奇怪的声音

发表于 2021-01-25 分类于 Challenge， 2020， 工业信息安全技能大赛， 济南站
Challenge | 2020 | 工业信息安全技能大赛 | 济南站 | 奇怪的声音

点击此处获得更好的阅读体验

WriteUp来源

来自MO1N战队

题目描述

> *某工控环境中泄露了某些奇怪的声音，你能获取到flag吗?Flag格式为：flag{}。*

题目考点

- *隐写*
- *SSTV*

解题思路

通过binwalk查看图片类型，使用-Me对文件进行分离

分离后发现出现几个文件

ICS.mp3，根据听到的声音猜测是SSTV编码，常见使用是在国际空间站进行图像传输的编码方式，使用的是彩色顺序制，将图像分解为扫描线后再将每条三基色单线，按照一定的次序，将每条三基色单线信号变换为不同的音频信号逐一发送出去，发送顺序一般为红色、蓝色和绿色，三基色中的每一种颜色在发送时都使用相同的速率，因而时间也是相等的，SCOTTIE和MARTIN也是这种方式，SSTV的YC制是为了缩短图片的传送时间出现的。典型的做法是将两路色差信号压缩成一个Y信号的周期来发送，也就是时间压缩，在电脑使用ROBOT36的方式转换SSTV。

方式组	*方式名*	*彩色类型*	*时间(秒)*	*扫描线数*	*单像素占时间 (毫秒)*	*VIS*	*标题行*
SCOTTIE	*S1*	*RGB*	*110*	*240*	*432*	*60*	*16*
S2	*RGB*	*71*	*240*	*275*	*56*	*16*	
S3	*RGB*	*55*	*240/2*	*432*	*52*	*8*	
S4	*RGB*	*36*	*240/2*	*275*	*48*	*8*	
DX	*RGB*	*269*	*240*	*1079*	*76*	*16*	
MARTIN	*M1*	*RGB*	*114*	*240*	*454*	*44*	*16*
M2	*RGB*	*58*	*240*	*214*	*40*	*16*	
M3	*RGB*	*57*	*240/2*	*454*	*36*	*8*	
M4	*RGB*	*29*	*240/2*	*214*	*32*	*8*	
HQ1	*Y+C/2*	*90*	*240*	*535*	*41*	*16*	
HQ2	*Y+C/2*	*112*	*240*	*666*	*42*	*16*	
ROBOT							
BLACK-WHITE	*8*	*BW*	*8*	*120*	*181*	*2*	*16*
12	*BW*	*12*	*120*	*275*	*6*	*16*	
24	*BW*	*24*	*240*	*275*	*10*	*无*	
36	*BW*	*36*	*240*	*431*	*14*	*无*	
ROBOT COLOUR	*12*	*Y+R/B*	*12*	*120*	*183*	*0*	*16*
24	*Y+C/2*	*24*	*120*	*284*	*4*	*16*	
36	*Y+R/B*	*36*	*240*	*275*	*8*	*16*	
72	*Y+C/2*	*72*	*240*	*431*	*12*	*16*	
AVT	*24*	*RGB*	*24*	*120*	*260*	*64*	*16*

方式组	方式名	彩色类型	时间(秒)	扫描线数	单像素占时间(毫秒)	VIS 标题行
90	RGB	90	240	489	68	无
94	RGB	94	200	489	72	无
188	RGB	188	400	489	76	无
125	BW	125	400	489	80	16

PASOKON TV

HIGH RESOLUTION|P3|RGB|203|16+480|208|113|16| |P5|RGB|305|16+480|312|114|16|| |P7|RGB|406|16+480|416|115|16|| |PD|PD50|YC|51|240|286|93|16| |PD90|YC|90|240|532|99|16|| |PD120|YC|126|480|190|95|16|| |PD160|YC|161|384|382|98|16|| |PD180|YC|187|480|286|96|16|| |PD240|YC|248|480|382|97|16|| |PD290|YC|290|600|286|94|16|| |WRAASE SC-1|24|RGB|24|120|||8| |48|RGB|48|240|||16|| |96|RGB|96|240|||16|| |WRAASE SC-2|30|R/2+G+B/2|30|240/2|368|51|8| |60|R/2+G+B/2|60|240|368|59|16|| |120|R/2+G+B/2|120|240|735|63|16|| |180|RGB|180|240|735|55|16|| |J.A.||||480|||| |PROSKAN|J120|RGB|120|240|||16| |WINPIXPRO|GVA125|BW|125|480|||| |GVA125|RGB|125|240|||16|| |GVA250|RGB|250|480||||| |MSCAN|TV1||||||| |TV2||||||||

