Side Channel Attacks On Cyber-Physical Systems Final Report: Challenges Resolution

Adam Henault
University of South Brittany
Lorient, France
henault.e2107041@etud.univ-ubs.fr

Florian Lecocq
University of South Brittany
Lorient, France
lecocq.e2202804@etud.univ-ubs.fr

Axel Gouriou
University of South Brittany
Lorient, France
gouriou.e2204185@etud.univ-ubs.fr

Philippe Tanguy
University of South Brittany
Faculty Advisor
Lorient, France
philippe.tanguy@univ-ubs.fr

***Abstract*—This article describes how we put in practice our hacking skills to uncover vulnerabilities and solve the challenges of the ESC 2023 competition. Our team BitsFromBZH received the provided cyber-physical system's challenge material and adopted a systematic approach to resolve each task. Our strategy emphasized the automation of assault techniques to efficiently retrieve all of the coveted flags.**

I. Introduction : Challenge hardware setup analysis

First of all, a comprehensive examination of the challenge's hardware configuration was conducted to ensure a thorough understanding of the components involved (refer to Figure 1). The master piece of the system is the Arduino Uno board which contains the ATmega328P AVR microcontroller running each challenge firmware. Indeed, a custom board shield that is plugged onto the Arduino was made for the ESC 2023 challenge, allowing other electronic modules to be connected. In addition to the UART console which is mainly used to communicate with the system, 4 actuators and 2 sensors made the challenges more interactive to solve by giving several ways to interact with the system, simulating a real world cyber physical system.

The actuators provide us an output flow of data by different ways:

- **Sound** : A buzzer and a relay are able to produce a "beep" at certain frequencies and a "clic clac" sounds that can contain encoded informations.
- **Vibrations** : a Vibrator module that generates mechanical waves
- **Visual indication** : A 7-segment display is able to show the 16 different digits from the hexadecimal notation.

Furthermore two sensors are used to input data to the system:

- **Microphone** : The Arduino is able to listen to its surroundings with a small microphone module by detecting a noise threshold.

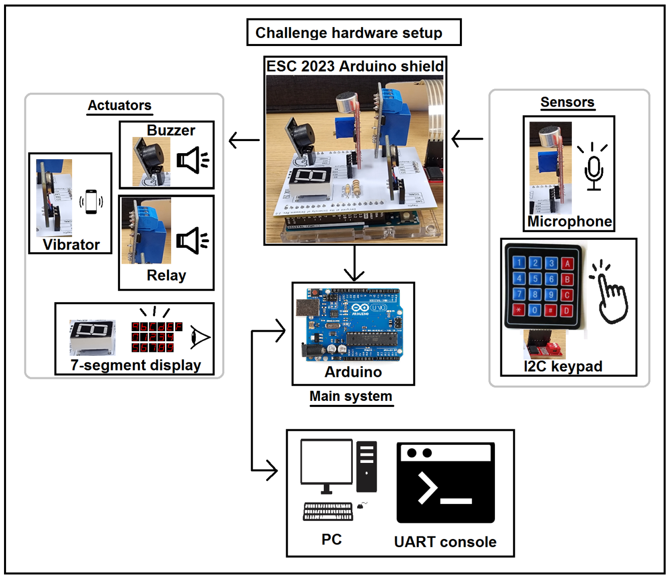


Fig. 1. Challenge hardware setup

- **Keypad** : A 4x4 button keypad is useful to enter some credentials like a real world CPS with a control access.

The challenge firmware is loaded into the Arduino so that we can use the same setup to solve all the challenges. Each of the challenges are not using all the actuators and sensors, in the following challenge resolution we will specify explicitly which one are used.

II. Challenge 1 : All white Party

A. Challenge description

This challenges consists of entering in a private party which requires an invitation that we don't have. To be able to find a way in we have to bypass the entrance gate security system. The problem is that an authentication process requires both a username and a 10-character digit PIN. Our goal is to devise a method to breach this security feature within a feasible *time frame*. Inputs are made via the UART console for the username and via the keypad for the password.

B. *Challenge resolution*

As we started to solve this challenge we saw that a hint was intentionally written in the challenge description because a particular word was in italic : amount of ”*time*”. We understood that the organisers indicated that the way to solve the challenge is to use a timing attack.

a) Finding the correct username : timing attack : An attempt to find the valid username through brute force would have been impractical within the time constraints of the event. Fortunately, the system's username validation protocol was vulnerable to a timing attack.

The timing attack consists in trying to find each characters of the username one by one by measuring the execution time of the program running on the Arduino's micro-controller. The vulnerability is that when the program verifies each characters one after the other, if it is the correct one or an incorrect one the execution time is different. This leakage allows us to find the correct username really quickly by simply guessing each character one by one.

