CSAW 2023 Embedded Security Challenge

Isabelle Friedfeld-Gebaide, Daniel Ivanovich, Andreas D. Kellas, and Emily Soto
Columbia University

I. All White Party

A. Overview

The All White Party challenge requires a timing side-channel attack to leak a username from the firmware; each correct character in the correct position in the username, when scanned from beginning to end, resulted in an additional 300ms delay in device vibration. We measured this delay while iterating through the character sequences of letters and numbers, and were able to recover the username `Barry`. The device then prompts the user to enter a 10-digit PIN number, which we were unable to recover.

The challenge first prompts the user to enter a username over serial connection, and if a valid username is presented, then prompts the user to enter a 10-digit pin number using the number pad. Correct entry of the username and password presumably results in the flag.

We did not know the username or password, and so we were required to conduct a side-channel attack to leak information from the Arduino. We observed that after each username entry, the Arduino vibrated the haptic sensor device after some baseline length of time. For each correct letter in the correct position in the username, the device delayed for an additional 300ms relative to the baseline before vibrating. For example, entering `B` resulted in an additional delay of 300ms before vibrating, while any other single letter (capital or lowercase) or number resulted in only the baseline delay; `Ba` resulted in 600ms of additional delay, while `BA` (or any other sequence of `B` followed by any other letter or number) resulted in an additional delay of 300ms. This observation allowed us to continue accumulating assumed correct letters until we leaked the username `Barry`, which was accepted by the Arduino. `Barry` is presumably a reference to Barry Manilow, the singer infamous for wearing all white suits and tuxedos.

When presented with the correct username, the Arduino displays a message indicating that a 10-digit PIN has been sent to a mobile phone, and that the user should enter it using the number pad. When presented with an arbitrary 10-digit input, the Arduino displays an error message ("Password SHA does not match:") and prints five numbers. We assumed that the five numbers (all below 256) are intended to be the first five bytes of a SHA hash of the correct 10-digit PIN. We implemented a brute-force search algorithm that scanned the search-space of 10-digit numbers and computed SHA hashes using multiple SHA algorithms (since we did not know specifically which algorithm was used to generate the target hash), but were unable to identify a sequence that started with these bytes.

B. Flag

Unsolved

C. Challenge Text

"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 glamour in a reasonable amount of *time*."

D. Approach

Based on the emphasis on the word "time" in the presented challenge text, we assumed that the challenge would require a timing side-channel attack to solve. Because of the noticeable delay between the transmission of a username and the vibration of the haptic sensor, we suspected that this delay could be measured and varied based on the username input. We did not immediately know whether the time variations would be in how long the sensor vibrated for, or in how long it would take for the vibration to occur after transmitting the username, so we set out to measure both.

For accurate measurements of the delay timing, we used an oscilloscope to measure the signal sent to the haptic sensor from the Arduino. We attached a probe to the Signal wire of the haptic sensor, and recognized that a 5V high signal from the Arduino resulted in the haptic sensor vibrating. We first tried sending messages of varying lengths and with arbitrary contents to see if any differences appeared; for all initial attempts, the signal was high for 600ms. Because we were not able to determine precisely when the Arduino received our message from the serial connection, we needed a technique for measuring the delay between the Arduino receiving the serial message and sending the high signal to the haptic sensor. To measure this, we sent a series of messages, where each message was sent periodically so that we could see the relative differences between the signal high. For example, by sending one message every three seconds, we could identify any relative timing delays between the rising edges of the high signal; if the delay is constant for all messages, then the rising edge should appear on a three second period, but if any message results in more or less delay, then the rising edge will be shifted.

In our initial attempts of sending messages of arbitrary contents and arbitrary lengths, we saw no immediate differences between the time that the signal was high (length of vibration), or the time that the signal was low (delay before vibration).

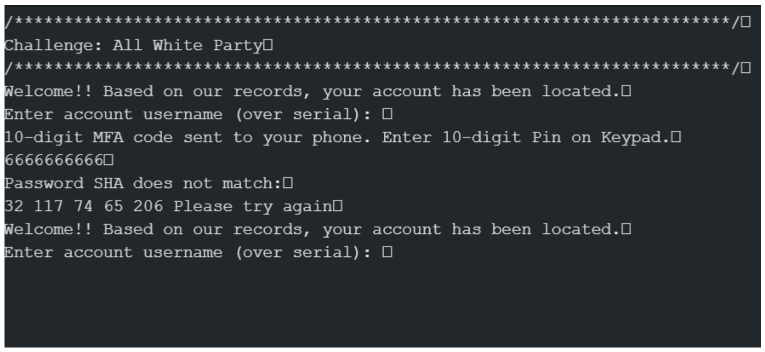
We next systematically created messages by sending a single character at a time, iterating through the lower and upper case characters and digits. We noticed that for the upper case character `B`, the rising edge of the signal appeared 300ms later than it should have for the period. We hypothesized that `B` was the first character of the username.

