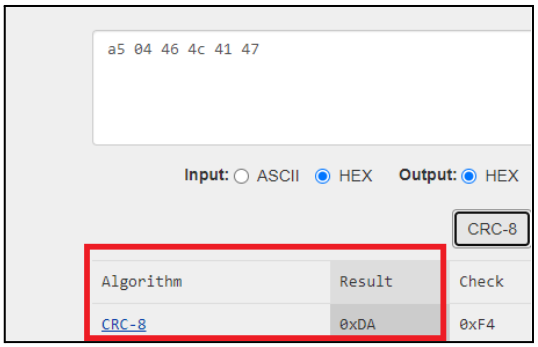

Fig. 9. CRC-8 automatic calculator

The hexadecimal sequence “a5 04 46 4c 41 47 represents a data frame. In this sequence, “da” is the calculated checksum that we discovered through a brute-force approach and the website. By sending the complete hexadecimal frame, including “da”, the system was able to verify and authenticate the data. As a result, the system interpreted the data correctly, and the decoded message or flag was **“SPyBRUNd”**. The successful verification of the checksum “da” allowed for the accurate retrieval of the intended message from the data frame. The binary representation of “SPyBURNd” is as follows:

- S: 01010011
- P: 01010000
- y: 01111001
- B: 01000010
- U: 01010101
- R: 01010010
- N: 01001110
- d: 01100100

To mitigate this side-channel attacks at the checksum level, consider using checksum algorithms resistant to such attacks. In addition, it’s to encrypt the message to prevent

unauthorized access. Keeping the message in plaintext can make it easier for potential attackers to exploit it, so encryption should be a crucial part of securing the data and therefore not to leave it clear.

Challenge Flag: "**SPyBRUNd**"

B. *czNxdTNuYzM*

Last challenge of this second week, "czNxdTNuYzM" a rather uncommon name at first glance with description:

“A cryptic dance of numbers unfolds before the eyes. Can you harmonize with the rapid rhythms of this challenge? In this realm of numbers, a symphony can conquer even the swiftest of mysteries.“

By launching this challenge, a series of numbers scrolled very quickly on the screen of the arduino, so our first attempt was to record with our phones this suite, and then watch slowly the numbers received.

We unfortunately could not get anything out of it.

Another attempt was to try to communicate with the screen, to read exactly the data that was coming, which also did not work.

IV. Challenges week 3

A. *Sock and Roll*

We did not take the time for this challenge, being focused on the others, so we have nothing to present.

B. *Vender Bender*

The last challenge in this competition is "Vender Bender":

“You roll up to that vending machine, and it's making a soft hum, like a well-tuned engine, promising you a sweet snack for your taste buds. You eyeball the colorful snacks, deciding if you want them chips for a salty crunch or that chocolate bar for a sweet fix. You've got some coins clinking in your hand, ready to drop 'em in and get that engine running. You pause, and think if there is a better way... a free way. Maybe if you trigger an error before the snack is dispensed, you can get your money back? Can you pump the brakes when you hear the gears whirring, and make off with your snack like it's a freshly greased wrench? Then let's take those taste buds on a delicious test drive.”

The first clues that enabled us to solve this challenge were the part of the engine noise mentioned in the challenge description and the message sent to the arduino's serial monitor: "After credit is received, send "ERR" to jam the motors.".

```
23:21:21.050 -> /*****************************************************************************/
23:21:21.050 -> Challenge: Vender Bender
23:21:21.050 -> /*****************************************************************************/
23:21:21.050 -> After credit is reciever, send "ERR" to jam the motors.
23:21:23.584 -> Motor movement SUCCESS. Snack was dispensed for $2. Insert another credit for a new snack.
23:21:28.589 -> After credit is reciever, send "ERR" to jam the motors.
23:21:31.092 -> Motor movement SUCCESS. Snack was dispensed for $2. Insert another credit for a new snack.
```

Fig. 10. Serial output on monitor start-up

Then we had to deduce the right timing to send the message to jam the engine. For this step, we carried out several unsuccessful tests such as spamming the "ERR" message through serial, sending the message during the engine noise generated by the relay and also when the engine (relay) was not active. We also connected the relay pins to a logic analyzer to get a better view of the signal emitted by the relay. With the help of these various tests, we finally found the right timing, i.e. the precise moment when the motor noise was triggered.

