2017 25th Euromicro International Conference on Parallel, Distributed and Network-Based Processing

Sound Covert: A Fast and Silent Communication Channel through the Audio Buffer

Ofir Shwartz
Electrical Engineering Department
Technion - Israel Institute of Technology
ofirshw @ tx.technion.ac.il

Yitzhak Birk
Electrical Engineering Department
Technion - Israel Institute of Technology
birk @ ee.technion.ac.il

***Abstract*—We present Sound Covert, a novel technique using the machine's audio buffer to secretly (and silently) communicate between applications. Sound Covert can achieve more than 2.6Mbps, four orders of magnitude faster than previously published covert channels. Sound Covert can be used for bidirectional communication between malicious applications anywhere within a given VM (including across users and sand boxes), as well as for sending information from within a VM to an application running on the host OS of the same physical machine. It can also be used for sending information from a compromised external web page to a local malicious application through an innocent, unaltered web browser running in the target application's VM. Similarly, Sound Covert enables a malicious application in a VM that also runs a web server to send data to a malicious partner application running on any machine (or VM) that has a web browser.**

***Keywords*—Covert channel; Steganography; Security; Application security; Virtual machine**

I. INTRODUCTION

Computer security is a huge challenge. As the complexity of systems grows, so does the likelihood of finding security breaches in them. Furthermore, financial benefits for successful attacks have been on the rise, so the expertise attained by attackers is higher than ever.

Cloud providers use virtual machines (VMs) to control resource allocation, and to achieve isolation among different clients running on the same physical machine. Additionally, they use various measures to protect applications running in their cloud from external threats. Attackers continuously try to overcome the separation and protections, and by that harm or manipulate the application's integrity and the security of the data. In this paper we expose a major vulnerability.

A. *Covert Channels and Steganography*

Covert channels [17] have been proposed as a means for attackers to establish (unauthorized) communication among processes. A covert channel may serve to connect processes running in the same VM, across sand boxes, across VMs and even across physical machines, typically via a conduit that is not intended for communication.

A covert channel within a physical machine typically comprises two components: a sender application that affects software or hardware resources of the system, and a receiver application that monitors the resources' behavior. Both the sender and the receiver must know the intended technique, and each must expect the existence of the other. Typically, at least one of these applications has to be maliciously implanted or altered, either the sender for spying on its VM and sending data about it, or the receiver for secretly receiving foreign orders.

This work was supported in part by the Hasso Plattner Institute.

Steganography [37] is a means of hiding data inside an innocent stream of information, without being noticed. The sender manipulates a file, typically containing media, for implanting the secret data into it, and then sends it to the receiver. The receiver decodes the secret data from it. The manipulated file, if played to a human normally, should be indistinguishable from the original file.

In this paper we propose a new covert channel that uses a variant of audio steganography.

B. *Related Work*

A previous study [1] used the processor's temperature to represent information (E.g., high temperature='1' and low temperature='0'). The sender controls the processor's fan (and thus its temperature), while the receiver monitors the processor's temperature to get its message. Another [2] used cache hits/misses to send information: the receiver puts initial data in a specific block, and the sender either causes ('1') or not ('0') its evacuation. The receiver times the memory access and by that decodes the data.

CPU based covert channels include branch prediction biasing [6,7], memory access timing [8,9], and various shared CPU resources such as locks [10] and function units [11]. Shamir at al. [12] showed a light based covert channel, allowing an attacker outside a building to use an infra-red light source received by a paper-scanner located inside a building.

In [13,14,18], an acoustic network was proposed, forming a sound-based communication link between proximally located computers with speakers and microphones. This work suggested using ultrasonic frequencies that are beyond the human hearing range, yet can be played and detected by standard speakers and microphones. Later, [15] suggested using the speaker and microphone of mobile phones.

Steganography in audio files was proposed by using various techniques. These include 1) manipulating the least significant bit (LSb) of the audio samples, where the secret message itself is spread over the sample's LSbs [38]; 2) manipulating the parity bit of a selected region [40], such that correct and incorrect parity checks decodes the in-sample bit number to be looked at; 3) phase manipulation of the audio region [23], where $\pi/2$ or $-\pi/2$ phase is added to audio regions according to the data bits to be sent; 4) spread spectrum information injection [29], where the data bits are spread across the entire frequency spectrum; and 5) ultrasonic frequency information injection [39], whereby high frequency signals (beyond the human perception) are added to decode the data bits.

Many of these works suggested to compress and encrypt the message before being injected. All these require the resulting file to stay in a lossless format. In [22], it is shown that audio steganography is also possible in the presence of lossy audio compression such as MP3 [35].

