

# Is there an Echo?

---

## Challenge Details

**Description:** Maybe next time you should record your music in an acoustically treated space.

**Category:** Forensics

**Attached Files:** [256.wav](#)

## Research

The attached file is a WAV file. On playing it, we can hear a complex composition with guitar strums and a loud bassy voice in the background. As one could guess from the category and the challenge description, the sound file is hiding something in its echoes. A simple Google Search revealed a lot of papers about Echo Hiding and Steganography in Audio files.

Some sample papers we encountered:

- [Experiments with and Enhancements to Echo Hiding](#)
- [Echo Hiding](#)

We also found a presentation from [Columbia University](#) which explains the concept of Echo Hiding in Audio files. This was by far the easiest way to understand what was really happening.

We learnt that looking at the Cepstrum of the audio file would allow us to see the embedded echoes easily. We googled and found out this could be done easily on an audio forensic old man's favourite tool - Audacity.

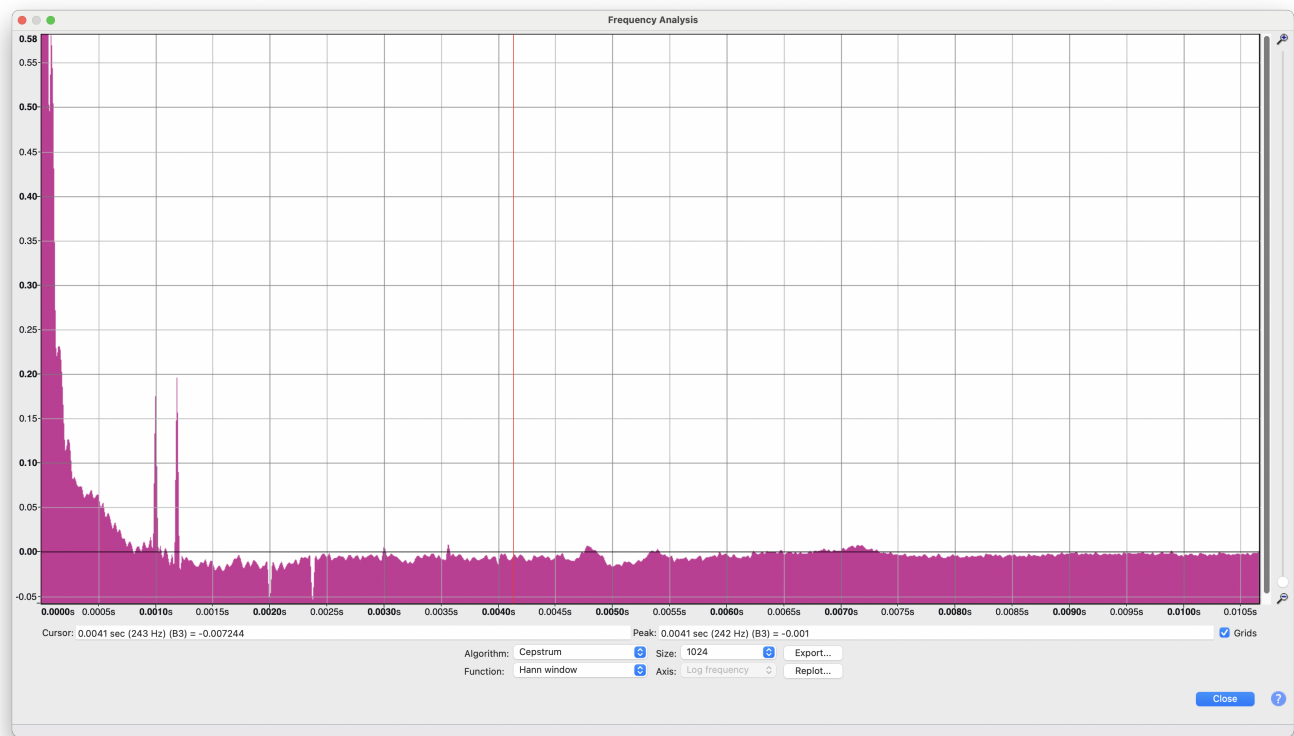
We also found a repository with this very particular method of Echo Hiding implemented in MATLAB. The repository can be found [here](#).

## Analysis

We followed these steps to look at the Cepstrum of the audio file:

1. Open the WAV file in Audacity.
2. Select the entire audio.
3. Go to **Analyze -> Plot Spectrum**.
4. In the **Plot Spectrum** window, select **Cepstrum** in the **Algorithm** dropdown.

