Yucheng Yang†, Jack West†, George K. Thiruvathukal, Neil Klingensmith, and Kassem Fawaz*

Are You Really Muted?: A Privacy Analysis of Mute Buttons in Video Conferencing Apps

Abstract: Video conferencing apps (VCAs) make it possible for previously private spaces — bedrooms, living rooms, and kitchens — into semi-public extensions of the office. For the most part, users have accepted these apps in their personal space without much thought about the permission models that govern the use of their private data during meetings. While access to a device's video camera is carefully controlled, little has been done to ensure the same level of privacy for accessing the microphone. In this work, we ask the question: *what happens to the microphone data when a user clicks the mute button in a VCA?* We first conduct a user study to analyze users' understanding of the permission model of the mute button. Then, using runtime binary analysis tools, we trace raw audio flow in many popular VCAs as it traverses the app from the audio driver to the network. We find fragmented policies for dealing with microphone data among VCAs — some continuously monitor the microphone input during mute, and others do so periodically. One app transmits statistics of the audio to its telemetry servers while the app is muted. Using network traffic that we intercept en route to the telemetry server, we implement a proof-of-concept background activity classifier and demonstrate the feasibility of inferring the ongoing background activity during a meeting — cooking, cleaning, typing, etc. We achieved 81.9% macro accuracy on identifying six common background activities using intercepted outgoing telemetry packets when a user is muted.

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

Yucheng Yang†: University of Wisconsin-Madison, E-mail: yang552@wisc.edu
Jack West†: Loyola University Chicago, E-mail: jwest1@luc.edu
George K. Thiruvathukal: Loyola University Chicago, E-mail: gkt@cs.luc.edu
Neil Klingensmith: Loyola University Chicago, E-mail: neil@cs.luc.edu
***Corresponding Author: Kassem Fawaz:** University of Wisconsin-Madison, E-mail: kfawaz@wisc.edu

1 Introduction

As the de facto alternative for in-person meetings during the COVID-19 pandemic, the demand for online video conferencing for professional and personal use increased significantly. Video Conference Apps (VCAs), such as Zoom, Slack, Teams, and Webex, became available on all modern devices and operating systems. To support their functionality, these VCAs require access to the device's microphone and camera. Operating systems (OSes) provide the users with permission controls that allow the app to access the microphone and camera. Once granted, the app has access to both hardware resources until the user revokes the permission.

In addition to OS-based controls, VCAs provide their users with two privacy control mechanisms during a call: turning off the camera and muting the microphone. In most OSes, such as Windows and macOS, turning off the camera from the app engages an OS-level control which prevents the app from accessing the camera. A visible hardware indicator (e.g., a light near the camera) informs the user whether an app is accessing their camera. On the other hand, the implementation of the mute button is app-dependent and rarely has a visible hardware indicator. OSes do not expose an easily accessible microphone switch to the apps without going through many steps (e.g., via a control panel).

Apart from smart speakers, which pose tangible privacy threats, the mute button has received little attention in the context of VCAs. Previous research investigates users' privacy attitudes towards VCAs and alludes to the mute button as a privacy control tool available to the users during a virtual meeting [17, 25]. However, the mute button's privacy implications during the interactions between the user and VCAs have not been adequately addressed.

This paper investigates the privacy issues associated with the mute button in VCAs, focusing on whether a mismatch exists between the user's perception of the mute button and its actual behavior. We follow a two-pronged strategy to guide our investigation. First, we

† Both authors contributed equally to this work.

BY-NC-ND

design a user study to uncover what the users think the mute button does (i.e., their understanding) and what they believe it should do (i.e., their expectations). Second, we compare the user study findings against an empirical investigation of the actual behavior of the mute button across a range of VCAs and operating systems.

We conducted a user study with 223 participants recruited from Prolific. Our user study revealed that the participants perceive the mute button of VCAs as a privacy control, preventing other meeting participants from overhearing them. We observed a dichotomy in the understanding of the mute button: participants were split about whether a VCA accesses the microphone after they click the mute button. However, most of them indicated that the VCA should access the microphone only when unmuted.

Based on the findings from the user study, we empirically characterized the conditions in which the VCA actively queries the microphone in different operating systems. This task was challenging because OSes only log microphone accesses for each app; they do not provide fine-grained statistics about microphone queries. We addressed this challenge by instrumenting Windows, macOS, Linux, and the Chromium browser to track the fine-grained microphone queries by popular VCAs. We conducted a set of experiments on each VCA-OS combination to monitor the API accesses of each VCA under different conditions. We discovered that all of the apps in our study could actively query (i.e., retrieve raw audio) the microphone when the user is muted. Interestingly, in both Windows and macOS, we found that Cisco Webex queries the microphone regardless of the status of the mute button.