C. *Our Contribution*

In this paper, we present Sound Covert, a novel covert channel technique that uses the machine's audio memory buffer and audio loopback feature [3,4] along with a variant of audio steganography to permit covert communication between applications running on the same physical machine. Sound Covert exploits several salient properties of the audio system in most modern computers: 1) access is permitted to all processes, 2) a loopback buffer permits any application to read the audio buffer's content, 3) the common volume control does not affect the buffer content, and 4) the vast majority of sound values do not produce any sound (too faint), so data can be encoded within this range, creating a "silent" channel.

Sound Covert is not a traditional covert channel by Lampson’s [17] definition, as this definition requires the manipulation of a resource that was not intended for communication, whereas the loopback feature was intended for transferring audio data. However, Sound Covert uses audio loopback to transfer general purpose data (not limited to audio), and in a covert manner (no sound is actually produced), similarly to the acoustic network [13,14,18].

Sound Covert is not a traditional audio steganography technique either. Audio steganography requires an original audio file for implanting data into it; this file is then sent to the recipient to decode the secret message out of it, and the modified audio file itself (only being a seemingly innocent transporter) does not need to be played. Sound Covert may create its own standalone audio stream, and it requires the audio stream to be played for the message to be transmitted. However, some variants of Sound Covert may use audio steganography as well (see Section V).

Operating at over 2.6Mb/s (we implemented a pair of sender-receiver applications that enable unauthorized file copy at this speed), Sound Covert permits communication between any pair of malicious applications running within the same VM, even in different sand boxes, between such applications running under the host operating system (an OS running outside any VM), and from any malicious application running on a given physical machine to a partner application running under that machine's host OS.

Additionally, we were able to use Sound Covert to fetch data from the web by a receiver application that does not access the network directly. We used a non-audible audio source stored within a compromised web page, and played by the (innocent, unaware) web browser on which the receiving application runs. Playing audio through a web browser does not require user approval, so a user does not notice any unusual activity. The receiver application does not access the network, so its activity goes unnoticed by the firewall and anti-malware. This enables various audio sources for Sound Covert, such as video streaming and web radio, where the server-side traces may be completely removed later. Similarly, an innocent, unaltered web server running in a given VM can be used with Sound Covert to enable the sending of information from a malicious application running in this VM to a partner application anywhere, provided that a web browser is running in the receiving application's VM.

Relative to prior art, Sound Covert is generally inferior in coverage within a physical machine because of its inability to permit direct inter-VM communication (unless a web server and browser are available). When specifically compared with the Acoustic Network, it moreover cannot cross physical machine boundaries without the aid of a web server and/or browser; however, Acoustic Network requires speakers and microphones, rendering it irrelevant to cloud computing farms and data centers, whereas Sound Covert only uses the built-in audio interface. Acoustic Network can be detected by audio measuring equipment, whereas Sound Covert does not produce any sound. Finally, Sound Covert may provide over 2.6Mbp/s, which is 10,000 times faster than previous works. Such a dramatically higher data rate may allow new (potentially malicious) use cases for covert channels.

The main contributions of this paper are:

- Introducing Sound Covert: a novel technique for a covert channel using the audio buffer and audio loopback;
- A covert channel that is four orders of magnitude faster than previously proposed ones, and with significant applicability to clouds;
- Introducing Silent Stream: analysis of the IEEE 754 32-bit floating-point audio representation, and suggestion of a method for inaudible data encoding;
- Providing experimental results for showing the applicability and data rate of Sound Covert.

The remainder of the paper is organized as follows. Section II presents our assumptions; Section III provides a detailed description of Sound Covert, including design alternatives for coding and operation; Section IV details implementations of Sound Covert for various settings; Section V discusses extensions of Sound Covert to more complex settings; Section VI details possible measures against Sound Covert, and Section VII offers concluding remarks.

II. REQUIREMENTS AND THE AUDIO INTERFACE

In this section we describe the requirements (system, computing environment) for using Sound Covert, and provide some background on the modern audio interface architecture.

A. *Requirements*

We assume a computer with an audio interface of any kind, e.g., a built-in audio interface exists in nearly every computer.

Two Trojan or compromised applications are assumed to have somehow been placed in the computer and are running; both only need normal user level privileges. One acts as the sender and the other as the receiver. The receiver must run on an operating system with audio-loopback support (such as Windows Vista and above [3] or Linux [4]). No special requirements for the operating system that runs the sender.

The sender may run in a VM, sandbox or host operating system; the receiver must run anywhere within the same VM or under the host operating system. In any case, the sender’s and receiver’s operating systems may be different from each other. (As will become clearer later, the receiver needs access either to the audio buffer of the sender's VM or to that of the host OS, hence the restrictions.)