As an attacker, to measure the execution time of the program over UART we simply measure the time between the moment when we send the guessed username and the moment we receive the answer message from the system.

For instance, the temporal profile associated with an attempt to ascertain the initial character is plotted in the figure 2. We can clearly see the letter ”B” induces a significant increase in the program execution time. This is how we found that it was the first correct character.

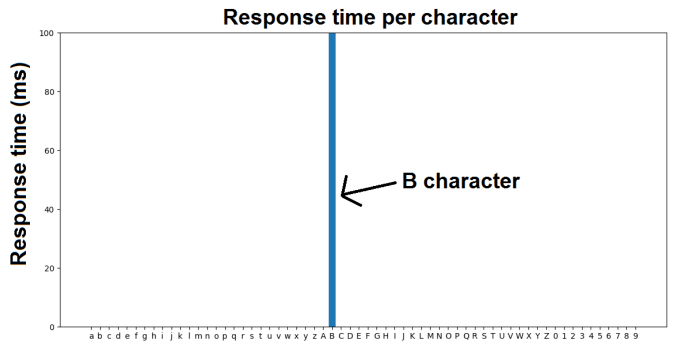


Fig. 2. Response time per character while trying to guess the first character of the correct username

Subsequent to this discovery, we replicated the procedure for the remaining characters, employing an automated approach through a Python script to perform the task sequentially. This algorithmic strategy enabled us to ascertain that ”Barry” was the legitimate username.

b) Finding the correct password : hash collision : After entering the correct username, the system asks to enter the 10-digit MFA code (that is sent to Barry's phone) on the keypad. Unfortunately, this 10-digit PIN password which needs to be typed in is not vulnerable to the same timing attack method. Another method is needed to find the password.

A pivotal aspect of this part involved analyzing the system's UART feedback upon incorrect PIN entry. It is said that the password SHA doesn't match and gives us five decimal values (167 98 212 144 128) that are always the same with the username ”Barry”. These printed log outputs leaked some critical information. The difficulty was to identify what they were corresponding to and how the system was verifying the password hash.

We understood that these five decimal values which were displayed were the first 40 bits on the Most Significant Bits (MSB) side of the SHA1 hash of the username ”Barry” (Figure 3). The security mechanism simply compares the selected 40 bits of the SHA1 hash of ”Barry” with the 40 bits of the SHA1 hash of the 10-digit PIN that we enter.

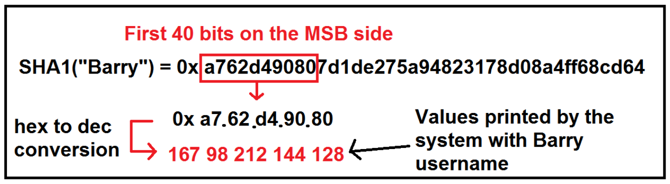

First 40 bits on the MSB side
SHA1("Barry") = 0xa762d490807d1de275a94823178d08a4ff68cd64
hex to dec conversion
0x a7.62.d4.90.80
167 98 212 144 128
Values printed by the system with Barry username

Fig. 3. Finding what the values printed by the system are corresponding to

There are two security problems here :

- The hashing method used is SHA1 which is now considered obsolete and susceptible to hash collision attacks.
- The system is not comparing the complete 160 bit SHA1 hash of the username with the one of the password but only the 40 bits on the MSB side. This makes the hash collision attack faster to execute.

Furthermore, we unintentionally found a third vulnerability: the system tolerates additional characters appended to the correct username, considering them as valid. For instance, ”Barry1234” is accepted, despite altering the resulting SHA1 hash.

In light of these vulnerabilities we decided to put in practice the hash collision attack that we coded in a Python script. The goal was to identify a combination of a username seen as correct paired with a 10-digit PIN sharing the same 40-bit SHA1 hash. The details of this collision discovery process relying on bruteforce is described in Figure 4.

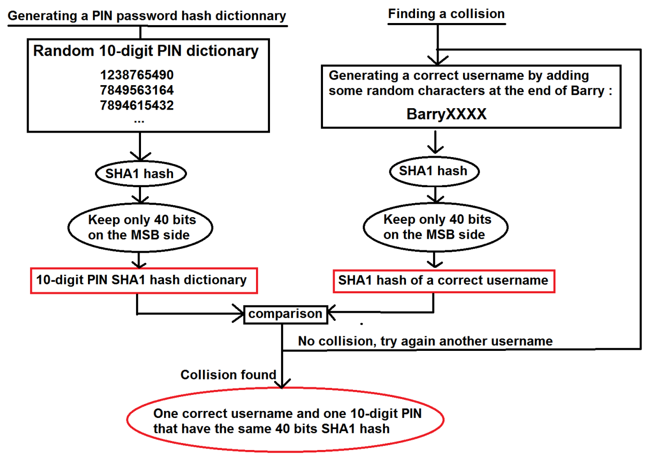

Generating a PIN password hash dictionnary
Random 10-digit PIN dictionary
1238765490
7849563164
7894615432
...
SHA1 hash
Keep only 40 bits on the MSB side
10-digit PIN SHA1 hash dictionary
Finding a collision
Generating a correct username by adding some random characters at the end of Barry :
BarryXXXX
SHA1 hash
Keep only 40 bits on the MSB side
SHA1 hash of a correct username
comparison
No collision, try again another username
Collision found
One correct username and one 10-digit PIN that have the same 40 bits SHA1 hash

