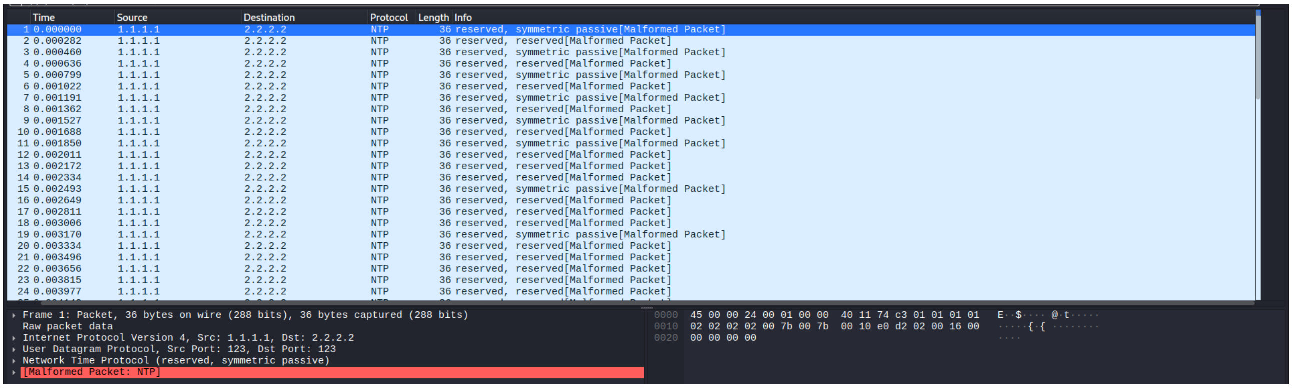
The Clockwork Keylogger

so we are given with a pcap file so we open it with wireshark first
which is filled with malformed NTP packets



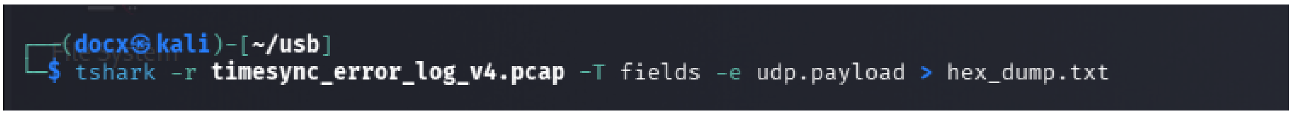
```
   Time       Source        Destination   Protocol  Length Info
 1 0.000000   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
 2 0.000282   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
 3 0.000460   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
 4 0.000636   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
 5 0.000799   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
 6 0.001022   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
 7 0.001191   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
 8 0.001362   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
 9 0.001527   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
10 0.001688   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
11 0.001850   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
12 0.002011   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
13 0.002172   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
14 0.002334   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
15 0.002493   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
16 0.002649   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
17 0.002811   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
18 0.003006   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
19 0.003170   1.1.1.1       2.2.2.2       NTP        36 reserved, symmetric passive[Malformed Packet]
20 0.003334   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
21 0.003496   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
22 0.003656   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
23 0.003815   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
24 0.003977   1.1.1.1       2.2.2.2       NTP        36 reserved, reserved[Malformed Packet]
```

```
› Frame 1: Packet, 36 bytes on wire (288 bits), 36 bytes captured (288 bits)
  Raw packet data
› Internet Protocol Version 4, Src: 1.1.1.1, Dst: 2.2.2.2
› User Datagram Protocol, Src Port: 123, Dst Port: 123
› Network Time Protocol (reserved, symmetric passive)
› [Malformed Packet: NTP]
```

```
0000  45 00 00 24 00 01 00 00  40 11 74 c3 01 01 01 01   E··$····@·t·····
0010  02 02 02 02 00 7b 00 7b  00 10 e0 d2 02 00 16 00   ·····{·{········
0020  00 00 00 00                                        ····
```