SCOTTIE2的方式组里RGB的扫描线数，单像素占时间为275，robot36进行对音频图像传输的解码，以下为解码后的图像

以下两份代码合并到一起

```
import numpy as np
import soundfile
from PIL import Image
from scipy.signal.windows import hann

from . import spec
from .common import log_message, progress_bar


def calc_lum(freq):
    """Converts SSTV pixel frequency range into 0-255 luminance byte"""

    lum = int(round((freq - 1500) / 3.1372549))
    return min(max(lum, 0), 255)


def barycentric_peak_interp(bins, x):
    """Interpolate between frequency bins to find x value of peak"""
    # Takes x as the index of the largest bin and interpolates the
    # x value of the peak using neighbours in the bins array
    # Make sure data is in bounds
    y1 = bins[x] if x <= 0 else bins[x-1]
    y3 = bins[x] if x + 1 >= len(bins) else bins[x+1]

    denom = y3 + bins[x] + y1
    if denom == 0:
        return 0  # erroneous
    return (y3 - y1) / denom + x


class SSTVDecoder(object):
    """Create an SSTV decoder for decoding audio data"""
    def __init__(self, audio_file):
        self.mode = None
        self._audio_file = audio_file
        self._samples, self._sample_rate = soundfile.read(self._audio_file)

        if self._samples.ndim > 1:  # convert to mono if stereo
            self._samples = self._samples.mean(axis=1)

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, traceback):
        self.close()

    def __del__(self):
        self.close()