We can now see the following graph



We can see two peaks of outliers at approx 0.0010 and 0.0012 seconds. These should be the deltas at which the messages are embedded.

To look at the exact deltas, we exported the Cepstrum data to a TXT file and inspected it. We found two entries around those deltas which had levels of 0.18 and 0.19, about 10 times higher than the rest of the data. These are underlined in green in the below screenshot.

Lag (seconds)	Frequency	Level
0.000021	48000.000000	2.179618
0.000042	24000.000000	1.569559
0.000063	16000.000000	0.535822
0.000083	12000.000000	0.581637
0.000104	9600.000000	0.498231
0.000125	8000.000000	0.241910
0.000146	6857.142857	0.231387
0.000167	6000.000000	0.226982
0.000188	5333.333333	0.178086
0.000208	4800.000000	0.115562
0.000229	4363.636364	0.126957
0.000250	4000.000000	0.120996
0.000271	3692.307692	0.084299
0.000292	3428.571429	0.085354

0.000292	3420.1571429	0.000334
0.000313	3200.000000	0.075674
0.000333	3000.000000	0.074005
0.000354	2823.529412	0.074065
0.000375	2666.666667	0.061070
0.000396	2526.315789	0.066533
0.000417	2400.000000	0.064948
0.000438	2285.714286	0.071167
0.000458	2181.818182	0.060596
0.000479	2086.956522	0.064411
0.000500	2000.000000	0.064478
0.000521	1920.000000	0.051002
0.000542	1846.153846	0.056142
0.000562	1777.777778	0.039508
0.000583	1714.285714	0.045440
0.000604	1655.172414	0.037880
0.000625	1600.000000	0.026797
0.000646	1548.387097	0.034793
0.000667	1500.000000	0.023366
0.000687	1454.545455	0.026746
0.000708	1411.764706	0.015691
0.000729	1371.428571	0.016287
0.000750	1333.333333	0.006880
0.000771	1297.297297	0.013144
0.000792	1263.157895	0.008376
0.000812	1230.769231	-0.000976
0.000833	1200.000000	0.005880
0.000854	1170.731707	0.007077
0.000875	1142.857143	0.001431
0.000896	1116.279070	0.012688x
0.000917	1090.909091	0.006812
0.000937	1066.666667	0.006312
0.000958	1043.478261	0.018996

0.000979	1021.276596	0.012872
0.001000	1000.000000	0.180035
0.001021	979.591837	0.013467
0.001042	960.000000	0.009082
0.001063	941.176471	-0.006409
0.001083	923.076923	0.005432
0.001104	905.660377	-0.000884
0.001125	888.888889	-0.013610
0.001146	872.727273	0.000347
0.001167	857.142857	-0.002470
0.001187	842.105263	0.199845
0.001208	827.586207	-0.008427
0.001229	813.559322	-0.003399
0.001250	800.000000	-0.008407
0.001271	786.885246	-0.004311
0.001292	774.193548	-0.008929
0.001313	761.904762	-0.007802
0.001333	750.000000	-0.011881
0.001354	738.461538	-0.015232
0.001375	727.272727	-0.015088
0.001396	716.417910	-0.010371
0.001417	705.882353	-0.011626
0.001437	695.652174	-0.007357

The exact deltas were 0.001000 and 0.001187 seconds. To calculate the number of samples by which they are offset, we can multiply them with the sampling rate i.e. 48000 Hz.

$0.001000 * 48000 = 48 \text{ samples}$   $0.001187 * 48000 = 57 \text{ samples}$

We can now use these offsets to extract the hidden message from the audio file. The only thing we don't know, however, is the length of the message. From the name of the file, I could guess that it had 256 characters. But after a lot of trials and failed experiments, I finally found out (a very irritating) relationship:

```
data, sample_rate = librosa.load("256.wav", sr=None)
N = 256
L = len(data) // N
```



Now, we are fully equipped to extract the message from the audio file.