Fig. 4. Hash collision attack algorithm

By running our program we found a hash collision with the following credentials to beat the challenge: `username=Barryahhssuuw PIN=1112233669`. This combination of username and PIN password shares an identical 40-bit hash, by entering it in the system the challenge is resolved.

In summation, we successfully entered an entrance gate that was protected with 3 authentication methods : A badge that we found on the floor, the username that we determined through timing analysis and a 10-digit PIN password that we acquired via hash collision. This experience underscores that even systems employing multiple authentication mechanisms can be rendered vulnerable if their implementation harbors exploitable vulnerabilities.

III. Challenge 2 : Bluebox

A. Challenge description

The second challenge's scenario introduces a secret underground lair where we uncover the hardware for the legendary blue box hack. This hack refers to an electronic device called "Blue box" that was used between the 1960s and the 1980s to generate in-band signaling tones allowing an illicit user to place long-distance calls without paying. Our goal is to recreate this hack quickly by decoding the telephone frequencies and recreate the iconic audio tones before the authorities arrive to shutdown our secret operation.

Upon initialization of the Arduino with this challenge firmware, the buzzer emits four consecutive random tones. Each one of the ten possible different tones has a unique frequency and corresponds to a keypad digit from 0 to D (and *). To win the challenge we have to replay the four tones that just had been emitted by pressing the corresponding digits on the keypad and validate the entry by entering #. An example of the recording of four random tones played can be seen on figure 5.

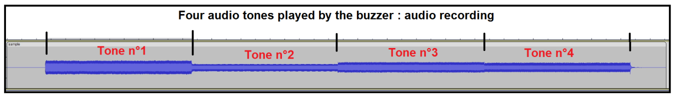


Fig. 5. Example of four audio tones played by the buzzer

B. Challenge resolution

To accurately discern each tone, we recorded audio samples for all tones using a smartphone. Subsequently, we conducted a frequency extraction process by applying a fast Fourier Transform (in Python) to the audio signal, identifying the fundamental frequency in the plotted graph (Figure 6).

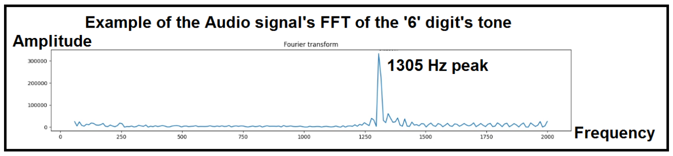


Fig. 6. Example of a signal's FFT

All the obtained frequencies are shown in figure 7.

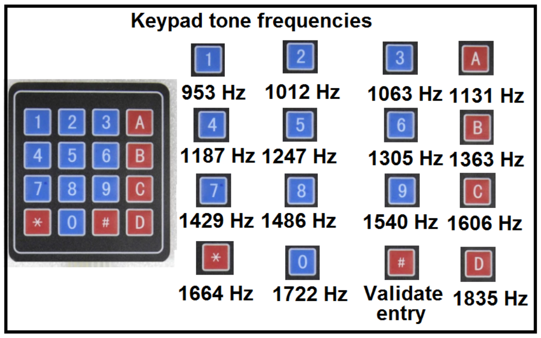


Fig. 7. Keypad tone frequencies analysis

At this stage we could just record the audio of the four random tones generated and manually extract the four frequencies as we did before to know what digits need to be entered to win but we wanted to automatize the process. So we wrote a Python script that give us automatically the heard digits from an audio recording.

After obtaining automatically the four random correct digits with our program, we pressed them on the keypad and validated each entry with #. This successful completion of the challenge was accompanied by the presentation of the flag, given with tones played by the buzzer. Employing the same automatic frequency extraction program on the flag's recording enabled us to decode it: `flag=B339B009`.

IV. CHALLENGE 3 : OPERATION SPITFIRE

A. Challenge description

In the third challenge we play the character of an accomplished spy amid a digital battleground. We are assigned the mission to decipher an intricate maze of wire traffic acquired from the mysterious hacker collective SPIitFire. Our remote team has gained access to a surveillance camera that allows us to clandestinely exchange messages.

Our goal to uncover the flag of this challenge is to find out how to communicate with the security camera that is represented by the Arduino.

