**Problem Statement**  
           The title of our project is Biometrics Hacking: Voice Extraction. Our goal is to compromise an individual’s identity through voice.  Here, we try to generate a fake voice identification by creating a passphrase using voice samples.  The manual procedure is too tedious and inaccurate. So we try to design a program to automatically extract the passphrase from voice samples.   
          Input is the voice samples in .wav format of the person to be impersonated and the passphrase. The output is the .wav file of the passphrase. We make use of Java speech APIs and a speech recognition system (which is written entirely in Java) called Sphinx-4 SDK to extract the required words from the passphrase.

**Experimentation**

In order to evaluate our Voice Extraction project, we ran several experiments in which we tried to convert different phrases to audio from different sample input files. Since KWS is the function we use from Sphinx-4 without editing anything, we don’t have any explanation on how it works as well as any comment on how it performs. All we could do is trying to improve the input files and manipulate the timestamps returned from KWS to achieve the best result.

First, we tested the program by making it recognize individual words. Our program used KWS to determine all the occurrence of a particular word within the sample files. It returned a list of timestamps which denoted the start time and the end time of the keyword for each occurrence. Then the program selected the timestamps of the first occurrence and calculated its duration. By using sample rate and number of bytes per frame, the program determined how many important bytes of the word and how many bytes should be skip until the word was spoken in the voice sample.

Next, the sample file was broken down into bytes and using the number of skipped bytes and the number of important bytes, the program was able to extract the keyword and write it out to a new file. The new file would have the same exact format of the input file.

After trying to extract individual words, we proceeded to extract multiple words from a set of sample files in order to ensure that we were able to produce an audio file with a coherent sentence. So after extracting each individual word of the phrase, they were combined together into one AudioInputStream and the stream was written out to a new file. There was one problem. If we just used the result from KWS, all the words would be spoken continuously without any break between one to another. This would make the output file sound unnatural. Therefore, we decided to add a padding of 0.05s on each side of the word to make it more natural. An ammount of padding bytes was subtracted from the number of skipped bytes and twice of that amount was added to the number of important bytes.

Next, we moved on to test the sample file provided by the Voice Hacking Contest. However, Thong’s laptop couldn't finish running the program when passing in those sample files. We waited for more than 20 minutes until deciding to turn it off. Thinking that the duration and the size of the file were the problem, a function to split files into 3 minutes sections was added in order to optimize the performance.

The next problem was that the program couldn't recognize any word from the sample file in the Voice Hacking contest. Even thought we tried to increase their amplitude through Audacity, still no word was recognized. Then we realized that the sample file from KWS had the audio format of 16000 Hz, 2 bytes/frame and mono and one of the .jar file that we needed to add for the KWS to work had the name of “WSJ\_8gau\_13dCep\_16k\_40mel\_130Hz\_6800Hz”. Therefore, a function to convert input files to the same format of 16000 Hz, 2 bytes/frame and mono was added in order to convert voice samples before passing it to KWS.

Lastly, we noticed that if we were trying to extract smaller words with one or two syllables, the application often extracted a sub portion of a larger word that may have similar sounds within it. For example, searching for the word “knowledge” might find the timestamps for the word “acknowledged”. In order to find the entire word rather then find partial words, we decided to modify a particular file called “NoSkipGrammar.java” in the Sphinx-4 source code. The file built up the grammar for a phrase in order to do a search. We made some modifications to add a <silence> grammar at the beginning and the end of a Key word in hope that the KWS would not find parts of words.

**Result and Analysis**

The program was finished and all the functions worked properly, but the results were not very accurate. The results for each improvement were saved and can be examined in the .zip file.

First, for individual word experiments, the program was able to extract each word finely with some exception. For words with its next word pronounced similar to it, the program extracted both of them. For example, while extracting “quantum”, because “number” was next to it and “tum” and “num” were pronounced similar, the result was “quantum-num”. Also, words with “s” at the end were not good either. “This” became “there is”, and “symbolizes” become “symbolize as”. As for explaining why these things happens, probably because of how Sphinx-4 defines the way of pronouncing each syllable and also the quality of Java Speech API.

Second, when extracting a whole phrase, without the padding at first, the result was that each word was spoken continuously next to each other without break in between, making it sound very unnatural. But after the padding, there were breaks between words around 0.1 second, making the whole phrase sounded naturally and much better.

Now, after adding the functions to split and to convert input files to the optimal format, the program seemed to run a little faster. Before, when running it with sample files from the Voice Hacking contest, the program could not recognize anything, even unknown or silence. With those functions, the program could run on Thong’s laptop and recognize words now. Still, the result was not very accurate. We also tried to record one of our team member voice, Govind, that had the similarity score around 10 when he tried to impersonate the person in the contest by just calling the system and tested against the program. The result was similar, still with low accuracy.. Both of them had one similar way of speaking: all the words they spoke, or murmured actually, seemed to stick together, no clear break between them. Without the .doc files provided with the sample files, we could hardly figure out what he was reading. Therefore, we believed this was the cause of the problem.

For the improvement of editing the NoSkipGrammar.java in order to make KWS not to find parts of words, the result didn’t have any noticeable change, unfortunately.

**Problem Encounter**

There were quite a few problem we have encountered. Some we tried to fix, but some we don’t know how to.

First, the quality of the voice samples is a big problem. If the voice of the person is not clear enough, or the words spoken run together then the application has a hard time spotting the key words. The voice samples of the Voice Hacking contest or Govind’s voice were similar and, the program had a hard time recognizing accurately.

Second, the KWS gives out multiple occurrences when the word is spoken in a voice sample with a high degree of false positive. Some of them may be right, and some of them may be wrong. But we have no way to determine which instances are the right ones, or which instances is the best match out of all the occurrences. If the program takes each of those occurrences and run the speech recognition against it, the result is still the same. The only way of determining it is by listening to it ourselves, which defiles the automatic point of the program and our original goal.

Third, even though we tried to find a way to improve KWS from Sphinx-4 SDK by looking for how it recognizing words, there was no result from that. We were able to find a file in which Sphinx-4 SDK defines how to pronounce each and every syllable, we weren’t sure how to edit or improve them. We believe that it would take much more time to be able to understand all the components in KWS in order to edit and improve it.

Lastly if a key word is not available in the sample audio files, then we are unable to extract it and thus omit it from the result. A worse case scenario is when the word is absent from the sample file but the Sphinx-4 SDK has a false match.