We followed our instrumentation efforts with an analysis of Webex, a popular VCA for the enterprise setting. We analyzed how it processes the queried microphone data to determine whether any audio-derived data leaves the device. This analysis also proved challenging as the VCAs, such as Webex, encrypt outgoing traffic. Further, tracking the data flow within apps is not straightforward because they employ proprietary and obfuscated libraries. To facilitate tracking of audio data, we performed a backward search from the encrypted network traffic to locate the inputs to the encryption functions. This search allowed us to decrypt the contents of the network packets sent by Webex to its servers. We discovered that Webex sent periodic packets containing audio-derived telemetry data to its servers, even when the microphone was muted. Although these packets are transmitted at a low rate (once per minute), their audio-derived values correlate with the volume levels of background activities.

To verify our hypothesis, we present a classifier to fingerprint background activities from these telemetry values. Training this classifier was also challenging. Without access to the proprietary algorithm that generates the audio-derived data, it is not feasible to use existing audio datasets to create training data for the classifier. Furthermore, the training data has to represent real-world situations, including realistic noise types and varying volume levels. We address this challenge by collecting Webex-based telemetry data corresponding to more than 200 hours of background activities. Our evaluation of the classifier with over-the-air data shows that telemetry data from Webex can conclusively fingerprint a set of popular user activities, such as music, chatting, and vacuum cleaning. We demonstrate that even with user data that is compressed and transmitted on a minute-by-minute basis, some activities have unique patterns that are discernable in Webex's telemetry data.

Our key contributions are as follows.

- User Study: We conduct a user study with 223 VCA participants to assess their understanding and expectations regarding the mute button (Sec. 3).
- Audio Access Tracing: We analyze VCAs' fine-grained access to the microphone; we found that most VCAs have access to audio-derived data even when the user is muted (Sec. 4).
- Webex-based Case Study We conduct a thorough system-level study of the Webex Windows client. We discover that, in contradiction to its claims in the privacy policy, Webex sends periodic audio-derived data to its servers (Sec. 5).
- Background Activity Detection: We present a design for a machine learning model the infers background activities from Webex's audio-derived data (Sec. 5.3).
- Mitigation Strategies: We distill our findings in the form of mitigation strategies that provide users with better control over the mute button (Sec. 6).

2 Related Work

In the following, we discuss the recent results about the privacy of VCAs. While there is existing research studying possible exfiltration of audio and video data from mobile apps [37], we focus on the research specific to VCAs. We also include related work about mute but-

tons in the context of smart speakers. Finally, we discuss the work surrounding background activity recognition, which is relevant for our analysis in Sec. 5.3.

Privacy Issues in VCAs

The security and privacy of video conferencing platforms has been studied since the early 2010s. In 2013, Kilpi et al. examined privacy and security issues in future (at that time) videoconferencing technologies [25]. They discuss the mute button as a necessary privacy control for the users. More recently, Emami-Naeini performed an online user study to understand the user concerns with VCAs [17]. They found that users are concerned about the security and privacy properties of VCAs. They also found that individuals consider the mute button as a privacy control: they perceive privacy violations from forgetting to press the mute button.

During the pandemic, more people were exposed to privacy and security risks caused by VCAs [34, 38]. In 2019, Zoom xed a camera leakage vulnerability caused by its casual use of a local web server [35]. Meanwhile, real-time background blurring for VCAs is widely adopted to protect user's privacy in an o ce or home environment [36, 45]. However, VCAs may leak a user's video privacy in many ways. Kagan et al. [23] demonstrated that collage images of video conference meetings posted on public websites may leak sensitive information such as users' names, ages and genders. Altschaffel et al. [8] showed that tra c patterns of encrypted metadata and multimedia data exchanged during VCA meetings, can be used to identify increased activity in front of camera or even identify users. There are also concerns with the information that VCAs collect about their users. For example, Consumer Reports identi ed privacy concerns with the data collection practices of popular VCAs, such as Zoom, Google Meet, Microsoft Teams, and Cisco Webex [40]. These concerns centered around the purposes of collecting metadata from the meetings.

In this paper, we follow-up on these previously-reported vulnerabilities and privacy studies. In particular, we study the users' understanding and expectation of the mute button, and whether they match the VCAs' behavior. We focus on the interaction between the VCA and the user's microphone when the user presses the mute button, as opposed to previous research that studies the mute button in the context of protecting the user's privacy from other meeting participants.