We sent another sequence of potential usernames to the Arduino, but this time fixed the first character as `B` and for the second character, we iterated through the lower and upper case characters and digits. In order to observe the full time delay, we interspersed each message with a message that we suspected was incorrect, we arbitrarily chose the letter `g`. So, for example, every three seconds, we send one message from the sequence of `[g, Ba, g, Bb, g, Bc, ...]`. For each guess attempt, we observed a 300ms delay in the period, as expected from the `B` character, and on the attempt for `Ba` we observed another 300ms delay, for a total of 600ms. We therefore hypothesized that the username began with `Ba`.

We continued in this manner of identifying the next character in the suspected username by identifying the letter that added an additional timing delay relative to the other letters. We eventually recovered the string `Barry`, which received a positive message from the Arduino over the serial connection, rather than the negative response we received for all other tested usernames.

The Arduino then presented us with the message `10-digit MFA code sent to your phone. Enter 10-digit Pin on Keypad`. We entered an arbitrary 10-digit PIN using the keypad, and received the message `Password SHA does not match: 32 117 74 65 206`. We guessed that the five numbers were the decimal representation of five bytes, since they were all values less than 256 and presented in the context of presumably a SHA hash. We assumed that the task was to identify the 10-digit PIN that, when hashed with a SHA hash function, results in a hash that begins with the five bytes presented. We wrote a script that iterated through all possible 10-digit PINs and computed their hashes using the SHA-1,SHA-224, SHA-256, SHA-384, SHA-512, SHA3-224, SHA3-256, SHA3-384, and SHA3-512 algorithms, but were unable to identify a PIN that resulted in such a hash.

In retrospect, we suspect that we also could have noticed a timing delay between the time that our script sent a message to the serial console attached to the Arduino and the time that we received a response back from the serial console. This approach would have been easier to automate, since it would not require visual inspection of the oscilloscope, but would have had less precise timing measurements from software events.



```
/***********************************************************************/
Challenge: All White Party
/***********************************************************************/
Welcome!! Based on our records, your account has been located.
Enter account username (over serial): 
10-digit MFA code sent to your phone. Enter 10-digit Pin on Keypad.
6666666666
Password SHA does not match:
32 117 74 65 206 Please try again
Welcome!! Based on our records, your account has been located.
Enter account username (over serial): 
```

II. Bluebox

A. Overview

The Bluebox challenge required the accurate identification of a sequence of audible tones played by the Arduino for an audio replay attack. The user is expected to play back the tones by using the keypad to enter values that correspond to tones. The Arduino first played a sequence of four tones, and expected the user to enter four keypresses that reproduced the tones, in order. Pressing a key on the keypad resulted in a distinct tone played by the speaker on the Arduino. The Arduino then played a sequence of eight tones, and again expected the user to reproduce the sequence with eight corresponding keypresses. After the user reproduces the tones, the Arduino sends the flag over the serial console to the user.

B. Flag

B339B009

C. Challenge Text

"In a secret underground lair, you and your team uncover the hardware for 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. Its time to unearth the blue box technological history before the authorities arrive to shutdown your operation."

D. Approach

We first recognized that each keypress resulted in an audible tone produced by the Arduino speaker, where each key had the same tone. We suspected that the four tones played by the Arduino when the device powered on were able to be reproduced by pressing the keys on the keypad.

We used a musical instrument tuning app on a mobile phone ("T1" app on a Pixel 6 Pro) to determine the pitch and frequency of each of the tones produced by the keypad presses. The used the phone microphone to display the pitch as a musical note and the frequency in hertz of the tone. We pressed each keys on the keypad and recorded the frequency of the tone produced.