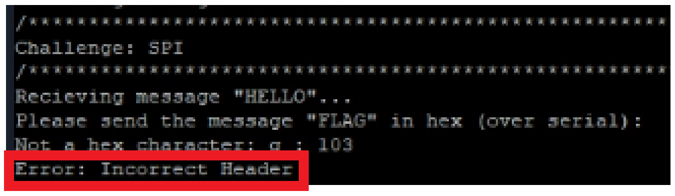
B. Challenge resolution

When starting the challenge, it is written on the Arduino serial communication that the system is receiving the message ”HELLO” and the relay module emits a distinct ”Clic Clac” audio signals for a duration of 13 seconds. We understood the relay represents the way of clandestinely exchange messages with our remote team and that the sounds emitted are representing the ”HELLO” encoded message data but we didn’t know how it was encoded at first.

After that the system asks to enter the message ”FLAG” in hexadecimal notation over serial and here is the difficulty of this challenge.

Thanks to the error log messages over serial, when trying to input data we figured out the flag message that needed to be entered had to respect a specific communication protocol. The crux of the challenge involves the reverse engineering of this protocol to be able to forge a new correct packet which contains the ”FLAG” message. In the following sections, we provide a comprehensive account of our step-by-step approach to perform a reverse engineering analyse of this communication protocol.

The first error log message that was printed when entering some data was *ERROR: Incorrect header* (Figure 8). Furthermore, by entering some non hex character that triggered error messages, we saw that the system was only analyzing the first two characters.



```
/*****************************************************
Challenge: SPI
/*****************************************************
Recieving message "HELLO"...
Please send the message "FLAG" in hex (over serial):
Not a hex character: g : 103
Error: Incorrect Header
```

Fig. 8. Error log from incorrect header packet

At this point we understood that the first two hex characters of this protocol’s packet are the header part. Consequently, we devised a Python script to systematically brute force the header by exhaustively testing all possible two-character combinations. This only took 20 min to find that the correct header was *0xA5* and that it was always the same header used.

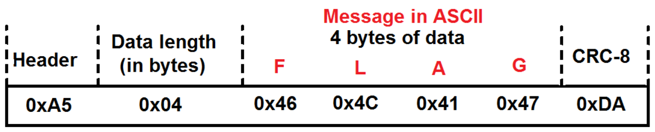
After having the right header, two other errors were printed out when trying to craft a packet *ERROR: Incorrect length* and *ERROR: Data length should not exceed 10*.

Further analysis performed on random data after the header led us to deduce that the following two hexadecimal characters signified the data length in bytes (second part of the packet) followed by the data themselves (third part of the packet). So we crafted a packet containing the header *0xA5*, followed by the data length of 4 *0x04* and by the ASCII representation of the four ”FLAG” letters : *0x46 0x4C 0x41 0x47*.

Then the following error occured : *ERROR: Bad CRC*. We understood that a Cyclic Redundancy Check (CRC) was the last part of the packet. This is a common method in communication protocol to verify if errors occurred during the transmission of a message.

Since we deal with a small packet, we assumed that a CRC-8 algorithm was applied on the whole message (header + data length + data) resulting in a one byte value written at the end of the packet.

Once the reverse engineering of the communication protocol was successful, we crafted a packet containing the ”FLAG” message and sent it over serial as shown in figure 9.



Header	Data length (in bytes)	Message in ASCII 4 bytes of data F	L	A	G	CRC-8
0xA5	0x04	0x46	0x4C	0x41	0x47	0xDA

Fig. 9. Crafted packet containing the FLAG message

The buzzer indicates that the challenge is over by playing a winning melody meaning that the crafted packet that we sent was correct. The flag was not displayed in the serial console but was relayed through an encoded message played by the relay, lasting for 15 seconds, akin to the ”HELLO” message at the challenge’s beginning of the challenge.

To decode the two audio messages sent by the relay (HELLO and the flag of the challenge), we recorded an audio sample for each one, capturing the ”Clic Clac” sounds and wrote a Python script that automatically extract the binary data from these audio samples using the same packet method as before, giving us the message.

As seen in Figure 10, each peak in the raw audio recording of the flag message corresponds to a ”Clic” or ”Clac” sound, indicating relay activation or deactivation (transition). While the relay is active, it represents binary data 1; when deactivated, it signifies binary data 0. Each bit of data persists for a fixed duration, indicative of asynchronous communication.