Mute Button in Voice Assistants

Researchers have also considered the privacy issues from always-listening smart home devices[6, 26]. Smart home devices continuously process the raw audio to detect a trigger word or phrase. As such, the privacy threats arise from these devices accidentally or maliciously recording the user's background activities [7]. Researchers have rst discussed the e cacy of the physical mute button as a privacy control to mitigate these threats. The mute button was found to be inconvenient and su ering from user trust issues [14, 26]. Follow-up works proposed other privacy controls, such as ultrasound jamming [14, 15, 41], cutting the power [14], and employing interpersonal communication cues [32].

Contrary to the smart device case, VCA users widely utilize the mute button to prevent others from listening to their background activities (Sec. 3). Users trust that other meeting participants cannot hear them after applying the mute button. However, the behavior of the VCA, after applying the mute button, is less understood. In this paper, we characterize the operation of the mute button from the perspective of the interaction between the user and the VCA.

Activity Fingerprinting

Finally, we discuss research about ngerprinting activities from audio-derived data. User activities and contextual information, including walking, driving, and riding, can be inferred from ambient sound. Lu et al. [29] presents an audio event classi er that identify user's current activities utilizing the microphone input of mobile phones. Not only the ambient sound, but encrypted audio tra c can be used to infer user's private information. Previous studies proved that encrypted IoT tra c might leak private information of their environment, including device status and user activities. Tra c analysis of the video streams from home security cameras enables monitoring daily activity patterns [11, 16, 28]. Li et al. [27] further demonstrated the possibility of detecting ne-grained activities, including dressing, moving, and eating, from encrypted home security camera traf- c. They selected features such as tra c packet size and length distribution. Similar to encrypted tra c analysis, Schuster et al.[39] performed an encrypted Video Stream Identi cation by analyzing bitrate burst and time interval of video streaming tra c. They utilized the segment transmission mechanism of MPEG-DASH and successfully identi ed Net ix video titles using a trained classi er.

Kennedy et al. and Wang et al. [24, 43] demonstrated that an attacker can infer which voice commands a user says to a smart speaker, by eavesdropping and analyzing outgoing encrypted tra c from smart speakers to a cloud server. Wang et al. [43] further manifested the incoming tra c from the server also leak voice commands information. Moreover, Bae et al. [5] presented a video streaming service identi cation attack by monitoring video downstreaming tra c through LTE networks with high accuracy.

These research works demonstrate that data derived from audio streams can be used to ngerprint their content and is therefore relevant to our discussion in Sec. 5.3 about inferring the background activities while the user is muted.

3 User Study

Our rst objective is to study the user perceptions of the mute button along with their understanding of its functionality. Towards that end, we conduct an online user study with 230 VCA users. Our study aims to answer two questions about VCA users: (1) When do they think the VCA accesses their microphone? and (2) When do they think the VCA should access their microphone?. Answering these questions allows us to characterize the user's understanding and expectations of the mute button, respectively. In the following, we describe the design of the user study, the recruitment, and the ndings.

3.1 Study Design

We designed a Qualtrics survey[1] to help answer our research questions. We used partial disclosure to hide the fact that the study was about the privacy implications of the mute button. The description of the survey and its title focus on capturing the users' general experience with VCAs during the pandemic. The survey has four major sections; the rst section collects optional demographic information. The second section collects information about the preferred VCA and frequency of usage.

The third section asks the respondents about their experience with the mute button. We adapt the questions from Lau et al. [26], which studies the mute button in smart speakers. In particular, we probe the users about their usage of the mute button, their reasons, and their understanding of its functionality using questions in Table 1. This section contains three open-ended questions and two multiple-choice questions.

The last section adopts a re ned version of Internet Users' Information Privacy Concerns (IUIPC-8) from Groÿ [21] to measure the participants' privacy concern. This survey section contains the rst mention of privacy, after the respondents have answered the questions related to the mute button. Finally, the survey includes two attention checker questions and was exempted by the IRB at our institution.

Participant Recruitment and Demographics

We recruited participants from the Proli c data collection platform. We employed Proli c's prescreening criteria to enforce gender balance and to forward the survey to only those who have worked from home during the COVID-19 pandemic with 90% approval rate in previous studies. Before conducting the survey, we conducted a pilot study with 15 users to calibrate the payment and ensure that the study design is clear. Through Proli c, we were able to recruit 299 participants, where we kept 223 responses from participants who passed the attention checkers. The median completion time was 8 minutes, and we paid each participant $1.5; the median hourly rate was $11.

Among our participants, 96.8% are between 18 to 44 years old, 63.2% of them work in sales, service, management and professional industry, and 82.5% achieved at least a college degree. During COVID-19, 54.7% of our participants answered that they have used video conferencing apps more than once a day and 40% of them used once a day or once every few days. The most popular video-conferencing app among the participants is Zoom, and the other popular apps include Microsoft Teams, Google Meet and Cisco Webex.