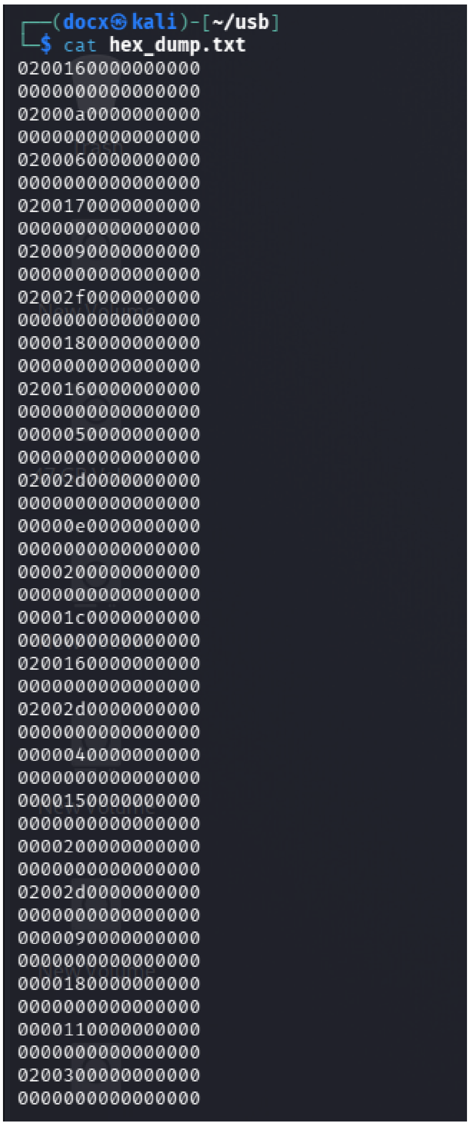
so we cant get any data from this
we use this command to get the hex data from the pcap file

tshark -r timesynce_error_log_v4.pcap -T fields -e udp.payload > hex_dump.txt



```
┌──(docx㉿kali)-[~/usb]
└─$ tshark -r timesync_error_log_v4.pcap -T fields -e udp.payload > hex_dump.txt
```

which gives us the hex values into a file named hex_dump.txt

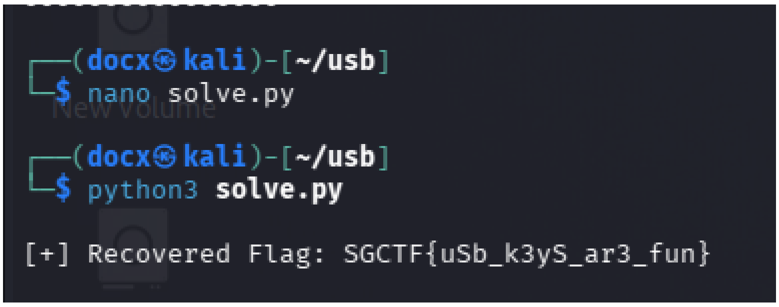


```
┌──(docx㉿kali)-[~/usb]
└─$ cat hex_dump.txt
0200160000000000
0000000000000000
02000a0000000000
0000000000000000
0200060000000000
0000000000000000
0200170000000000
0000000000000000
0200090000000000
0000000000000000
02002f0000000000
0000000000000000
0000180000000000
0000000000000000
0200160000000000
0000000000000000
0000050000000000
0000000000000000
02002d0000000000
0000000000000000
00000e0000000000
0000000000000000
0000200000000000
0000000000000000
00001c0000000000
0000000000000000
0200160000000000
0000000000000000
02002d0000000000
0000000000000000
0000040000000000
0000000000000000
0000150000000000
0000000000000000
0000200000000000
0000000000000000
02002d0000000000
0000000000000000
0000090000000000
0000000000000000
0000180000000000
0000000000000000
0000110000000000
0000000000000000
0200300000000000
0000000000000000
```

