4.2 Analysis Methodology

To understand what happens when the user presses the mute button on desktop VCA clients, we utilize various OS-based tools to trace audio data as it is transferred from the operating system to the app. Our objective is not just to establish whether the app has permission to access the microphone when muted. Instead, we aim to understand whether the app actually reads microphone data when the user is muted.

Linux

Audio data transfer from the Linux kernel to the VCAs is mediated through PulseAudio and ALSA. ALSA is a kernel subsystem that provides a kernel-level interface to the audio hardware, and PulseAudio is a userland process that interfaces with ALSA and provides higher-level features like mixing and multiplexing. All the VCAs we studied interface with the userland PulseAudio process.

To intercept audio data in transit from PulseAudio to a VCA, we use the DynamoRIO runtime code manipulation system [1], which allows us to inject foreign code into a running process. Our additional code, written in C, is called each time a fresh buffer of microphone data arrives from PulseAudio. We write the audio buffer's address in the process's memory space to a log file. We then trace the buffer addresses from the log using IDA Pro. The contents of the buffer are the raw audio bytes from the microphone. DynamoRIO oversees the process's execution by loading and running modified basic blocks one at a time, which substantially slows the app's execution, occasionally causing it to crash.

Windows

Although it is possible to track microphone access by monitoring the system registry [22], we were not able to track transfers in real time from the microphone to the VCA. The registry only records times at which an app opens or closes a connection to an audio device. The OS registry—linked to a visual indicator in the system tray—does not distinguish detailed API calls which encode information about whether a VCA is reading audio data or accessing status flags about microphone activity.For fine-grained and detailed information, we intercept syscalls from the VCA to the operating system.

In Windows 10, syscalls are obfuscated behind a userland API library which acts as an intermediary between the apps and the OS. The Windows API library is similar to the Linux/Unix C library syscall wrappers, except that there is no one-to-one mapping between the parameters that the app passes to the API and the parameters that the API passes to the OS. Instead, the API functions as a higher-level wrapper around system calls, and there is no official documentation available from Microsoft detailing how to call the operating system directly.

Windows implements many special-purpose API functions for actions like accessing the microphone, which in Linux and Unix are all handled as files. We develop a two-step process to trace audio data in transit from the Windows OS to the native VCAs. First, we use a tool called API Monitor [12] to instrument the userland API with hooks to log pointers to the inputs and outputs of several microphone-related API calls. We then use a live binary analysis tool called `x64dbg` [4] to read the contents of the buffers out to a log file. We utilize an anti-anti-debugging library called Scylla-Hide [3], which hides the fact that an app is being debugged to prevent the app from crashing.

Chromium

Chromium acts as an intermediate layer between the operating system and the browser based VCAs. To verify whether web-based VCAs access the microphone while muted, we inject our own logging code in the source of Chromium. We instrument the following three browser functions in Chromium, which are responsible for transporting audio from the operating system to the VCA[2]. First, the browser initiates audio-related `read_data` function, which retrieves the raw microphone data from the operating system and stores it in a raw audio buffer. Then it calls `encode` and `send_stream` functions, which transforms the raw audio into an encoded stream and transfers the encoded audio stream to the web-based VCAs.

macOS

An audio subsystem manages microphone data created by Apple via `AVFAudio` or the `AVAudioEngine` interfaces [10]. These interfaces have the same purpose and interact with the audio hardware in userland. VCAs make a system call to `mach_msg_trap` within either an audio interface thread managed by Apple and retrieve

2 Appendix B includes more details about the functions inside Chromium.