We map the responses to the IUIPC-8 question to a score based on seven-point Likert scale, representing participant's privacy attitudes. The average scores is 2.02 for all participants, implying that most participants are privacy-conscious in our study. The value of Cronbach Alpha Index is 0.7915 for privacy attitudes responses from 223 participants, which indicates a good internal consistency and reliability of these responses.

1 The full survey can be found here: https://osf.io/szd4x/

Open Ended Questions	Multiple Choice Questions	Multiple Choice Answers for Q4 and Q5
Q1. Why do you use the mute button?	Q4. For your most frequently used video meeting app, when do you think it has access to your microphone?	S1. When the app is not running.
Q2. What activities do you perform or take place in your background when you are muted?	Q5. For your most frequently used video meeting app, when should it have access to your microphone?	S2. You start the app but are not in a meeting.
Q3. Please describe what does the app do when you press the mute button.		S3. You're in a meeting but you apply the mute button in the app.
		S4. You're in a meeting and you are unmuted.
		S5. You leave the meeting while the app is still running.

Table 1. The main questions used in the user study. Q1-Q3 are open-ended questions with answers coded by researchers. Q4 and Q5 are multiple choice questions where the participant selects one or more statements from S1-S5 in response. The full list of questions is available in the Appendix.

Fig. 1. The distribution of the codes about reasons users reported for using the mute button as extracted from answers to Q1.

Fig. 2. The distribution of the codes about the background activities as extracted from answers to Q2.

Fig. 3. The distribution of the codes about the users' understanding of the mute button operation from answers to Q3.

3.2 Findings

We report the key findings from our user study, through analyzing the participants' responses. We coded the responses to the open-ended questions (Q1, Q2, and Q3) following this procedure. For each question, two authors independently coded the responses, after which they generated a consolidated codebook describing the responses. For Q1, we settled on five codes about the reasons for which participants use the mute button. For Q2, the codebook consists of twelve codes describing the background activities. The codebook for Q3 contains nine codes representing the participants' description of the mute button operation. Then, each coder independently coded the first 30 responses for each question; the resulting Cohen's kappa is 0.85 for Q1, 0.90 for Q2, and 0.82 for Q3, indicating strong agreement [30]. The coders split and coded the rest of the responses. See detailed codebooks of the open-ended questions in Appendix E.

Usage Patterns: We start by analyzing the responses to Q1, where 214 participants out of 223 indicated that they have used the mute button before. The responses for Q1, as shown in Fig. 1, reveal two main reasons why users employ the mute button: (1) hide background activities and (2) avoid interrupting or disturbing others on the call. It is interesting that the participants regard the mute button as a privacy control measure to prevent others from hearing them. For example, P19 mentioned the reason for using the mute button is: So that people won't listen to private activities or conversations.

The responses for Q2 indicate an array of background activities the participants perform while muted, as indicated in Fig. 2. Participants mentioned more than one activity in their responses; For example, P166 mentioned: Talking, loud video watching, cat activity (meows, occasional falling and crashing of items), cleaning (including vacuuming). The most prevalent activity was related to preparing food, cooking, snacking, or eating. Other frequent activities include chatting, watching TV, cleaning, typing, or watching online videos. We elaborate more on these background activities in Sec. 5.3.

Fig. 4. The distribution of responses to Q4. The statements S1-S5 are defined in Table 1.

Fig. 5. The distribution of responses to Q5. The statements S1-S5 are defined in Table 1.

Understanding of the Mute Button: As indicated earlier, we asked the participants two questions (Q3 and Q4) to gauge their understanding of the mute button. To gain initial insights into the participants' understanding of the mute button, we study the coded responses to Q3, as evident from Fig. 3. The most frequent response was that by using the mute button, the app prevents others in the call from hearing the user. For example, P16 indicated that: It doesn't produce my audio on the other participant's platform or computer. Moreover, other participants focused on the interface change when the mute button is pressed, as in the case of P119: It shows me the mic with a line crossing it signalling it is not working. . Meanwhile, 59 participants mention that the mute button disables the microphone. For example, P161 mentions: When I press the mute button, my microphone is muted and disabled on the app from picking up any sound waves from where I am.

For Q4, we provide five situations, S1-S5, in our user study as shown in Table 1. The responses to Q4 indicate that the participants exhibit a diverse understanding of the operation of the mute button, as shown in Fig. 4. Out of the 223 responses, 69 participants selected only S4 as a response to Q4. These participants think the app only accesses the microphone when they are in the meeting and the mute button is not pressed.