## Exploit

We used the repository we had found earlier and translated the code into python. The script can be found [here](#).

```
import numpy as np
import librosa

FILE_PATH = '256.wav'
N = 256
DELAY_1 = 48
DELAY_2 = 57

class EchoDecoder:
    def __init__(self, signal, chunk_size, freq_d0, freq_d1):
        self.signal = signal
        self.chunk_size = chunk_size
        self.freq_d0 = freq_d0
        self.freq_d1 = freq_d1
        self.num_chunks = len(signal) // chunk_size
        self.chunked_signal = self.signal[:self.num_chunks *
self.chunk_size].reshape(self.num_chunks, self.chunk_size)

    def _compute_rceps(self, chunk):
        spectrum = np.fft.fft(chunk)
        log_spectrum = np.log(np.abs(spectrum) + 1e-6)
        rceps = np.fft.ifft(log_spectrum).real
        return rceps

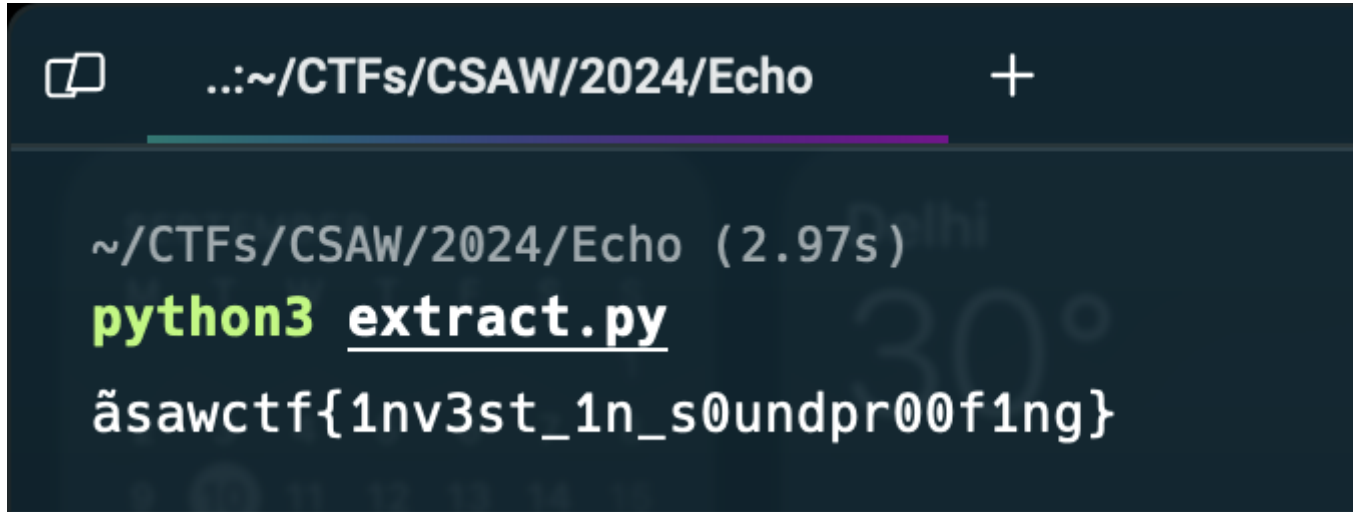
    def _decode_chunk(self, chunk):
        rceps = self._compute_rceps(chunk)
        return int(rceps[self.freq_d1] > rceps[self.freq_d0])

    def decode(self):
        binary_data = [str(self._decode_chunk(chunk)) for chunk in
self.chunked_signal]
        num_bytes = self.num_chunks // 8
        binary_data_chunks = np.array(binary_data[:8 *
num_bytes]).reshape(num_bytes, 8)
        ascii_chars = [chr(int(''.join(bits), 2)) for bits in
binary_data_chunks]
        message = ''.join(ascii_chars)
        return message

def main():
    data, sample_rate = librosa.load(FILE_PATH, sr=None)
    L = len(data) // N
    decoder = EchoDecoder(data, L, DELAY_1, DELAY_2)
    decoded_message = decoder.decode()
```

```
print(decoded_message)

if __name__ == "__main__":
    main()
```

A terminal window with a dark background. The title bar shows a window icon, the path '..:~/CTFs/CSAW/2024/Echo', and a plus sign. The terminal content shows the current directory as '~/.CTFs/CSAW/2024/Echo' with a load time of '(2.97s)'. The command 'python3 extract.py' has been executed, resulting in the output 'csawctf{1nv3st\_1n\_s0undpr00f1ng}'.

```
..:~/CTFs/CSAW/2024/Echo +

~/CTFs/CSAW/2024/Echo (2.97s)
python3 extract.py
csawctf{1nv3st_1n_s0undpr00f1ng}
```

## Final Flag

csawctf{1nv3st\_1n\_s0undpr00f1ng}