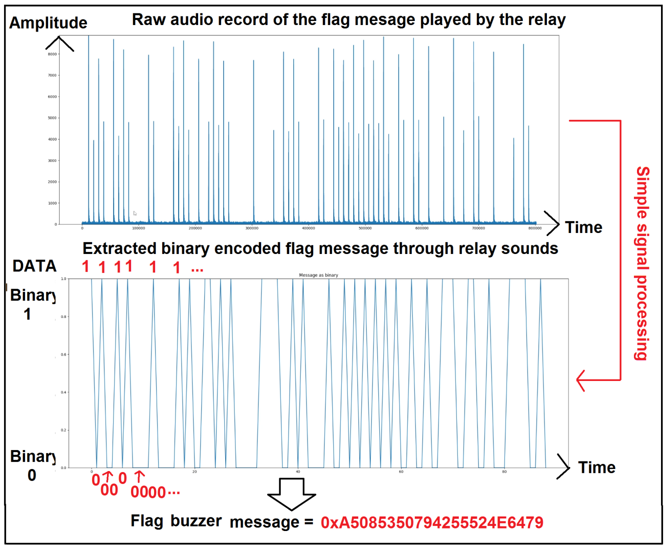


Fig. 10. Python program signal processing and decoding the audio record of the flag packet through the relay

Finally we obtained the decoded message automatically from the relay with the Python program :

- HELLO buzzer sound : length = *5* DATA = *HELLO*
- FLAG buzzer sound : length=*8* DATA=*SPyBURNd* CRC=*0x79* raw=*0xa5085350794255524e6479*

The challenge is won, the flag is `SPyBURNd`. The main hacking skill which was put in practice here was reverse engineering. It is worth noting that the two CRCs in the encoded ”HELLO” and flag packets were found to be incorrect, despite the data contained within the packets appearing correct. The reason for this discrepancy remains unclear, though it is presumed to be an error.

V. Challenge 4 : czNxdTNYzM

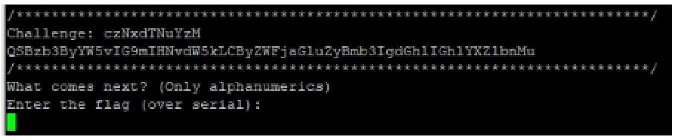
A. Challenge description

The fourth challenge scenario is very enigmatic. It deals with a cryptic dance of numbers and it is asked if we can harmonize with the rapid rhythms of this challenge. In this realm of numbers, a symphony can conquer even the swiftest of mysteries.

When starting the Arduino, a strange output data comes from the UART and the 7-segment display light up quickly. We need to understand what is happening and what are all of those mysterious data.

B. Challenge resolution

The data sent over serial by the Arduino is shown in figure 11. It includes the title of the challenge and a strange string.



```
/**********************************************************************/
Challenge: czNxdTNuYzM
QSBzb3ByYW5vIG9mIHNvdW5kLCByZWFjaGluZyBmb3IgdGhlIGhlYXZlbnMu
/**********************************************************************/
What comes next? (Only alphanumerics)
Enter the flag (over serial):
```

Fig. 11. Serial output from the Arduino for the challenge 5

Then it is asked for what comes next and to enter the flag.

The first thing we noticed is that the challenge’s title is not common and could hide a hint to solve the challenge. So after investigating we found that it was the Base64 representation of the word ”s3qu3nc3” meaning sequence. Also, ”A soprano of sound, reaching the heavens.” is the Base64 representation of the strange string which is given.

Another conspicuous element was the rapid activation of the 7-segment display. As it was asked in the scenario of this challenge ”Can you harmonize with the rapid rhythms of this challenge” we understood that some data were printed at a rapid rhythm from the 7-segment display. So we recorded a video of it and watched it in slow motion to extract the data that were printed.

The sequence of number that was displayed is a very specific integer sequence called OEIS A008336 invented by Bernardo Recaman. We found it by searching the numbers on Google. This is a recursive sequence and each next element can be calculated from the previous one using a formula.

The 24 first numbers of this sequence are printed on the 7-segment display and it is asked ”What comes next ?” in the serial console, so we wrote a Python script which sent the 25th integer number of this sequence to the Arduino.

The challenge is defeated and the flag itself is the 25th element of the integer sequence that we talked about: `flag=97349616`.

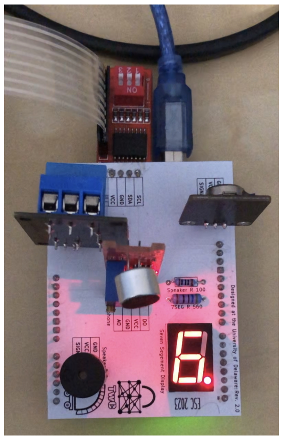


Fig. 12. Seven Segment displaying numbers

VI. Challenge 5 : Sock and Roll

A. Challenge description

The 5th challenge takes place in a whimsical and colorful Happy Socks factory where we are caught captivating in a slightly bizarre world. However, what was initially a delightful visit has turned into an unexpected challenge. We have been mysteriously locked inside one of the factory's rooms. The laughter and cheerfulness of the factory have given way to a sense of urgency as we realize we need to escape.