Further, we found that the participants were split in their selection of S3 as a response to Q4. Nearly half of the participants (111) did not select S3, indicating that the app does not access the microphone when the mute button is pressed. The other half indicated that the app accesses the microphone, even when muted. Interestingly, we observe that 49 participants selected S2, S3, S4, S5 when responding to Q4, indicating that the app accesses the microphone as long as it is running. Also, we observe that 36 participants selected S3 and S4, indicating that the app accesses the microphone as long as the user is in a meeting. In all the cases above, we found no correlation between the responses and the IUIPC-8 privacy attitude scores.

Expectations of the Mute Button: Finally, we analyze the responses to Q5, about when do the participants think the VCAs should access the microphone. The responses reveal that the participants have clear expectations about the operation of the mute button, as indicated in Fig. 5. Among the 223 responses, 173 participants selected only S4 as a response to Q5. These participants indicated that the app should only access the microphone when the meeting is running and the user is unmuted. Interestingly, 27 respondents selected both S3 and S4 as a response to Q5.

In conclusion, the results from the user study suggest that the user's understanding of the mute button does not match their expectations of its behavior. In the rest of this paper, we study the actual behavior of the mute button and analyze whether it matches user understanding and expectations.

4 Analysis of Mute Button

Following the results from our user study, we investigate whether the actual behavior of VCAs matches user expectation by focusing on desktop environments. Our objectives are to determine: (1) if VCAs actively access the microphone when muted and (2) what kind of indicators (if any) they give users that the microphone is being accessed.

4.1 Overview of VCAs and Platforms

There are two broad categories of runtime environments in which VCAs execute: native apps that run directly in the operating system and web apps hosted by a web

App	Windows	Linux	macOS
Zoom (Enterprise)	X	X	X
Slack	X	X	X
MS Teams/Skype	X	5	X
Google Meet			
Cisco Webex	X		X
BlueJeans	X		X
WhereBy			X
GoToMeeting	X		X
Jitsi Meet	X		X
Discord		X	

Table 2. A summary of the VCAs we studied. X: native app : web-based app5 : No implementation.

browser. Each has a di erent permission model for accessing the microphone. Most of the VCAs we study in this work have a native app implementation for the major operating systems (macOS and Windows) and a web app used on unsupported platforms (Linux and others). The VCAs that we studied (listed in Table 2) exhibit a consistent look and feel across platforms. Their implementation, however, on each platform is di erent, due to syscall interfaces and display APIs. Zoom on Windows, for example, is a self-contained Windows-speci c software package. Zoom on macOS has a similar user interface to its Windows counterpart, but the underlying code base appears to be di erent.

Native apps can collect data from the microphone with few restrictions. Web apps implemented in JavaScript request access to the microphone through a web browser, which generally has more restrictive policies for data collection and more tools that allow the user to control the app's access to hardware.

Browser Based Apps

Browser-based VCAs rely on their host browser to mediate their interactions with the operating system and the hardware. The browser-based VCAs that we studied are implemented entirely in JavaScript, and they use a special-purpose API called WebRTC [19] for driver interactions including microphone accesses that are typically not available to web apps. WebRTC is a native interface written in C++ and C, acting as a driver for the hardware within the browser that can call the operating system to access the microphone. Information transferred by WebRTC is subject to controls and policies of the browser. Web-based VCAs are sandboxed inside the browser and do not circumvent WebRTC.

There are two ways a user can mute a web-based VCA: (1) using a browser-level mute button or (2) using a WebRTC software mute signal from the app. Both techniques are more trustworthy than app-controlled mute because they are implemented and enforced by the browser, not the app.

The browser-level mute button completely disables microphone access to the VCA, as if the microphone is not active within the system. Web-based VCAs also implement an app-level mute button, which has similar functionality to the browser-level mute: it enables a software mute inside of WebRTC, disabling all audio transfers from the microphone. Users must trust the web-based VCA to use the software mute functionality rather than some internal mute button implementation. We found that all of the studied apps use the WebRTC mute functionality correctly. Furthermore, it is straightforward to verify that web-based VCAs correctly use the software mute functionality through source code audits and the WebRTC debugger built into Chromium.

Native Video Conferencing Apps

Native VCAs can directly call the operating system to retrieve audio data from the microphone. Most of them abide by the operating system (OS) rules to access the microphone data, with some exceptions. The OS imposes fewer restrictions on native apps than the browser runtime environment imposes on web apps.

All operating systems utilize a permissions-based access system to retrieve data from the microphone. In most cases, apps must have explicit permission to access hardware resources such as the microphone. Each app follows three steps to con gure and use the microphone: (1) user approval, (2) driver initialization, and (3) audio data retrieval. Windows and macOS require the user to explicitly provide permission for each app, which the app retains inde nitely while it runs (unless the user revokes the permission).