B. *The Audio Interface Architecture*

Modern audio interfaces typically comprise an Intel High Definition Audio (HDA) compatible controller [16] that is connected to the memory controller via PCI Express or some other system interconnect, and audio codec chips (Fig. 1). The HDA controller contains DMA channels and required controllers, and the codec is merely a D/A and A/D conversion chip. The HDA controller is commonly implemented as part of the chipset, so it exists in nearly every computer.

The audio codec chip contains one or more D/A and A/D converters, each of which is connected directly to a physical audio port of the machine. A converter may get its data directly by DMA (or may send data by DMA), at a constant bitrate, forming an aggregated output audio stream.

Each DMA channel uses a memory buffer resides in the main memory. The audio driver manages the DMAs, and also reads and writes data from and to the memory buffer, upon demand. The data in the buffer is either the aggregated output audio stream generated by mixing together all the application's private streams, or an input stream received from a codec.

The application's audio streams are stored in private buffers in the main memory, and the mixing is the numerical sum of the respective values contributed by each audio stream.

Modern operating systems support audio loopback, by simply duplicating a selected aggregated audio stream's memory buffer (Fig. 2) into an input buffer; therefore, the loopback audio buffer content is bit-accurate in the case of a single stream. In the past, audio loopback was supported only as part of the device driver of some audio devices. However, audio loopback has been part of the operating system itself in Windows Vista and beyond as well as in any Linux kernel (see ALSA [4]). It is thus ubiquitous.

In the case of a VM (Fig. 3), the aggregated audio stream is created within the VM, and the VM managing application plays its own aggregated stream as any other application running on the host OS. However, the loopback buffer of a VM is managed internally, and it is originated by its aggregated audio stream only. Therefore, a VM cannot receive a loopback audio stream originating outside the VM, but it may create an audio stream that is received outside.

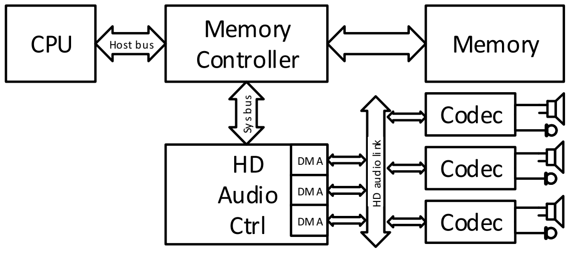

Fig. 1. The Audio Interface Architecture

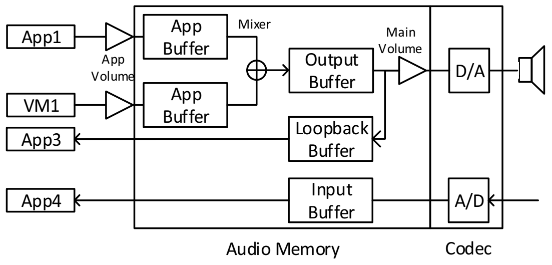

Fig. 2. The Audio System and Audio Loopback

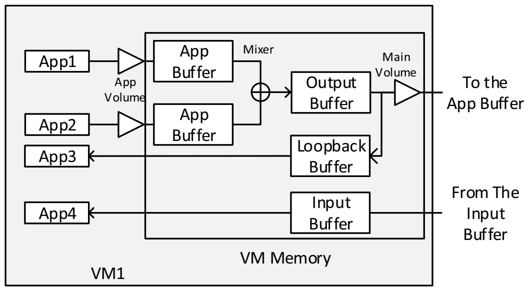

Fig. 3. The VM's Connectivity to the Audio System

III. SOUND COVERT

We now present Sound Covert. First, we discuss the advantages of the audio interface being used for a covert channel. Then, we discuss audio data encoding, and present how it may be used for data transfer.

A. *The Method*

Sound Covert exploits the machine's audio buffers to transfer data between applications running on the same machine. The sender encodes the data as an audio stream and plays it on the audio device. The receiver 'listens' to the sound being played using audio loopback, and decodes its message. Accessing the audio buffer, which is commonly enabled by default in modern operating systems, does not require permission.

As most of the applications running on the same machine see the same audio buffers, audio loopback may act as a memory buffer, through which data may be communicated between applications. This is also feasible on a machine with many audio devices, as the receiver may listen to all the existing devices simultaneously. It is, nonetheless, required that both sender and receiver be able to access the same buffer. In a non-virtualized machine, this enables communication between any applications. In a virtualized environment, each VM has its own buffer, from which outgoing audio data is copied to the machine's buffer (of the hypervisor or the host OS). Therefore, the receiving application must be running either in the sender's VM or directly under the host OS.