The only way to exit this colorful world and return to reality is to use the "Happy Tap Dancing Socks Message Machine 2000" that we found which seems to be transmitting some strange message. The goal to obtain the flag is to send a distress signal using this cutting-edge messaging technology.

B. Challenge resolution

When starting this challenge, the buzzer emits some sound patterns with a specific tone and the microphone listen to it. The serial communication indicates if the message sent is received or not by the microphone by printing the default message which is captured : "Everything is OK" as shown in figure 13.

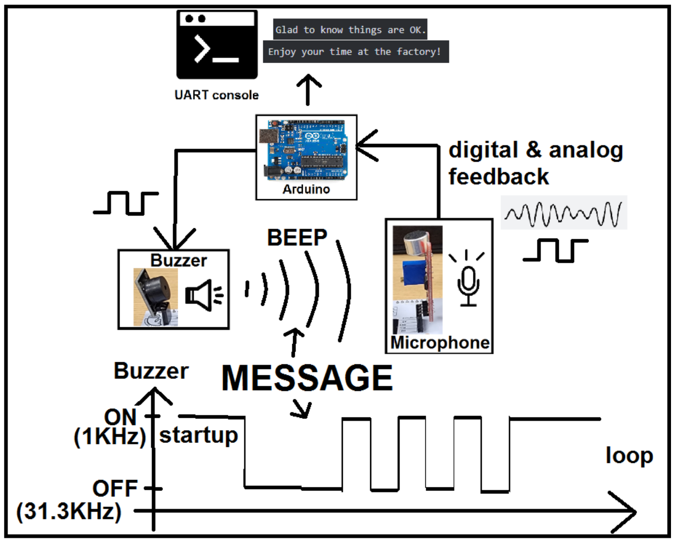

```
Glad to know things are OK.
Enjoy your time at the factory!
```

Fig. 13. Challenge 5 emitting a strange message

At first, comprehending the operation of this cutting-edge messaging technology remained elusive. Despite numerous days of experimentation, involving attempts such as Morse code, diverse sound patterns, and various interactions with the buzzer and microphone, the solution remained unreachable.

So we took a step back and analyzed the system while running to have a better view of how it worked :

- The buzzer : The Arduino sends a square wave to it with an adjustable frequency. To turn the buzzer ON, a square wave of 1.1 KHz frequency is used whereas a 31.3 KHz frequency square wave is used to turn it OFF (As humans we can't hear that frequency).
- The microphone digital sensor output : This digital output is equals to a logic '1' if the sound threshold is crossed, and '0' if the actual ambient sound level is below the threshold. To sum it up, if the buzzer is turned ON, this output is '1' and if the buzzer is OFF it is '0' : it is used to detect the buzzer activation pattern.
- The microphone analog output : This output is the raw audio signal coming from the microphone with an average value of 1.7v. It has a small AC amplitude that is null when the buzzer is OFF and which is equal to a 0.7v square wave of 1.1 KHz when the buzzer is ON. This microphone's pin is wired to an analog input of the Arduino.

We inferred that the Arduino was not only detecting the buzzer activation pattern but also the frequency received. The Arduino is running at a high frequency of 20 MHz and its Analog to Digital Converter is easily capable of sampling the AC analog signal received which has a much lower frequency (KHz range).

After having these precious information, we did researches on distress signal frequencies and found something interesting. In the radiotelegraphy domain several international distress frequencies exist and are used for emergency cases in a survival craft context. Some can reach up to 3000 Km. The one that caught our attention was 8364 KHz, but we kept 8364 Hz so it is in the Arduino's measurement range.

So we disassembled the challenge hardware setup by removing everything and by keeping only the Arduino main board and the I2C keypad. Then we took a second Arduino which served as a hacking tool and wired it up to the CSAW Arduino as shown in figure 14. We kept the I2C keypad to bypass the mandatory challenge hardware setup check which happens at every startup but the keypad is not used.

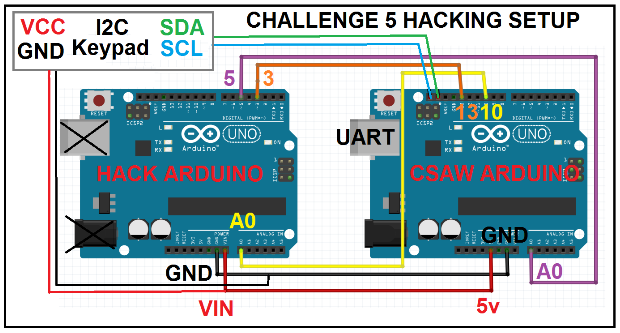


Fig. 14. Challenge 5 automatic hacking setup with a second Arduino

To find out what is the pinout of the CSAW Arduino we proceeded to the hardware reverse engineering of the CSAW ESC 2023 Arduino shield.