Once the user approves the access for the app, the app must create an interface to the audio drivers. Some OSes, like Windows, o er users a visual cue that indicates when the app is using the microphone. But unlike the WebRTC browser runtime, none of the major operating systems we are aware of support enforce a software mute. This lack of an OS-mediated software mute means each native app must implement its own internal mute functionality. Even when a software mute is active, apps can still access the microphone while the user is muted.

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 bu er of microphone data arrives from PulseAudio. We write the audio bu er's address in the process's memory space to a log le. We then trace the bu er addresses from the log using IDA Pro. The contents of the bu er are the raw audio bytes from the microphone. DynamoRIO oversees the process's execution by loading and running modi ed 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 ags about microphone activity.For ne-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 o cial 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 les. 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 bu ers out to a log le. 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 bu er. 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_trapwithin either an audio interface thread managed by Apple and retrieve

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

raw audio bytes from the microphone. All of the VCAs we studied connect to the microphone using either of these interfaces and make the same system calls when reading bytes from the microphone.

To monitor VCAs' microphone accesses we use a XCode tool called Instruments [9] and the standard Unix networking tool tcpdump. Instruments logs all system calls and their arguments to a user interface in the Apple system log. tcpdump records network tra c while any of the VCAs are running. We attach Instruments to a live VCA and perform a tcpdump on the networking interface to extract and monitor the data ow from microphone to the VCA. We then observe the results from Instruments to correlate behavior patterns with with Windows evaluation. VCAs in macOS behave similarly to their Windows implementations.

4.3 Findings

To understand how VCAs consume microphone data, we conducted experiments on each app-OS combo shown in Table 2. We installed all VCAs and registered two accounts for each app on each of the four operating systems. The app-OS combinations that are only accessible in a browser are tested in Linux on Chromium. We initiated the meeting app for each meeting experiment and used the techniques explained above to trace microphone data from OS to VCA under two conditions: mute button toggled on and mute button toggled o . Most platforms we studied display a visual indicator to alert the user that an app is accessing the microphone[3]. We found three broad policies that VCAs follow to read data from the microphone while muted:

1. Continuously sampling audio from the microphone: apps stream data from the microphone in the same way as they would if they were not muted. Webex is the only VCA that continuously samples the microphone while the user is muted. In this mode, the microphone status indicator from an operating system remains continuously illuminated.
2. Audio data stream is accessible but not accessed: apps have permissions to sample the microphone and read data; but instead of reading raw bytes they only check the microphone's status ags: silent, data discontinuity , and timestamp error . We assume that the VCAs, like Zoom, are primarily interested in the silent ag to tell if a user is talking while the software mute is active. In this mode, apps do not read a continuous real-time stream of data in the same way as they would while unmuted. Most Windows and macOS native apps[4] can check if a users is talking even while muted but do not continuously sample audio in the same way as they would while unmuted. In this mode, the microphone status indicator in Windows and macOS remains continuously illuminated, reporting that the app has access to the microphone. We found that applications in this state do not show any evidence of raw audio data being accessed through the API.
3. Software mute: apps instruct the microphone driver to completely cut o microphone data. All of the web-based apps we studied used the browser's software mute feature. In this mode, the microphone status indicator in the browser goes away when the app is muted, indicating that the app is not accessing the microphone.

The notable exceptions to these trends are the Microsoft VCAs (Teams and Skype) and Cisco Webex. Microsoft VCAs are much more di cult to trace because they do not use the standard Windows userland API. Instead, they directly make calls to the operating system. Since the Windows syscall interface is undocumented, we could not determine how Teams and Skype use microphone data when muted. More interestingly, we observe that Cisco Webex unlike the rest of the Windows native VCAs continuously accesses the microphone while muted. Using x64dbg, we were able to trace Webex's copied audio bu er until that bu er reaches the stack. We discovered that while the app was muted, Webex's audio bu er contains raw audio from the microphone. In the next section, we focus our data ow analysis on Cisco Webex in Windows because of its popularity [5] in the enterprise setting and, more importantly, its unusual behavior.

Recall that our user study reveals two main observations: participants are split whether the VCAs access their microphone while muted, and expect them to access the microphone only when they are unmuted. Our results from this section indicate that the participants

3 Some Linux distributions do not provide any visual indication that the microphone is in use.

4 Except Skype and Teams, which we cannot observe because they do not use the conventional Windows API.

5 The monthly statistics from Cisco Webex include 600 million participants and 6 billion calls [42].

are largely unaware of the operation of the VCAs. More importantly, the behavior of these apps violates user expectations. This mismatch between user expectations and app behavior highlights privacy issues with the design of the mute button.