For simplicity, we initially assume that no other audio is currently being played in the system. The case of multiple concurrent audio sources (targeting the same audio device) is discussed in Section V.

Encoding data as an audio stream raises an important question: Are speakers connected to the machine? If not, audibility is not a problem and the sender may send its data as an audio stream (almost) without any further restrictions. Yet, most operating systems do not allow non-privileged processes to determine whether speakers are connected or not. Since this information is rarely available, the sender must assume that speakers are connected.

In order to avoid drawing the user's attention to the covert communication, the sender may choose to mute the main

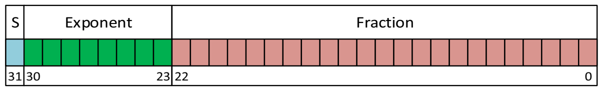

Fig. 4. IEEE 754 Single Precision Floating Point

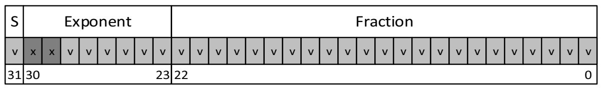
S
Exponent
Fraction
v x x v v v v v v v v v v v v v v v v v v v v v v v v v v v v v
31
30
23
22
0

Fig. 5. Using IEEE 754 for Data Transfer

volume of the audio interface (Fig. 2); although so doing has no implication on the data being transferred using audio loopback, it has a visible implication in modern operating system's GUI, and may also mute the user's audio being played (which may draw the user's attention). Note that there is also a per-application volume controller (Fig. 2) that simply multiplies the application's digital audio data by a constant $0 \leq c \leq 1$ before it is combined with other applications' streams. Therefore, muting the sender's volume ($c = 0$) would block its ability to communicate.

Alternatively, the actual audio stream used for encoding the data may be 'silent', inaudible. To do so, we suggest two possible approaches for encoding the data:

- Encode the data using ultrasonic frequencies, beyond the perception of the human ear, and employ a decoding filter at the receiver
- Encode the data while limiting the audio amplitude, such that the audio device produces an inaudible output. ('Silent Stream', discussed next.)

The use of ultrasonic frequencies was proposed in [13,14,39], both for covert communication in the open air (using speakers and microphones), and for audio steganography. Its advantage is being able to communicate simultaneously with another audio stream being played, and its disadvantages are a relatively low communication bit rate and a possible introduction of audible noise. In this work we mostly focus on our novel Silent Stream approach, yet in Section V we also discuss the possible benefits of using ultrasonic frequencies. In fact, the coding method is orthogonal to the main idea of Sound Covert.

Encoding data as an audio stream requires a good understanding of the audio data representation. Not only is this required for producing 'legal' values (or else the audio stream may get discarded) and for producing a Silent Stream, it also determines the attainable data rate.

Audio representation in modern operating systems uses 32-bit IEEE 754 single precision floating point numbers [5], which supports a huge range of values. An important observation is that much of this range is non-audible. Data can be encoded using the non-audible levels only, yet utilizing most of the 32 bits. For that reason, encoding data with Silent Stream is also much simpler to implement than modulating it over ultrasonic frequencies. We will now discuss the IEEE 754 32-bit single precision floating point, restrictions that come when it is used for audio, and the way we use it for data encoding.

B. Data Encoding

In this subsection we present a method for representation of data as an audio stream, which results in a Silent Stream. For simplicity, our objective is to fix certain bits of the audio stream, while all others can assume any combination of values.

First, we describe the 32-bit IEEE 745 single precision floating point audio data representation, and its requirements for a legal audio stream. Then, we experimentally determine the largest signal amplitude that still forms a Silent Stream, and derive the identity of the bits whose values must be fixed along with their required values for that purpose. Finally, using the same number of non-fixed bits found before, we choose the identity of the fixed and unfixed bits differently, such that the audio stream's amplitude is as low as possible.

An audio stream comprises a sequence of samples sent at a given rate for each of the channels supported by the audio device. Each sample comprises 32 bits representing the floating point range [-1.0:1.0] in 32-bit IEEE 754 single precision floating point format. In this format (Fig. 4), bit 31 is the sign s, bits 30:23 are the biased exponent e, and bits 22:0 are the fraction f. (We use the notation of F[i:j] to refer to the IEEE 754 bit range [i..j].) The value of the floating point number is:

$$fval = (-1)^s \cdot (1 + \textstyle\sum_{i=0}^{22} f_{22-i} 2^{-(i+1)}) \cdot 2^{(e-127)} ,$$

where f_n stands for the n'th bit of f, and $n = [0..22]$.