After having successfully jammed the motor for the first time, the serial monitor sends us back a message telling us that the engine has been successfully jammed and that it's on its first retry of 5 attempts. This is a clear indication that we need to jam the engine 5 times in a row to succeed in this challenge.

This part of the challenge was the most complex, given that the intervals between each of the 5 motor jams were different, and some could be very short.

We first tried to make a python script that would send the "ERR" message through the serial port manually by pressing the ENTER key, but without success, as the intervals were too short or too varied between some jams.

In a second step, we tried to use the sound factor generated by the relay to send the jam message at the right time with a python script using the internal microphone and taking into account that the relay generates noise both at start-up and at the end. Unfortunately, there were too many additional factors preventing accurate detection, such as ambient noise, the faintness of the sound emitted by the relay and the sensitivity of the microphone.

Finally, we used a python script based on the opencv and numpy libraries to detect the status of the led (on and off) using a camera. This script initiated a serial connection with the arduino, then sent the message "ERR" each time the led was switched on, and returned the serial output following this action.

```python
import cv2
import numpy as np
import serial

# Initialize the camera
cap = cv2.VideoCapture(0)  # Use the default camera

# Variables to keep track of LED state
led_state = False

board = serial.Serial("/dev/cu.usbmodem1401", 115200)
print("Arduino connected!")
print(board.readline())

input("Start LED recording...")
count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    # Extract the red color channel
    red_channel = frame[:, :, 2]
    # Use thresholding to detect LED changes
    _, binary = cv2.threshold(red_channel, 50, 255, cv2.THRESH_BINARY)
    # Count the number of white pixels in the binary image
    white_pixel_count = np.sum(binary == 255)
    # Set a threshold for considering the LED as on or off
    threshold = 1000
    # Check if the LED state has changed
    if white_pixel_count > threshold and not led_state:
        led_state = True
        print("LED turned on")
        if count != 5:
            # Sending jam message and returning the serial output
            board.write(b"ERR")
            print(board.readline())
        else:
            while True:
                print(board.readline())
    elif white_pixel_count <= threshold and led_state:
        led_state = False
        print("LED turned off")

    # Display the camera feed
    cv2.imshow("LED Detection", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

# Release the camera and close the OpenCV window
cap.release()
cv2.destroyAllWindows()
```

Fig. 11. Python script detecting LED state change

After 5 iterations, the flag was successfully returned to us. This light factor is still unstable due to ambient light factors, but can be greatly improved and made more precise by bringing the LED as close as possible to the camera detecting the change of state.

```
LED turned on
b'Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 1/5\r\n'
LED turned off
LED turned on
b'Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 2/5\r\n'
LED turned off
LED turned on
b'Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 3/5\r\n'
LED turned off
LED turned on
b'Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 4/5\r\n'
LED turned off
LED turned on
b'Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 5/5\r\n'
LED turned off
LED turned on
b'/********************  YOU BEAT THE CHALLENGE!!!  ********************/\r\n'
LED turned off
LED turned on
b'Place the following flag in your report\r\n'
LED turned off
LED turned on
b'mMmCaNdY\r\n'
LED turned off
```

Fig. 12. Serial output after jamming engine 5 times

To mitigate this kind of side channel attack, implementing a robust error handling mechanism could prevent exploitation of errors. In the case of the "Vender Bender" challenge, sending "ERR" to jam the motor was possible due to a vulnerability in error handling. In order to prevent the timing aspect of this side channel attack, we could design a more secure management of payment in the machine and of product distribution during payment, for example by preventing the engine from engaging once the snack has been selected and the credit inserted.

Challenge Flag: "**mMmCaNdY**"