5 Webex Case Study

Based on our findings from the previous section, we perform an in-depth analysis of the microphone access pattern in Cisco Webex[6]. We focus on Windows 10 as it is the most widely used operating system at home and in enterprise[7]. As Webex continuously samples the user's microphone (when muted), we need to study whether audio-derived data leaves the local device.

Determining whether audio-derived data from a VCA is leaving on the network port is not a straightforward task because the network flow from VCA to a server is encrypted. Raw dumps of the network traffic from Wireshark are not informative about precisely what the network traffic carries to the VCA's server. And we know that VCAs send and receive network packets that do not contain any audio or video data, so counting network packets cannot give us an indication of whether audio-derived data is leaving a device while the VCA is muted. Instead of directly logging network packets, we need to track how audio data is processed within a VCA.

5.1 Methodology for Traffic Interception

Fig. 6 depicts the flow of data from microphone to network in native Windows apps. Understanding how a particular VCA handles data from the microphone requires tracking the data as it traverses the chain of processing shown in Fig. 6. Most of the data processing in the VCAs we study is handled by proprietary DLLs [8]. Tracking data through function calls from the main VCA process to a DLL is unreliable because runtime binary analysis tools like IDA Pro [2] and x64dbg [4] often cause the app to crash when they single-step through function calls to a DLL. And since each VCA uses a different set of external DLLs, we could not establish a single workflow to analyze all VCAs. Existing tools such as TaintDroid [18] are able to establish the data flow within an application in older Android versions. However, in native applications designed for Windows and macOS, flow tracing is difficult and sometimes impossible.

It is easy to see when an app accesses the hardware (networking and microphone) by monitoring Windows API calls (see Sec. 4.2), but we are not aware of any tool that can automatically follow data through an entire Windows app. Tracking microphone data after each instruction is not straightforward. Such data initially exists inside of dynamically-allocated memory buffers. Upon each access, this data might move to a new buffer after undergoing a transformation, such as encryption, compression, or encoding. Further, the new buffers may originate from different allocator functions to be stored in the main process's memory image or in an external DLL's memory image. Race conditions among the threads in Webex compound the difficulty of tracing: all memory accesses at a specific address of the stack require stoppages, logging, and memory analysis, all of which take time to perform.

However, we do not necessarily need to show a linkage between every successive subroutine that handles microphone data in a VCA to demonstrate that audio-derived data leaves on the network. We can already dynamically trace the audio into the app. We need to show that data from that buffer leaves our machine and is transmitted to a Webex server.

To design such a system, we first map all of the outgoing traffic from Webex. The most efficient way of doing so is to use the Microsoft Network Monitor (MNM). We observe Webex's network traffic using the MNM while the app is muted and unmuted, and we notice a set of packets that are periodically going to a user metrics Cisco server. Now that we have our packets, we need to ensure that the audio buffer within Webex is accessed in the muted state.

Binary tracing on the audio buffer's read/write access using x64dbg always ends up in stack space which thwarts our further tracing. However, while following the bytes and logging API calls (from Sec. 4), we noticed that Webex calls encoding libraries which access in some time correlation with the audio bytes. We then trace API calls to encryption methods to verify what is happening using the API Monitor. We capture all input arguments and output buffers as a log file from these calls while the user was muted during a Webex meeting. The log contains timestamps, input parameters to the API call, and the resulting output buffer. With the results

6 We used Webex client version 41.12.3.11 through our study.

7 90% of respondents to our user study used Windows.

8 Dynamically Linked Libraries (DLLs) are the Windows implementation of shared libraries.

Fig. 6. Data ow of audio bytes within a Windows 10 VCA. This pipeline is generalizable across the Windows platform. Our system attaches to the bolded modules.

Fig. 7. Correlation between audio gain reported by Webex and input audio signal power level (in dbA) when noise removal mode is enabled. Although we cannot observe the raw audio while muted, the statistics reported by Webex leak information about the user's background noise.

of the function logged, we compare the encrypted bu er to network tra c leaving the machine and notice a one-to-one match between the encrypted bytes (from Wireshark) and the data sections of network packets from Webex. Consequently, we link the data regions of outgoing user metrics packets to our post-encrypted output bu ers. Upon observing the input, we notice that the input arguments in these cases are in plain-text where detailed data is compressed using base64 encoding. Decoding the input arguments revealed the packet content to be a JSON structure[9], which contains audio-derived data and other data elements.

5.2 Findings for Tra c Interception

The data we capture from the API hook is a JSON array with unencrypted and unobfuscated attribute names such as: audioMaxGain, audioMeanGain, audioMinGain, and many others. These JSON arrays are transmitted by Webex once per minute to https://tsa3.webex.com, a telemetry server, while the user is muted. The names of these attributes suggest that the JSON array contains audio-derived statistics, most probably connected to the automatic gain control employed by Webex. Our aim is to further analyze the attributes to understand the relationship between the recorded audio levels and these attributes values when the microphone is muted.