Once we created this table, we restarted the Arduino to trigger the challenge tone sequence. While the Arduino played the tones, we used the tuning app to display the tone frequencies on the phone screen, and used another phone to

TABLE I
KEYPAD MAPPING TO FREQUENCY OF TONE GENERATED

Keypad	Hertz
1	953
2	1010
3	1067
A	1129
4	1180
5	1248
6	1307
B	1364
7	1426
8	1480
9	1543
C	1600
$*$	1664
0	1722
D	1838

film and playback the recording of the tuning app screen display. When the sequence completed, we played back the recording, checking the frequencies that were played by the Arduino and pressing the key that mapped to the correct frequency in our table. The Arduino responded by playing a longer eight tone sequence, but we repeated the process described above of recording the sequence while using the tuning app and mapping the tone frequencies to the appropriate keypresses. Once we entered the appropriate keypresses, the Arduino responded by sending the flag over serial connection.

Retrospectively, we could have automated the process of determining the challenge tone frequencies by using an audio processing library like PyAudio to programmatically extract the tone frequencies from a recording of the Arduino challenge sequence, and map them to the correct keypresses that a user should input.

III. czNxdTNuYzM

A. Overview

The czNxdTNuYzM challenge (whose name is the base 64 encoding of `s3qu3nc3`) required the use of an acoustic input to slow down the visual display of information for retrieval. On device power-up, the Arduino quickly displayed a sequence of values on the seven segment display, but too quickly to be read. When a high-frequency tone was played into the Arduino microphone, the sequence of values on the seven segment display slowed down such that they could be read visually as sequences of decimal numbers, dots, and underscores. When transcribed, the digits formed a sequence of numbers computed by: $a(n+1) = a(n)/n$ if $n|a(n)$ else $a(n) * n$, starting at $a(1) = 1$. The Arduino then prompted the user over serial connection to enter the next number in the sequence `97349616`, which was accepted as the flag.

B. Flag

97349616

C. Challenge Text

"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."

D. Approach

On device power-up, the Arduino sent base 64 encoded messages over serial connection; one was the challenge name, and the other was the base 64 encoding of the message "A soprano of sound, reaching for the heavens." The seven segment display also began flashing values, but too quickly to be observed.

Based on the challenge text and the base 64-encoded message that referenced harmonies and sopranos, we guessed that the microphone was on and would receive inputs. Using the same tuning app ("T1") that we used in the Bluebox challenge, we played different tones while the Arduino was flashing the sequence on the seven segment display. We observed that display sequence slowed down considerably when a high tone was played, such that the values could be read by manual visual inspection. We specifically played a high-frequency note of 1974 Hz, corresponding to note B6 on the tuning app. We then made a visual recording of the sequence on the seven segment display and played it back in slow motion, to transcribe the sequence below:

```
1.
2.
6.
2_4.
1_2_0.
2_0.
1_4_0.
1_1_2_0.
1_0_0_8_0.
1_0_0_8.
1_1_0_8_8.
9_2_4.
1_2_0_1_2.
8_5_8.
1_2_8_7_0.
2_0_5_9_2_0.
3_5_0_0_6_4_0.
1_9_4_4_8_0.
3_6_9_5_1_2_0.
1_8_4_7_5_6.
3_8_7_9_8_7_6.
1_7_6_3_5_8.
4_0_5_6_2_3_4.
```

The underscores and periods were flashed on the seven segment display alongside the numbers. We spent a considerable amount of time trying to determine the patterns between the numbers on each row above. Then, thinking back to the decoded name of the challenge (`s3qu3nc3`), we tried treating numbers separated by underscores as digits of larger numbers and periods as the end of each number. By searching the online encyclopedia of integer sequences, we were able to find the rule for the sequence:$a(n+1) = a(n)/n$ if $n|a(n)$ else $a(n) * n$, starting at $a(1) = 1$. This let us quickly determine the next number, 97349616. We sent this over serial, and the output informed us that this was not only correct but also the flag.

IV. Operation SPItFire

A. Overview

The Operation SPItFire challenge required the analysis of a covert channel over an SPI device to craft and send messages over the covert channel. The Arduino uses the relay device to encode an SPI protocol message for the string `HELLO`. The user is prompted to send an encoding of the message `FLAG` using the same SPI protocol, which required including a valid message header, encoded message, and CRC. When we sent the correct encoding of `FLAG` to the Arduino, the Arduino transmits another message containing the actual flag, encoded using the SPI protocol. We decoded this message to recover the flag.

B. Flag

SPyBURNdy

C. Challenge Text

"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 camera, allowing you to clandestinely exchange messages. Your objective: find out how to communicate with the security camera, and acquire to uncover the coveted password flag that can be used to infiltrate their security footage."

