Norrathep Rattanavipanon

Erik Henriquez

18th March 2016

**Advanced Operating Systems: Project Final Report**

**Project Goal**

Our project aims to simplify the method of recording and storing audio that is being sent from any application to a sound card. For this project, we intend to test our program by recording audio from the Spotify music player on a Linux operating system.

**Motivation**

With the rise of media streaming services like Netflix and Spotify, users are more open to pay for monthly access to copyrighted media. Unlike media files in a user’s local storage, streamed media is encrypted and limits the user’s access to their media. Our team wanted to learn more about how this is done on the user client and how to circumvent these protection measures.

**Design**

Our main objective was to record audio streaming from Spotify. To do this, we decided to run Spotify on a computer running Ubuntu 14.04 LTS, which runs atop Linux 4.4.1. We chose to use an operating system based on the Linux kernel so we could have access to the audio stream from the application level down to the kernel level.

After researching the various audio interfaces available on a Linux system, we determined that using the Advanced Linux Sound Architecture (ALSA) framework would work best for our project. It is one of the most popular sound interfaces on Linux, thus making our project easily replicable. Additionally, it has two layers that can be edited: the Kernel layer and the API layer.

By having Spotify direct its audio stream to ALSA, we were able to determine that the Spotify application itself decrypted the audio files before outputting to the operating system’s sound interface. With this in mind, we chose to attempt recording the decrypted audio using three different implementations.

**Implementation**

**Method 1 - Modification at Kernel Layer**

ALSA driver components are built into the Linux kernel source code, which is publicly available at [www.kernel.org](http://www.kernel.org). For this method, we downloaded and worked with Linux version 4.4.1. After searching through the kernel source code, we learned that the function responsible for audio playback is snd\_pcm\_playback\_ioctl1 in *linux/sound/core*. We added our code to this function, allowing us to capture the audio buffer and save it to a file as a raw format. This saved file can be processed to other playable formats (e.g. mp3, wav) later. Here’s a simplified version of our modification for the kernel method.

*static int snd\_pcm\_playback\_ioctl1(..., unsigned int cmd, void \_\_user \*arg)  
{  
 ...  
 switch (cmd) {  
 case SNDRV\_PCM\_IOCTL\_WRITEI\_FRAMES:  
 {  
 ...*

*result = snd\_pcm\_lib\_write(substream, xferi.buf, xferi.frames);  
 \_\_put\_user(result, &\_xferi->result);  
 struct file \*rfp = file\_open("out.raw");  
 file\_write(rfp, (char\*) xferi.buf, xferi.frames);*

*...*

*}*

*...*

*}*

**Method 2 - Modification at API Layer**

ALSA also published their API source code on their website (<ftp://ftp.alsa-project.org/pub/lib/>). With access to this code, we can also achieve the same goal by working at the API level. We modified the *snd\_pcm\_writei* function in *src/pcm/pcm.c* such that the audio buffer is captured and saved to a file while being played on a computer. Here’s a simplified version of our modification.

*snd\_pcm\_sframes\_t snd\_pcm\_writei(..., const void \*buffer, ...)*

*{*

*...*

*saveToFile(buffer)*

*return \_snd\_pcm\_writei(pcm, buffer, size);*

*}*

**Method 2.5 - Modification at API Layer but Without Right Access**

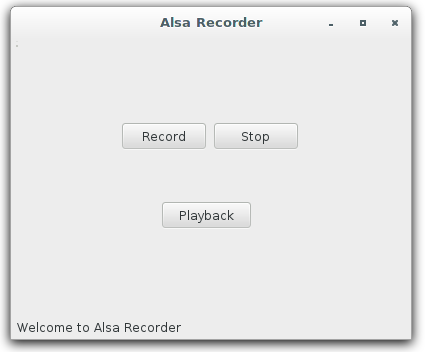
So far, we assume that we have full access to API source code, including modifying and recompiling it as many times as we want. However, we want to show that we do not require root privilege in order to capture protected streaming audio data; we only need to know what API an application uses for audio playback. By using an LD\_PRELOAD trick and *dlsym*, we can replace the original *snd\_pcm\_writei* with our new function at runtime. As a result, we can have full access to the incoming raw audio data and can do whatever we want with it, without modifying API source code at all.

Here’s a pseudocode implementation of this technique:

*static snd\_pcm\_sframes\_t (\*wrap)(..., const void \*buffer, ...) = NULL;*

*snd\_pcm\_sframes\_t snd\_pcm\_writei(..., const void \*buffer, ...)   
{  
 fwrite(buffer, sizeof(buffer[0]), size, fp);  
 wrap = dlsym(RTLD\_NEXT, "snd\_pcm\_writei"); // get the real writei fn  
 return wrap(pcm,buffer,size);  
}*

**GUI**



We also created a simple GUI to easily control our recording program. The Alsa Recorder GUI gives the user the ability to execute the recording scripts at the click of a button, as well as playback the recorded files. Whether our modifications are made through the Kernel, API, or preload method, this application can control the recording of a single application’s audio output.snd\_pcm\_writei

In order to perform correctly, the GUI executable has to be located in the same folder as the *start* and *stop* scripts. This folder will also contain the recorded audio, so the GUI’s playback function will search for the file in the same directory.

**Evaluation**

We tested all of our methods by trying to record local audio data as well as protected streaming audio data from Spotify. All of the implementations are able to successfully capture the audio data even when the data is encrypted at the network layer. Moreover, the recorded file has the same quality as it was produced.

The Alsa Recorder GUI is a very basic interface for our program. Currently, the location of the scripts it uses are hard-coded. One of our future goals is to expand the configuration options of this GUI, so the script locations and the location of the recorded file can be changed during run-time. We would also like to eventually include a Mixer view that would allow us to control the recording of multiple applications simultaneously.

**Closing Remarks**

This project gave us a good idea of how encrypted data can easily be captured at various levels of the operating system. In the future, we’d also like to look into the possibility of capturing encrypted video (such as from Netflix) in a similar manner, or discover new methods of doing this. Finally, it would be of great interest to us, as well as the media streaming companies, if we could find a method of preventing the methods of circumvention that we presented in this paper.