InfoSecIITR: CSAW ESC'23 Final Round Submission

Gyanendra Kumar Banjare, Manas Ghandat, Abhishek Kumar Singh, Priyansh Rathi and Rahul Thakur
Indian Institute of Technology, Roorkee
Roorkee, India

Email: gyanendrabanjare8@gmail.com, manasghandat7099@gmail.com, abhishekkrsingh05kr@gmail.com,
techiepriyansh@gmail.com, rahul.thakur@cs.iitr.ac.in

***Abstract*—In this report, we discuss our approaches and solutions for the CSAW Embedded Security Challenge'23. We utilized various Side Channel Attacks on the provided hardware, ranging from intercepting digital signals, and timing attacks to analyzing the spectrum of audio emitted.**

I. Introduction

The CSAW ESC'23 requires analyzing the functionality of challenges, discovering the vulnerable aspect, and performing side channel attacks on the hardware kit to find the flags. Some of the methodologies we used include using another Arduino to intercept the digital and analog signals, using Audacity to visualize the waveform of captured audio signal and plot it's spectrum and pyserial to automate serial interactions with the hardware.

The attached *.zip* file contains the demonstrations and the relevant files for challenges.

II. Challenges

We now present a brief overview of each challenge, followed by our detailed approach and solution.

A. Week 1

1) All White Party: Username: BarryLNRjltqd PIN Code: 408963512

Overview: The goal of this challenge was to figure out the personal details of an employee. This challenge has two parts. In the initial part, we had to enter the username of the person via serial. In the next part a multifactor authentication (MFA) code was sent to the user. We had to enter the MFA through the keypad.

Approach: When we enter any username, in the case the username is wrong, the haptic sensor is vibrated. We employed the byte by byte side channel attack in order to exploit this challenge. The password checking username is similar to the one given in Listing 1.

The loop breaks out as soon as one non matching byte is detected. This results in a visible change in the timing compared to the trace observed when the byte is correct. Using pyserial, we were able to measure the time difference and can bruteforce each byte to give us the username. The script is presented in Listing 2.

Listing 1: Byte by byte comparison

```
for (uint8_t i = 0; i < sizeof(correct_mem);
i++) {
  if (correct_mem[i] != data[i]) {
    mem_different = 1;
    break;
  }
}
```

Employing this timing based side-channel analysis, we were able to figure out that the first 5 characters of the username have to be "Barry" to qualify as a valid username. Thus, any username of the form "Barry" appended with any random string (possibly the empty string) works as a valid username.

For the MFA pin, we were not provided with any obvious hardware side-channel. But the software itself leaked 5 numbers after entering an incorrect MFA saying that the "Password SHA does not match". After some manual analysis, we found that the five numbers correspond to the first five bytes of the SHA-1 hash digest of the username. And guessing from the failure message, we want the first five bytes of the SHA-1 digest of the pin to match those.

At this point, the challenge finally boils down to finding a pair (s, p) of a string s and 10-digit pin p such that the first five bytes of the SHA-1 hash of the string s appended to "Barry" are equal to the first five bytes of the SHA-1 hash of the pin p.

$$sha1(\text{"Barry"} + s)[0..5] = sha1(p)[0..5]$$

This problem is roughly equivalent to finding hash collisions for a modified hash function with digest length $N = 40$ bits. We can use the classical collision of lists approach to find a collision in approximately $2^{N/2}$ iterations with a high probability. A python script for doing this is presented in Listing 2. The probability of finding a collision in one execution of this script is $1 - (1 - \frac{1}{2^{N/2}})^{2^{N/2}} \approx 0.632$.