D. Approach

When the Arduino device is powered-on, it immediately begins audibly clicking through the relay device, and sends a serial console message indicating that the message `HELLO` was transmitted. We first assumed that the relay device clicks may be an encoding of Morse code, or some similar encoding. For better measurements of the signals sent to the relay device, we used an oscilloscope to probe the signal sent to the relay device. With the oscilloscope display, we were able to quickly determine that the sequence of signals sent to the relay device did not encode Morse code; the signals did not cleanly encode dots or dashes that might be letters.

Recalling the challenge title and the references in the challenge text, we recognized that the challenge was likely an encoding of SPI protocol messages. We hypothesized that a high signal to the relay device corresponded with a 1-bit in the SPI message, and a low signal corresponded with a 0-bit. Because SPI runs on clock cycles, we needed to determine the length of each clock tick; using the oscilloscope we determined that the shortest interval of a high/low value in the sequence was approximately 200ms, leading us to hypothesize further that each 200ms interval corresponded with one bit in the SPI encoded message. Due to the nature of most SPI protocols using eight 8-bit registers, we determined this length made sense as it would produce 64 bits from the sequence of signals sent to the relay device.

We then went over the whole trace, transcribing the bits:

```
1010 0101 0000 0101 0100 1000
0100 0101 0100 1100 0100 1100
0100 1111 0011 1001
```

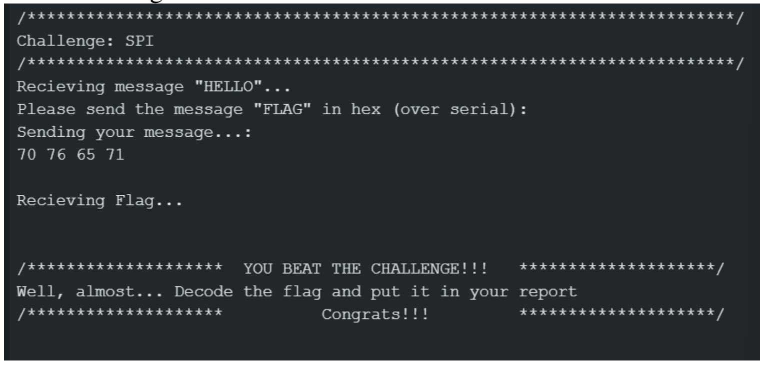
In hex, this is `a5 05 48 45 4c 4c 4f 39`, which when translated to ASCII (omitting the header value and computed CRC-8 value) it would be `HELLO`. Since the serial console output said `Receiving message "HELLO"`, we figured we were on the right track. We were then prompted to send the message `FLAG` in hex back over serial. We took `a5 05` to be a header, and tried sending back `a5 05 46 4c 41 47`. To this, we got `Error: Incorrect Length.` Further experimentation revealed that we were expected to send back exactly 8 bytes. When sending `a5 05 46 4c 41 47 10 10`, we received a different message: `Error: bad CRC.` This told us that the last byte (`39` in the HELLO message) was CRC, specifically CRC-8 (as `39` is only one byte).

We next had to determine what is included in the CRC: was the header included? Computing the CRC-8 of `48 45 4c 4c 4f` gave us `35`, while `a5 05 48 45 4c 4c 4f` gave us the correct CRC of `39`. This told us that the CRC was calculated using not only the payload, but also the header. Thus, we computed the CRC of `a5 05 46 4c 41 47 10` (padding the last character with a line feed) and sent `a5 05 46 4c 41 47 10 51`. This went through, but we got `No response, try again.` Thus we tried padding the last character as a null byte `00`, computed the new CRC, and sent `a5 05 46 4c 41 47 00 21`. This went through and was accepted, and the relay clicked back our encoded flag.

We once again used the oscilloscope to extract the bits from this output, getting

```
1010 0101 0000 1000 0101 0011
0101 0000 0111 1001 0100 0010
0101 0101 0101 0010 0100 1110
0110 0100 0111 1001
```

Decoding the payload from this, we got `SPyBURNdy`, which is our flag.



```
/****************************************************************************/
Challenge: SPI
/****************************************************************************/
Recieving message "HELLO"...
Please send the message "FLAG" in hex (over serial):
Sending your message...:
70 76 65 71

Recieving Flag...


/*******************  YOU BEAT THE CHALLENGE!!!   *******************/
Well, almost... Decode the flag and put it in your report
/*******************           Congrats!!!        *******************/
```

V. Vender Bender

A. Overview

This challenge implemented a mock vending machine that was vulnerable to a fault injection attack. The Arduino sent signals to the relay device, representing the vending machine motor movements. By sending an error message to the Arduino over the serial connection at the appropriate time, we were able to inject faults to the device. Doing this a sufficient number of times (five) resulted in the Arduino sending the flag over the serial connection.