The second Arduino has three simple tasks :

- Sample the signal that comes from the buzzer command pin of the CSAW Arduino and extract the frequency (1.1 KHz or 31.3 KHz) at high speed permanently.
- Depending on the extracted frequency control a digital output wired to the CSAW Arduino microphone digital input pin (replacing the microphone digital output) as followed : 1 KHz = logical '1' ; 31.3 KHz = logical '0'. This aims at emulating the microphone digital signal that should have been received. (the original signal from the buzzer activation pattern is not modified, it just passes through a different transmission canal).
- Synchronize with the activation pattern of the buzzer pin and emit an adjustable frequency digital square wave signal with specific frequencies as shown in figure 15. This signal replaces the microphone analog output and this time we intentionally modify the frequencies by introducing the SOS international 8364 Hz during the message transmission. (We didn't emit an analog signal as it should have been with the microphone but a digital signal also works fine)

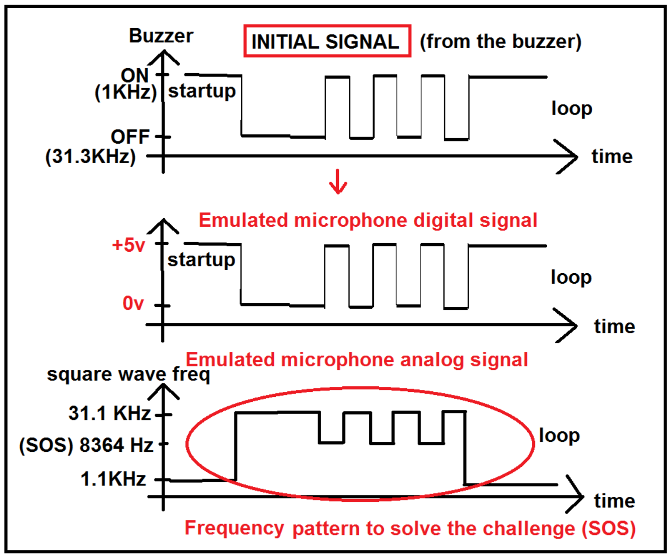


Fig. 15. Sending the activation pattern with the SOS frequency

Once both Arduino start, the CSAW Arduino begin by sending the buzzer signal sequence and the second one synchronize with it and send the SOS distress signal by using the SOS specific frequency and by emulating the microphone. The SOS is received by the CSAW Arduino and the flag is printed successfully and automatically on the first try: `flag=f0otN0t3`.

VII. Challenge 6 : Vender Bender

A. Challenge description

For the sixth challenge, we find ourselves in front of a vending machine. This one contains our favorite sweets and we're able to pay, but it seems that by forcing an error on the machine we can get our money back at the same time as our candy.

In this scenario, we imagined that the aim of the challenge is to provoke this error in order to get our money back and our sweet for free to obtain the flag.

B. Challenge resolution

When logging on to the UART console for the first time, we were prompted to send the "ERR" command to jam the vending machine. Considering the context of the challenge, we quickly understood that it was by sending the command that we were supposed to force the error on the machine.

Our initial attempt involved manually transmitting the "ERR" command through the UART console. However, this approach failed to elicit the desired response from the machine. Multiple iterations yielded no change in the machine's behavior. It became apparent that timing was a crucial factor in the process. So we tried to find the precise moment for sending the "ERR" command by automating the task with a Python script, as depicted in Figure 16.

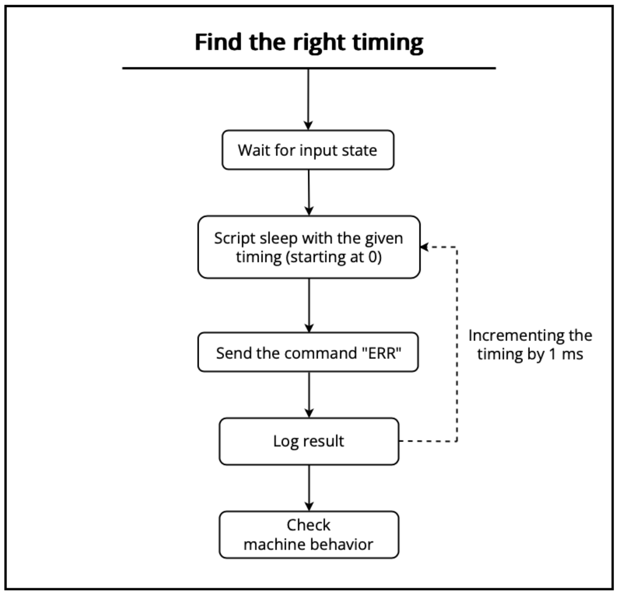


Fig. 16. "Find the right timing" automation

This method enabled us to find the right timing for triggering the error on the machine. However, even with the right timing, we didn't always get the same result, i.e. triggering the error. It's therefore possible that some randomness has been added to the operation of the snack vending machine.