Listing 2: Python script to find collision

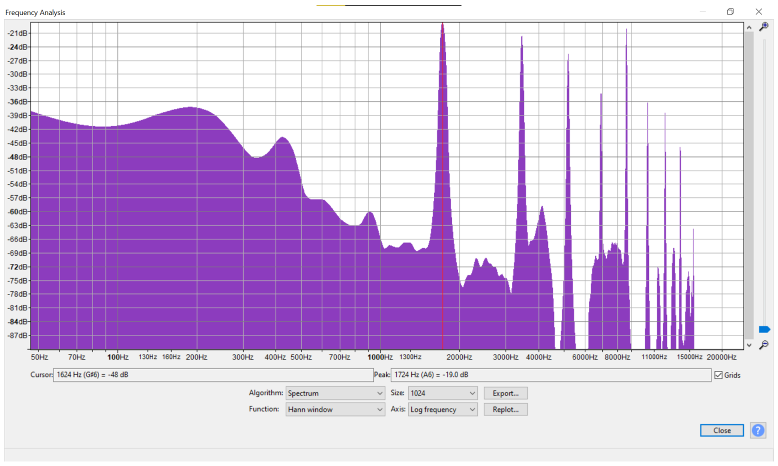
```
from hashlib import sha1
from tqdm import tqdm
import string
import random

charset = string.ascii_letters
digits = string.digits
m = 8

def get_random_append():
  return ''.join(random.sample(charset, m))

def get_random_pin():
  return ''.join(random.choices(digits, k=10))

d = {}
s = set()

n = 2**20

for i in tqdm(range(n)):
  a = get_random_append()
  h = sha1(('Barry'+a).encode()) \
    .hexdigest()[:10]
  d[h] = a
  s.add(h)

for i in tqdm(range(n)):
  p = get_random_pin()
  h = sha1(p.encode()) \
    .hexdigest()[:10]
  if h in s:
    print('Barry'+d[h], p)

# Output: BarryLNRjltqd 4089635127
```

Thus we were able to find a correct pair of username (`BarryLNRjltqd`) and pin (`4089635127`) that allowed us to successfully log in. Along with this pair we were able to find many other such pairs using the same script as Listing 2.

2) ***Bluebox:***

Overview: The goal of this challenge was to mimic the beat which is played when the challenge starts.The challenge accepts 4 numbers which had to be given as input through the keypad followed by a #. The beat which is played is random each time and thus we are not able to brute-force the number. Upon entering a number, a distinct beat would be played by the speaker for every number.

Approach: In order to solve the challenge we recorded the sounds made by each individual digit on the keypad and analyzed the sounds using Audacity. Using audacity we can plot the frequency spectrum of the sound wave recorded and thus can map each number with a corresponding sound frequency which appears as peak in the decibel v/s frequency plot which can be seen in figure 1. In order to minimize the error, we considered the first 3 peaks.The mapping of the keys is as follows:



Cursor: 1624 Hz (G♯6) = -48 dB
Peak: 1724 Hz (A6) = -19.0 dB
☑ Grids
Algorithm: Spectrum
Size: 1024
Function: Hann window
Axis: Log frequency

Fig. 1: Decibel v/s frequency plot of sound wave recorded

Key	Peak 1	Peak 2	Peak 3
1	1725	3446	5147
2	2871	4771	6679
3	3035	5066	7096
4	3206	4277	5349
5	3576	5464	8333
6	3764	6252	8748
7	3927	6542	9163
8	4286	7144	10002
9	4468	7439	10420
0	4629	7714	10803
A	5654	6786	7923
B	4099	5462	6829
C	4806	6409	8018
D	5512	7356	9198

In a similar manner, the frequency of the beat played at the start was recorded and analyzed the different beats against the frequency map which provided us with numbers that we had to enter. Thus after the correct numbers were entered, another beat was played which gave us the flag. The second beat was decoded in the same manner which gave us the flag.

B. Week 2

1) ***Operation SPItFire:***
Flag: SPyBURNd

Overview: The challenge was based on Serial Peripheral Interface (SPI), in which we have to figure out how to communicate with another device and how are the messages sent and received between devices. We first receive ”HELLO” using the communication protocol to figure out how the communication protocol works. We are then required to send the message ”FLAG” using the same protocol after which we receive the actual flag using the same protocol.

Approach: First we analyzed the periodic blinking of Relay’s LED, it seemed that the relay was transmitting binary signals, ON and OFF LED represented 1 and 0 respectively.

We then intercepted the signal on another arduino, and noticed the smallest time interval for which relay is ON, which turns out to be 200ms. We can now read the signal every 200ms, resulting in a stream of binary data. We found the hex data we need to send is in form of

Message Text	Header	Message Length	Message Data	CRC Value
FLAG	A5	04	464C4147	DA

After intercepting and decoding the final message we got the flag as *SPyBURNd*. The relevant scripts can be found at ***????***.