B. Flag

mMmCaNdY

C. Challenge Text

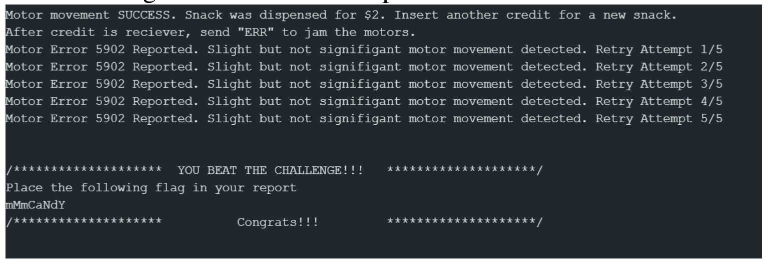
"You roll up to that there vending machine, and it's making a soft hum, like a well-tuned engine, promising you a sweet snack for your taste buds. You eyeball the colorful snacks, deciding if you want them chips for a salty crunch or that chocolate bar for a sweet fix. You've got some coins clinking 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 your hear the gears whirring, and make off with your snack like it's a freshly greased wrench? Then lets take those taste buds on a delicious test drive."

D. Approach

On startup, the Arduino device communicated with the user over serial connection to indicate `motor movement SUCCESS`. The relay device produced audible clicks that coincided with these messages, leading us to believe that the `motor movement` described was referring to the relay device. We confirmed this by connecting an oscilloscope to the SIGNAL wire of the relay device, and observed a 5V high signal sent to the relay for approximately five seconds, followed by a two second period of low signal.

We suspected, based on the prompt from the serial connection and from the challenge text, that we needed to send an `ERR` message to the device at some appropriate time to cause a `Motor Error 5902`. Through trial and error we determined that we had to send an `ERR` message over the serial connection at the beginning of the relay's high voltage signal period to cause a motor error and for the Arduino to retry movement. If the `ERR` message was sent over the serial connection during a period of low voltage signal rather than causing a motor movement error it simply prolonged the period of low voltage. By sending the `ERR` message over the serial connection at the correct time, we caused the period of high signal to end early and drop immediately to a short period of low voltage signal (roughly 200 ms) before returning to a high voltage period. After we repeated this process of injecting the `ERR` message four more times, the Arduino presented the flag by sending it over the serial connection.

Retrospectively, we reason that the challenge could possibly be automated by using a more advanced oscilloscope to trigger a serial `ERR` message when the relay signal transitions to high. Alternatively, we could also have tried using audio cues from the relay clicks to try to time the injection of the fault. Since we did not have an oscilloscope capable of this, we needed to manually send the `ERR` message when we visually observed the signal on the oscilloscope monitor.



```
Motor movement SUCCESS. Snack was dispensed for $2. Insert another credit for a new snack.
After credit is reciever, send "ERR" to jam the motors.
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!!!        ********************/
```

VI. Sock and Roll

A. Overview

This challenge utilizes the microphone and speaker of the Arduino. The Arduino sends a message in French over a serial connection that, when translated, is asking for help before printing to a serial connection that it is `Glad to know things are OK`. While the Arduino is printing over a serial connection it is also playing a sequence of three short tones followed by one long tone from the speaker (`C#`, 1000Hz). We suspect to solve this challenge that we would have to to play a sequence into the microphone at a specific Hz, most likely in relation to the TV show Lost due to the French and message counter being sent over the serial connection by the Arduino being a reference from the TV show.

B. Flag

Unsolved

C. Challenge Text

"In the heart of the whimsical and colorful Happy Socks factory, you find yourselves caught in a captivating and slightly bizarre world. This factory is a maze of vibrant rooms filled with the magic of sock-making machines, piles of socks adorned in all shades and patterns, and, of course, the joyful, dancing sock mascots that are the emblem of this fantastical place. However, what was initially a delightful visit has turned into an unexpected challenge. You have been mysteriously locked inside one of the factory's rooms, with no apparent way out. The laughter and cheerfulness of the factory have given way to a sense of urgency as you realize they need to escape. Luckily, you find the Happy Tap Dancing Socks Message Machine 2000, which seems to be transmitting some strange message. The only way to exit this colorful world and return to reality is to send a distress signal using this cutting-edge messaging technology."