```

```
    def decode(self, skip=0.0):
        """Attempts to decode the audio data as an SSTV signal
        Returns a PIL image on success, and None if no SSTV signal was found
        """

        if skip > 0.0:
            self._samples = self._samples[round(skip * self._sample_rate):]

        header_end = self._find_header()

        if header_end is None:
            return None

        self.mode = self._decode_vis(header_end)

        vis_end = header_end + round(spec.VIS_BIT_SIZE * 9 * self._sample_rate)

        image_data = self._decode_image_data(vis_end)

        return self._draw_image(image_data)

    def close(self):
        """Closes any input files if they exist"""

        if self._audio_file is not None and not self._audio_file.closed:
            self._audio_file.close()

    def _peak_fft_freq(self, data):
        """Finds the peak frequency from a section of audio data"""

        windowed_data = data * hann(len(data))
        fft = np.abs(np.fft.rfft(windowed_data))

        # Get index of bin with highest magnitude
        x = np.argmax(fft)
        # Interpolated peak frequency
        peak = barycentric_peak_interp(fft, x)

        # Return frequency in hz
        return peak * self._sample_rate / len(windowed_data)

    def _find_header(self):
        """Finds the approx sample of the end of the calibration header"""

        header_size = round(spec.HDR_SIZE * self._sample_rate)
        window_size = round(spec.HDR_WINDOW_SIZE * self._sample_rate)

        # Relative sample offsets of the header tones
        leader_1_sample = 0
        leader_1_search = leader_1_sample + window_size

        break_sample = round(spec.BREAK_OFFSET * self._sample_rate)
        break_search = break_sample + window_size

        leader_2_sample = round(spec.LEADER_OFFSET * self._sample_rate)
        leader_2_search = leader_2_sample + window_size

        vis_start_sample = round(spec.VIS_START_OFFSET * self._sample_rate)
        vis_start_search = vis_start_sample + window_size

        jump_size = round(0.002 * self._sample_rate)  # check every 2ms

        # The margin of error created here will be negligible when decoding the
        # vis due to each bit having a length of 30ms. We fix this error margin
        # when decoding the image by aligning each sync pulse

        for current_sample in range(0, len(self._samples) - header_size,
                                    jump_size):
            # Update search progress message
            if current_sample % (jump_size * 256) == 0:
                search_msg = "Searching for calibration header... {:.1f}s"
                progress = current_sample / self._sample_rate
                log_message(search_msg.format(progress), recur=True)

            search_end = current_sample + header_size
            search_area = self._samples[current_sample:search_end]
```

```
126
127             leader_1_area = search_area[leader_1_sample:leader_1_search]
128             break_area = search_area[break_sample:break_search]
129             leader_2_area = search_area[leader_2_sample:leader_2_search]
130             vis_start_area = search_area[vis_start_sample:vis_start_search]
131
132             # Check they're the correct frequencies
133             if (abs(self._peak_fft_freq(leader_1_area) - 1900) < 50
134                and abs(self._peak_fft_freq(break_area) - 1200) < 50
135                and abs(self._peak_fft_freq(leader_2_area) - 1900) < 50
136                and abs(self._peak_fft_freq(vis_start_area) - 1200) < 50):
137
138                 stop_msg = "Searching for calibration header... Found![:>4}"
139                 log_message(stop_msg.format(' '))
140                 return current_sample + header_size
141
142         log_message()
143         log_message("Couldn't find SSTV header in the given audio file",
144                     err=True)
145         return None
```

```
1
2   def _decode_vis(self, vis_start):
3       """Decodes the vis from the audio data and returns the SSTV mode"""
4
5       bit_size = round(spec.VIS_BIT_SIZE * self._sample_rate)
6       vis_bits = []
7
8       for bit_idx in range(8):
9           bit_offset = vis_start + bit_idx * bit_size
10          section = self._samples[bit_offset:bit_offset+bit_size]
11          freq = self._peak_fft_freq(section)
12          # 1100 hz = 1, 1300hz = 0
13          vis_bits.append(int(freq <= 1200))
14
15      # Check for even parity in last bit
16      parity = sum(vis_bits) % 2 == 0
17      if not parity:
18          raise ValueError("Error decoding VIS header (invalid parity bit)")
19
20      # LSB first so we must reverse and ignore the parity bit
21      vis_value = 0
22      for bit in vis_bits[-2::-1]:
23          vis_value = (vis_value << 1) | bit
24
25      if vis_value not in spec.VIS_MAP:
26          error = "SSTV mode is unsupported (VIS: {})"
27          raise ValueError(error.format(vis_value))
28
29      mode = spec.VIS_MAP[vis_value]
30      log_message("Detected SSTV mode {}".format(mode.NAME))
31
32      return mode
33
34  def _align_sync(self, align_start, start_of_sync=True):
35      """Returns sample where the beginning of the sync pulse was found"""
36
37      # TODO - improve this
38
39      sync_window = round(self.mode.SYNC_PULSE * 1.4 * self._sample_rate)
40      align_stop = len(self._samples) - sync_window
41
42      if align_stop <= align_start:
43          return None  # Reached end of audio
44
45      for current_sample in range(align_start, align_stop):
46          section_end = current_sample + sync_window
47          search_section = self._samples[current_sample:section_end]
48
49          if self._peak_fft_freq(search_section) > 1350:
50              break
51
52      end_sync = current_sample + (sync_window // 2)
53
54      if start_of_sync:
```

```
        return end_sync - round(self.mode.SYNC_PULSE * self._sample_rate)
    else:
        return end_sync

def _decode_image_data(self, image_start):
    """Decodes image from the transmission section of an sstv signal"""

    window_factor = self.mode.WINDOW_FACTOR
    centre_window_time = (self.mode.PIXEL_TIME * window_factor) / 2
    pixel_window = round(centre_window_time * 2 * self._sample_rate)

    height = self.mode.LINE_COUNT
    channels = self.mode.CHAN_COUNT
    width = self.mode.LINE_WIDTH
    # Use list comprehension to init list so we can return data early
    image_data = [[[0 for i in range(width)]
                   for j in range(channels)] for k in range(height)]

    seq_start = image_start
    if self.mode.HAS_START_SYNC:
        # Start at the end of the initial sync pulse
        seq_start = self._align_sync(image_start, start_of_sync=False)
        if seq_start is None:
            raise EOFError("Reached end of audio before image data")

    for line in range(height):

        if self.mode.CHAN_SYNC > 0 and line == 0:
            # Align seq_start to the beginning of the previous sync pulse
            sync_offset = self.mode.CHAN_OFFSETS[self.mode.CHAN_SYNC]
            seq_start -= round((sync_offset + self.mode.SCAN_TIME)
                               * self._sample_rate)

        for chan in range(channels):

            if chan == self.mode.CHAN_SYNC:
                if line > 0 or chan > 0:
                    # Set base offset to the next line
                    seq_start += round(self.mode.LINE_TIME *
                                       self._sample_rate)

                # Align to start of sync pulse
                seq_start = self._align_sync(seq_start)
                if seq_start is None:
                    log_message()
                    log_message("Reached end of audio whilst decoding.")
                    return image_data

            pixel_time = self.mode.PIXEL_TIME
            if self.mode.HAS_HALF_SCAN:
                # Robot mode has half-length second/third scans
                if chan > 0:
                    pixel_time = self.mode.HALF_PIXEL_TIME

                centre_window_time = (pixel_time * window_factor) / 2
                pixel_window = round(centre_window_time * 2 *
                                     self._sample_rate)

            for px in range(width):

                chan_offset = self.mode.CHAN_OFFSETS[chan]

                px_pos = round(seq_start + (chan_offset + px *
                               pixel_time - centre_window_time) *
                               self._sample_rate)
                px_end = px_pos + pixel_window

                # If we are performing fft past audio length, stop early
                if px_end >= len(self._samples):
                    log_message()
                    log_message("Reached end of audio whilst decoding.")
                    return image_data

                pixel_area = self._samples[px_pos:px_end]
                freq = self._peak_fft_freq(pixel_area)