Webex has two microphone modes: music mode and noise removal mode (the default mode). As the name suggests, noise removal mode refers to Webex removing background noise in real-time while the user is speaking. Music mode, on the other hand, transmits audio as the microphone hears it. We perform a small-scale experiment to study whether the audio attributes from Webex network tra c are correlated with the input audio for both microphone modes. We play episodes of the U.S. TV shows Friends and The O ce into a microphone during a Webex meeting while muted. To isolate environmental factors, we feed the audio from the TV shows directly into the Webex meeting through a virtual microphone interface. We repeated each experiment for both microphone modes.

We partition each audio le (corresponding to an episode) into a set of one-minute windows. We then compute the maximum and average magnitudes for each window to report their correlation with audioMinGain and audioMeanGain Note that the audioMinGain value would correspond to the maximum observed audio level because it requires less gain control. Further, the minimum and mean values depend the most on the input audio. On the other hand, the maximum depends more on the input device and the amount of silent moments, which are random in each episode.

Fig. 7 depicts the correlation between the estimated power levels and measured gain values for noise removal mode. As evident from the gure, the measured and estimated values exhibit high correlation; the correlation with the mean gain is higher as it is a more robust metric to window shifts. Note that we do not have access to the source code when computing the gain values,

9 An example of such a structure is here: https://osf.io/szd4x/

so a perfect correlation is unlikely. This correlation is slightly lower than that of the music mode (Fig. 11 in Appendix D), implying that the noise removal changes the input audio. Still, the measured audioMinGain and audioMeanGain are representative of the audio levels.

5.3 Classification of Background Activities

We established that Webex accesses the microphone while muted and sends audio statistics to their servers. Further, this data is highly correlated with the energy level received at the microphone, and appears to be indicative of the activity happening in the background. The logical question that follows is: is there a potential of learning the user background activities from audio statistics sent to Webex's servers? In the following, we describe how these statistics can fingerprint the user's background activities, when they are muted.

We analyze the inference of information from user's Webex telemetry traffic while being muted. For each one-minute window, this information contains three values that change relatively : mean, min, and max audio gains. An entity with access to this information, such as Webex's cloud service or any adversary able to view this traffic in transit, can perform this analysis to infer what activity is occurring in the user's environment.

5.3.1 Data Collection

We focus on the background activities from our user study of Sec. 3. In Fig. 2, we highlight twelve activities that happen in the user's background. Out of these activities, we do not consider: (1) silent and physical activities as they do not result in gain changes, (2) bathroom as it is unlikely that the user's microphone will pick up bathroom noises, (3) street noise as it does not represent a private activity, and (4) diverse noise such as TV shows which may contain all of the classes in a single 30-minute instance. As such, our objective is to identify whether the gain values can fingerprint six types of activities: (1) music playing, (2) cooking or eating, (3) people talking, (4) animal sounds (especially dog barking), (5) keyboard typing, and (6) cleaning.

To simulate the real-world environment with specific background activities, we choose multi-hour long ASMR YouTube videos that consist of single background activity. Each video is different such that the videos are produced by different people (YouTube users) doing the same task. The purpose of selecting the videos in such a way is to minimize the effects from the recording environment. We play each video over the air through a Webex meeting, while muting the microphone, and log the extracted gain values.

Fig. 8. Clusters of audio statistics data color coded by background activity type. Clusters are visually separable.

Our data collection consists of two Windows 10 machines. The first machine plays the videos using its speaker and hosts the meeting for the other machine. The other machine runs a Webex meeting client (without any other app running). One machine is equipped with a Logitech QuickCam Pro 9000 while the other uses a Logitech C920S Pro HD 1080p webcam for microphone input. Both machines then join the same meeting room and collect data simultaneously; on both, we mute the microphone, turn off the camera, and keep the default microphone settings.

We place the machines in a 125 ft × 75 ft × 105 ft room. We adjust the distance from the speaker to the two microphones and generate multiple datasets based on the varying distances. Webex only allows for meetings to last for 24 hours. For each Webex meeting, we can extract around 1440 data points, stamped with the corresponding label. Each data point corresponds to three features: audioMaxGain, audioMeanGain, and audioMinGain, representing three user metrics values from one minute of audio. In summary, we performed data collection over the course of two months, yielding over 180 hours of data points.

We visualize the distribution of the six background activities in Fig. 8. This figure shows that it is feasible to fingerprint background activities by analyzing the extracted gain values from Webex. Each activity exhibits