Now that we have hex_dump.txt filled with lines of hex codes, we need a script to translate those HID codes back into ASCII.

```
#!/usr/bin/env python3
import sys

# HID Keycode Map (USB HID Usage Tables)
# This maps the 3rd byte (Keycode) to the character.
USB_CODES = {
    0x04: "a", 0x05: "b", 0x06: "c", 0x07: "d", 0x08: "e", 0x09: "f",
    0x0A: "g", 0x0B: "h", 0x0C: "i", 0x0D: "j", 0x0E: "k", 0x0F: "l",
    0x10: "m", 0x11: "n", 0x12: "o", 0x13: "p", 0x14: "q", 0x15: "r",
    0x16: "s", 0x17: "t", 0x18: "u", 0x19: "v", 0x1A: "w", 0x1B: "x",
    0x1C: "y", 0x1D: "z",
    0x1E: "1", 0x1F: "2", 0x20: "3", 0x21: "4", 0x22: "5", 0x23: "6",
    0x24: "7", 0x25: "8", 0x26: "9", 0x27: "0",
    0x28: "<ENTER>", 0x2C: " ", 0x2D: "-", 0x2E: "=", 0x2F: "[",
    0x30: "]", 0x31: "\\", 0x33: ";", 0x34: """, 0x36: ",", 0x37: ".",
    0x38: "/"
}

# Shift Map (When Modifier 0x02 or 0x20 is active)
SHIFT_MAP = {
    "a": "A", "b": "B", "c": "C", "d": "D", "e": "E", "f": "F",
    "g": "G", "h": "H", "i": "I", "j": "J", "k": "K", "l": "L",
    "m": "M", "n": "N", "o": "O", "p": "P", "q": "Q", "r": "R",
    "s": "S", "t": "T", "u": "U", "v": "V", "w": "W", "x": "X",
    "y": "Y", "z": "Z",
    "1": "!", "2": "@", "3": "#", "4": "$", "5": "%", "6": "^",
    "7": "&", "8": "*", "9": "(", "0": ")",
    "-": "_", "=": "+", "[": "{", "]": "}", "\\": "|", ";": ":",
    """": "\"", ",": "<", ".": ">", "/": "?"
}

def decode_usb_traffic(filename):
    output = ""
    try:
        with open(filename, "r") as f:
            lines = f.readlines()
    except FileNotFoundError:
        print(f"Error: Could not open {filename}")
        return

    for line in lines:
        # Clean the line (remove colons if tshark added them, and newlines)
        line = line.strip().replace(":", "")

        # Skip empty lines
        if not line:
            continue
        # Parse hex string to bytes
        try:
            data = bytes.fromhex(line)
        except ValueError:
            continue

        # We need at least 3 bytes (Modifier, Reserved, Keycode)
        if len(data) < 3:
            continue

        modifier = data[0]
        keycode = data[2]

        # If keycode is 0, it's a key release event (ignore it)
        if keycode == 0:
            continue

        if keycode in USB_CODES:
            char = USB_CODES[keycode]

            # Check for Left Shift (0x02) or Right Shift (0x20)
            if modifier == 0x02 or modifier == 0x20:
                if char in SHIFT_MAP:
                    char = SHIFT_MAP[char]
                else:
                    char = char.upper() # Fallback

            output += char
        else:
            print(f"[!] Unknown Keycode: {hex(keycode)}")

    print(f"\n[+] Recovered Flag: {output}\n")

if __name__ == "__main__":
  decode_usb_traffic("hex_dump.txt")
```

save this code as solve.py and run it
which give us the flag



```
┌──(docx㉿kali)-[~/usb]
└─$ nano solve.py

┌──(docx㉿kali)-[~/usb]
└─$ python3 solve.py

[+] Recovered Flag: SGCTF{uSb_k3yS_ar3_fun}
```

SGCTF{uSb_k3yS_ar3_fun}