```

```
                image_data[line][chan][px] = calc_lum(freq)

        progress_bar(line, height - 1, "Decoding image...")

    return image_data
def _draw_image(self, image_data):
    """Renders the image from the decoded sstv signal"""

    # Let PIL do YUV-RGB conversion for us
    if self.mode.COLOR == spec.COL_FMT.YUV:
        col_mode = "YCbCr"
    else:
        col_mode = "RGB"

    width = self.mode.LINE_WIDTH
    height = self.mode.LINE_COUNT
    channels = self.mode.CHAN_COUNT

    image = Image.new(col_mode, (width, height))
    pixel_data = image.load()

    log_message("Drawing image data...")

    for y in range(height):

        odd_line = y % 2
        for x in range(width):

            if channels == 2:

                if self.mode.HAS_ALT_SCAN:
                    if self.mode.COLOR == spec.COL_FMT.YUV:
                        # R36
                        pixel = (image_data[y][0][x],
                                 image_data[y-(odd_line-1)][1][x],
                                 image_data[y-odd_line][1][x])

            elif channels == 3:

                if self.mode.COLOR == spec.COL_FMT.GBR:
                    # M1, M2, S1, S2, SDX
                    pixel = (image_data[y][2][x],
                             image_data[y][0][x],
                             image_data[y][1][x])
                elif self.mode.COLOR == spec.COL_FMT.YUV:
                    # R72
                    pixel = (image_data[y][0][x],
                             image_data[y][2][x],
                             image_data[y][1][x])
                elif self.mode.COLOR == spec.COL_FMT.RGB:
                    pixel = (image_data[y][0][x],
                             image_data[y][1][x],
                             image_data[y][2][x])

            pixel_data[x, y] = pixel

    if image.mode != "RGB":
        image = image.convert("RGB")

    log_message("...Done!")
    return image
```

Flag

```
1 flag{no32dpi3194dof2}
```

 # Challenge # 2020 # 工业信息安全技能大赛 # 济南站
工控协议数据分析
无人机shell利用