After having the timing right, the error looked like this: ”Motor Error 5902 Reported. Slight but not significant motor movement detected. Retry Attempt 1/5”. As you may have noticed, it’s necessary to trigger the error 5 times in succession. Initial attempts to reuse the timing for the first error for subsequent errors proved unsuccessful and we had to reuse the previous method to find out how to get a series of 2/5. So we went back to the same script as before, adding the first error trigger.

In summary, we conducted five brute-force timing operations to ascertain offsets. These timing offsets facilitated the generation of a 5/5 error sequence, successfully concluding the challenge (figure 17).

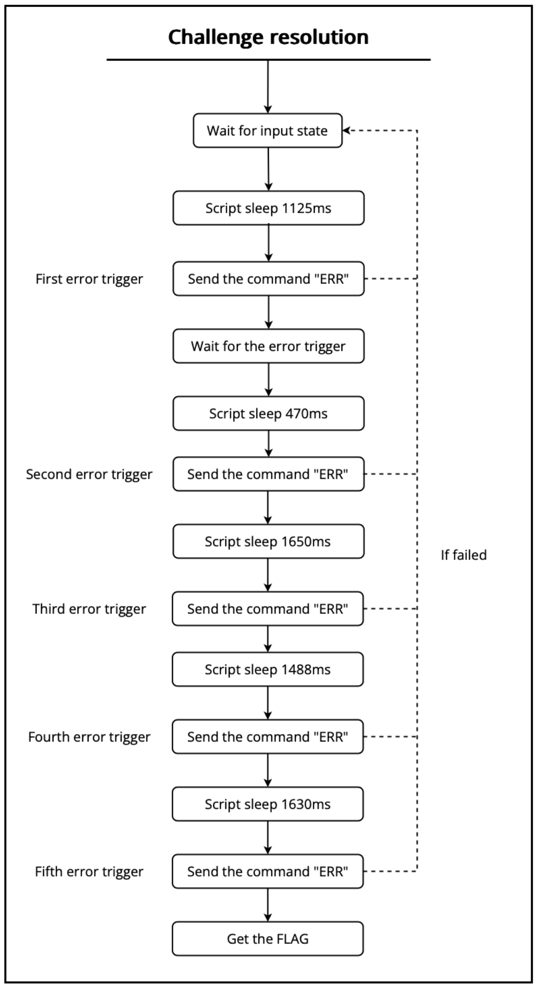
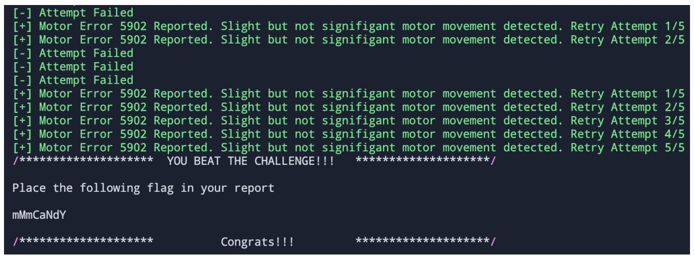


Fig. 17. Automating the challenge resolution.

Once the script with all the offsets is created, the challenge resolution is automated (figure 18). The script takes an average of 20 minutes to trigger the error sequence and retrieve the flag, `flag=mMmCaNdY`.



```
[-] Attempt Failed
[+] Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 1/5
[+] Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 2/5
[-] Attempt Failed
[-] Attempt Failed
[-] Attempt Failed
[+] Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 1/5
[+] Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 2/5
[+] Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 3/5
[+] Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 4/5
[+] Motor Error 5902 Reported. Slight but not signifigant motor movement detected. Retry Attempt 5/5
/********************  YOU BEAT THE CHALLENGE!!!   ********************/

Place the following flag in your report

mMmCaNdY

/********************          Congrats!!!         ********************/
```

Fig. 18. Automated challenge solving.

VIII. Conclusion

We had a lot of fun to solve all of the challenges of the CSAW embedded security 2023 edition. The hardware setup that represented a real world cyber physical system made this capture the flag competition even more exciting. The two last challenges required more time to be solved but the satisfaction when we succeeded was even better.

The hacking skills developed are really diversified and we learned a lot : Side channels, timing attack, hash collision, bluebox hack, protocol reverse engineering, audio analysis, Python scripting, automating an attack, finding vulnerabilities and exploiting them, PCB reverse engineering, different ways to encode a message, making a custom-made embedded system hacking tool.

Finally, we would like to thank all the people who prepared all these great challenges that were accompanied with well worked scenarios. In the end the most important thing we learned is that the best quality which an embedded system hacker can have is the persistence: *”When you are about to give up, success could be closer than you might think.” (BitsFromBZH)*