D. Approach

When first connecting to the Arduino we immediately noticed that the speaker was on and outputting a repeated sequence (at 1000 Hz). After some experimentation with an oscilloscope we determined the only other component on the Arduino that was receiving power was the microphone. We determined the microphone was on after `Veuillez nois aider` was sent over the serial connection in the first `Iteration` being sent and would stay on afterwards.

The message being sent over the serial connection is the same on each Iteration:

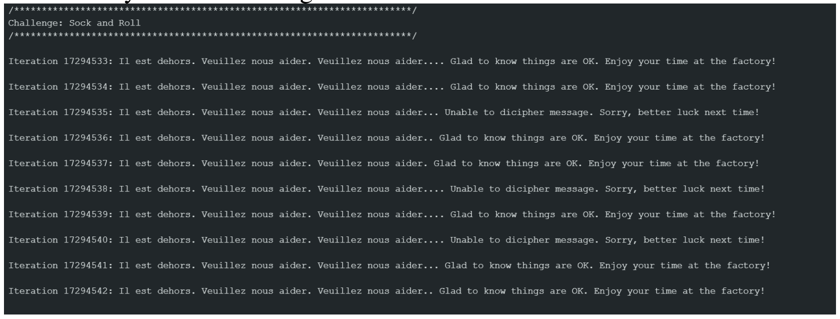
```
Il est dehors. Veuillez nous aider.
 ↪  Veuillez nous aider .... Glad to
 ↪  know things are OK. Enjoy your
 ↪  time at the factory!
```

The serial message would stay the same between iterations except for two things: the amount of periods between `Veuillez nous aider` and `Glad to know`, and the `Iteration` number which increments by one after each `Iteration`. When the message is translated from French it reads as `He is outside. Please help us. Please help us.` We also discovered this message is a reference from a radio broadcast from the TV show Lost, except the iteration numbers in the Lost TV show are different and have different messages for each corresponding number.

Further experimentation revealed that the only way to change the message that was being outputted over the serial connection was to have the speaker's sequence interrupted by unplugging the speaker or playing a different loud noise over the speaker's message. The second half of the serial connection message when the speaker was interrupted would be `Unable to decipher message. Sorry, better luck next time!`

Through experimentation with the oscilloscope we determined that the microphone was only sending the signal to the Arduino over the analog pin (`A0`); we were able to remove the speaker and microphone component entirely from the board and simply connect the signal being sent to the speaker and the analog pin that the microphone would have sent the signal back to the Arduino with.

We were unable to determine what message/sequence needed to be played into the microphone or the analog pin to successfully receive the flag.



```
/***********************************************************************/
Challenge: Sock and Roll
/***********************************************************************/

Iteration 17294533: Il est dehors. Veuillez nous aider. Veuillez nous aider.... Glad to know things are OK. Enjoy your time at the factory!

Iteration 17294534: Il est dehors. Veuillez nous aider. Veuillez nous aider.... Glad to know things are OK. Enjoy your time at the factory!

Iteration 17294535: Il est dehors. Veuillez nous aider. Veuillez nous aider... Unable to dicipher message. Sorry, better luck next time!

Iteration 17294536: Il est dehors. Veuillez nous aider. Veuillez nous aider.. Glad to know things are OK. Enjoy your time at the factory!

Iteration 17294537: Il est dehors. Veuillez nous aider. Veuillez nous aider. Glad to know things are OK. Enjoy your time at the factory!

Iteration 17294538: Il est dehors. Veuillez nous aider. Veuillez nous aider.... Unable to dicipher message. Sorry, better luck next time!

Iteration 17294539: Il est dehors. Veuillez nous aider. Veuillez nous aider.... Glad to know things are OK. Enjoy your time at the factory!

Iteration 17294540: Il est dehors. Veuillez nous aider. Veuillez nous aider.... Unable to dicipher message. Sorry, better luck next time!

Iteration 17294541: Il est dehors. Veuillez nous aider. Veuillez nous aider... Glad to know things are OK. Enjoy your time at the factory!

Iteration 17294542: Il est dehors. Veuillez nous aider. Veuillez nous aider.. Glad to know things are OK. Enjoy your time at the factory!
```