*2) **czNxdTNuYzM**: Overview:* In this challenge the 7 segment display was flashing some numbers and we had to find the next term in sequence. The next which we had to enter was the flag for the challenge. The numbers on the seven segment display were flashing so fast that it was very difficult to directly observe them. Thus we used slow motion video recording on our smart phone in order to analyze the numbers.

Approach: Upon analyzing the video (cite in zip/drive) frame by frame we can see the different numbers that appear on the screen. The dot on the seven segment display acts as a separator between the numbers. Also, due to the rapid change in the 7 segment display, some other segments light up as well as a result of some race between the row and column drivers which leaves a brief time when LED's are driven [1].

After extracting the numbers from the recording we need to find the sequence for which we used OEIS[2]. The sequence we found was as follows:

$$a(n+1) = \begin{cases} \frac{a(n)}{n}, & \text{if } n|a(n) \\ 1, & \text{n=1} \\ a(n) * n & \text{otherwise} \end{cases}$$

The last number which was shown in the 7 segment display was 4056234. Thus the next term of the sequence would be 97349616 which was the flag for this challenge.

C. Week 3

*1) **Sock and Roll**:*
Flag: f0otNOt3

Overview: In this challenge we get a prompt on the serial that says "He is outside. Please help us. Please help us" in french, and the speaker starts buzzing periodically that is some kind of signal. The microphone is active and is listening to the buzzing, and as soon as one periodic signal completes, and there were not any errors, we get "Glad to know things are OK. Enjoy your time at the factory!". We somehow need to send a distress signal to crack this challenge.

Approach: We can hear a long beep followed by three small beeps and again a long beep. We first tried to associate this signal with morse code, and wanted to somehow send SOS signal as a distress signal. We analysed the audio waveform in audacity, and couldn't find any relation to morse code or similar encodings. We then payed attention to the prompt, it seemed that the three small beeps were actually signaling an OK and the 2 long beeps were for syncing the speaker and the microphone. We thought if we only skip the three small beeps we can convey a distress. We first tried to edit the audio, removing the there small beeps and playing it on a computer near the microphone after removing the buzzer, but it didn't work. So we measured the time gap between 2 consecutive long beeps and removed the buzzer for only that time period and we got the flag.

*2) **Vender Bender**:*
Flag: mMmCaNdY

Overview: In this challenge we get a simple fault injection. We can inject fault by entering "ERR" in the serial. According to the challenge description, there is a vending machine, whose motor is presumably controlled by the relay of the hardware. The relay clicks signaling the start of the motor. We have to inject fault at the correct time to crack the challenge.

Approach: If we don't send "ERR" or send it at the wrong time, we get a prompt saying "Motor movement SUCCESS". After some tries, we were able to observe the behaviour of this challenge. The motor is jammed when we enter "ERR" at the same time when relay is turned on. We need to jam the motor five times to get the flag. At first, we manually entered the "ERR" upon hearing the relay click, but it was an inefficient process. Thus we automated this using another arduino to read the relay signal and then sending "ERR" as soon as we get a HIGH signal. The setup we used is at *'week3/VenderBender/setup.jpeg'* and the python notebook is at *'week3/VenderBender/script.ipynb'*.

III. Conclusion

The flags of the challenges are as follows -

A. Week 1

1) All White Party:
`Username: BarryLNRjltqd`
`PIN Code: 4089635127`
2) Bluebox: `B339B009`

B. Week 2

1) Operation SPItFire: `SPyBURNd`
2) czNxdTNuYzM: `97349616`

C. Week 3

1) Sock and Roll: `f0otNOt3`
2) Vender Bender: `mMmCaNdY`

Acknowledgment

Our team would like to thank and acknowledge the efforts of everyone at NYU Tandon and OSIRIS Lab for organizing this event and helping spread cybersecurity awareness.

References

[1] "Assessing and Mitigating Impact of Time Delay Attack: Case Studies for Power Grid Controls". In: (). URL: https://ieeexplore.ieee.org/document/8892729.

[2] "The On-Line Encyclopedia of Integer Sequences". In: URL: https://oeis.org/.