MEMO Number DC01
DATE: 04/28/2022
TO: Challenge Authors
FROM: AgricArtel
SUBJECT: DawgCTF 2022 "Ancient Bits" Writeup

1 INTRODUCTION

This challenge requires the player to decode an RS-232 transmission into its ASCII message. The player is given a csv file containing voltages and times, captured from an oscilloscope.

2 SOLUTION

In order to solve this, the player should have some background knowledge regarding serial communication with RS-232. Below are the steps I took to solve the challenge:

2.1 Steps to Solve

2.1.1 Initial Steps

The csv file '`scope0.csv`' contains two columns: time and voltage. Using this, we can plot this in MATLAB:

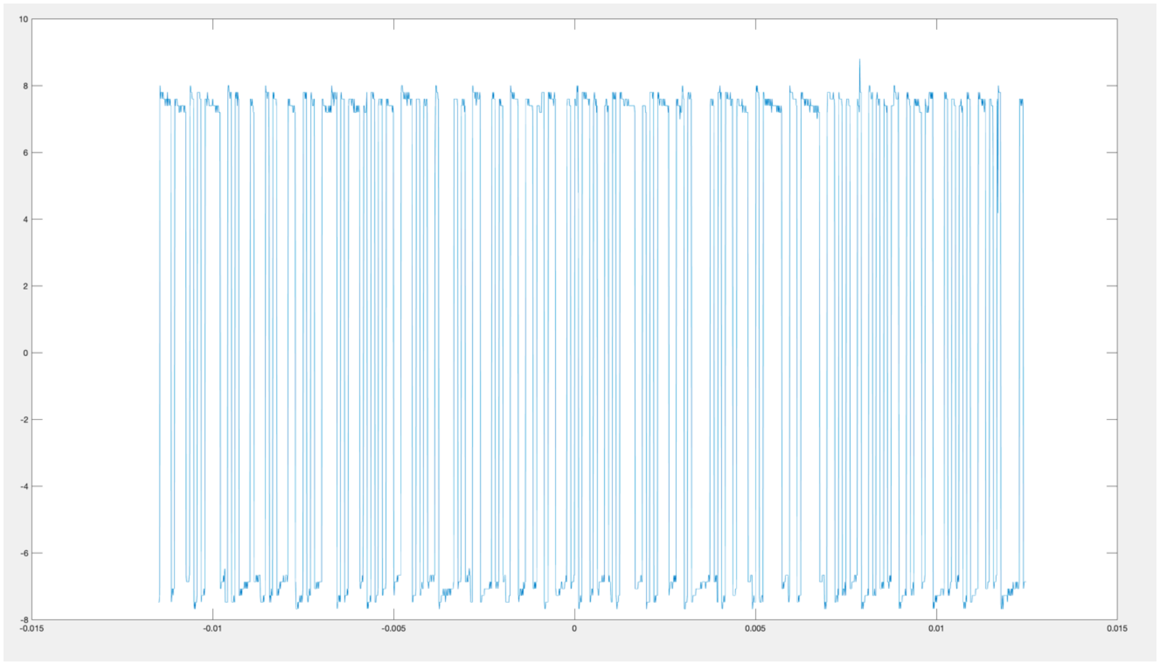

Figure 1: Plot of voltage over time

From this plot, we can see that this data represents some sort of digital communication. Since we only have one waveform, let's assume this is a serial transmission. From the description of the challenge, this data was captured from a machine running MS-DOS 6.7. This is before USB's time, so we can conclude that this serial transmission came from a COM port. Therefore, the serial communication standard used is RS-232.

2.1.2 Converting to Bitstream

Next, I converted the raw voltage data into ones and zeroes in MATLAB. After that, we need to account for the sampling rate of the oscilloscope. Upon closer inspection of the data, each bit of the transmission is made up of about 8-9 samples. I then accounted for this and converted the binary samples into a stream of characters, for each bit of transmission.

Below is the MATLAB script `make_bitstream.m`:

```
%
% DawgCTF 2022 - Ancient Bits
% Craig Stone
%
% NOTE: THIS SCRIPT DEPENDS ON THE DATA ALREADY IMPORTED INTO THE
% WORKSPACE
% Where v is the second column in the csv file.
%

bits = v > 0;
new_index = 1;
current = -1;
samples_per_bit = 8;
sample_index = 1;
bitstream = zeros(250,1);

for i = 1:length(bits)
    % Shift from 1 to 0
    if bits(i) ~= current
        current = bits(i);
        sample_index = 2;
    % Finished sampling one bit
    elseif sample_index == samples_per_bit
        bitstream(new_index) = bits(i);
        new_index = new_index + 1;
        sample_index = 1;
    % In the middle of sampling a bit
    else
        sample_index = sample_index + 1;
    end
end
```

2.1.3 Decoding the Bitstream

I exported the bitstream created in MATLAB to a csv file, where each line had a one or zero corresponding to each respective bit in the transmission. For the final step, I wrote a C++ program to decode the RS-232 bitstream into its ASCII values.

RS-232 uses start and stop bits surrounding each byte. A start bit is a positive voltage, and a stop bit is a negative voltage. This transmission used one stop bit, and 8 bits per byte. Knowing this, we can align the transmission, so we start decoding on a start bit.

The transmitted data is in a Big-Endian format. However, RS-232 has no specification on the endianness of the data. So, this took a little bit of trial and error.

I used a counter to keep counting every tenth bit. For the values greater than one, and less than ten, I combined that bit with the rest of the bits for that respective byte. When the counter reaches ten, the byte is completed and is ready to be pushed into the output string.

Below is the C++ program `Decode_UART.cpp`:

```cpp
#include <stdio.h>
#include <iostream>
#include <vector>

#define FILENAME "bitstream.csv"

int main(int argc, const char * argv[]) {
    FILE* fd = fopen(FILENAME, "r");
    char* buf = (char*) malloc(sizeof(char) * 3);
    char* flag = nullptr;
    size_t len = 3;
    std::vector<char> ascii_stream;
    int i = 1;
    char build = 0;
    uint8_t value = 0;

    if (!fd) {
        return -1;
    }
    while (getline(&buf, &len, fd) != -1) {
        value = (buf[0] - 48);
        if (i == 10) {
            ascii_stream.push_back(~build);
            build = 0;
            i = 0;
        }
        else if (i > 1) {
            value <<= i - 2;
            build |= value;
        }
        i++;
    }
    i = (int) ascii_stream.size();
    flag = (char*) malloc(i * sizeof(char));

    for (int j = 0; j < i; j++) {
        flag[j] = ascii_stream.at(j);
    }
    std::cout << flag << std::endl;

    free(buf);
    free(flag);
    return 0;
}
```

3 CONCLUSION

This challenge should be quite difficult for players inexperienced with signal processing. However, with a little bit of research online about RS-232, I estimate an average completion time of 4 hours. I spent about 2.5 hours completing the challenge.

I predict quite a few support tickets regarding my challenge, so this document should help clarify everything.