Legal stream

Audio values must be within the range of [-1.0:1.0], or else the data is considered illegal and the audio driver may choose to discard playing the whole audio stream. Setting aside $(-1)^s$ for expressing the sign, the remainder is a positive number, denoted pfval, with the following requirement:

$$pfval = (1 + \sum_{i=0}^{22} f_{22-i} 2^{-(i+1)}) \cdot 2^{(e-127)} \leq 1.$$

Silent Stream

A Silent Stream can be produced using only a small fraction of this range, $(-\varepsilon : \varepsilon)$, as [-1.0:1.0] refers to the maximum audio amplitude.

We generated sine waves with different amplitudes, $\varepsilon \cdot \sin(x)$, striving to find the non-audible range. The values of ε were chosen using fval from eq. (1) with all the bits of f set to '1' (F[22:0]=0x7FFFFF), running through different values of e (F[30:23]). We used this approach because of the +1 added to the value of the fraction; this restricts its dynamic range, thereby causing the value of the exponent to be the sole value limiter. Given e, setting the bits of f to any other combination of values will only decrease the value of fval, staying within the non-audible range. (The experiment for non-audibility is fully described in Section IV.A.)

We found that $e \leq 111$ (F[30:23] $\leq$0x6F) resulted in a Silent Stream, limiting ε to $0 \leq \varepsilon \leq 3.051\ x\ 10^{-5}$. Therefore, by fixing bits 4 and 7 of e to '0' (F[27], F[30]=0), we may freely set all its other bits and still keep it within the range of $0 \leq e \leq 111$ (although not reaching all possible values of e). All the f_n bits (F[22:0]) are unrestricted. This results in 30 non-fixed bits that can freely be used for data representation (F[31], F[29:28], F[26:0]).

Smallest Amplitude

We found the maximum signal amplitude that is 'silent', and derived from that which bits to fix and which to use for data encoding. However, we can use the exact number of fixed bits (2), yet produce a stream with a much smaller amplitude, far below the audio device's threshold for producing any sound. (See Section IV.A. for audio detectability by sensing equipment.)

We choose to fix bits 6 and 7 of e (F[30:29]) to '0', limiting e to 63 (F[30:23]<0x3F). This will limit ε to a much smaller value than before, $0 \leq \varepsilon \leq 1.084\, x\, 10^{-19}$, which is 14 orders of magnitude smaller than the non-audible limit we found. As before, all the f_n bits (F[22:0]) can be used freely.

Conclusion: For using an audio stream for data transfer in Sound Convert, we put our data in bits F[31], F[28:0] (30 out of the 32 bits) of the 32-bit floating point number (Fig. 5, where useable bits are marked as 'v' and unusable in 'x'), and send the audio stream for play. This audio stream, containing our data, is a legal Silent Stream.

Data rate: A typical audio device employs a 44.1 kHz sampling rate for each of two channels (stereo), and each sample is a 32-bit floating point number. The sampling rate and number of channels are device specific, yet to our knowledge no modern audio device goes below the values stated above. The sample size is defined by the operating system, and 32 bits are always supported.

As shown before, we may use 30 bits of each sample to send our data, so the maximum data rate is 44,100 x 2 x 30 = 2.646 Mbps. This is based on minimum characteristics of any common audio device, so it is a lower bound.

Further data rate improvement

Silent Stream requires that $e \leq 111$, thus 112 different values of e may be used. This yields in $\log_2 112 = 6.8$ effective bits of e, optimally. Using e in an optimal manner requires more complicated encoding than our previously suggested sub-optimal scheme (where the bits of e are not fixed), at the benefit of 0.8 extra effective bit (out of 30), which is 2.66% improvement in data rate. For the added complexity and yet small additional benefit, we recommend our sub-optimal simple encoding scheme.

IV. IMPLEMENTATION

In this section, we describe our implementations of Sound Covert and the experimental results. To test our Silent Stream method, we experimented with bit-accurate audio streams played alone (no other audio activity) to transfer data. More complex situations such as simultaneous audio streams are discussed in Section V.

As a preliminary step, we conducted a set of experiments to detect the range of values (in the audio buffer) that can be used without creating an audible result.

A. *"Silence" Range Determination*

To find the non-audible ("silent") range of audio amplitudes$(-\varepsilon{:}\,\varepsilon)$, we generated a low amplitude 400Hz sine wave with $\varepsilon = 1.175\, x\, 10^{-38}$, and gradually increased its amplitude. Then we played it using professional audio equipment and listened to the result. We used Roland SC-D70 audio interface, connected digitally to Roland DS-7 amplified studio monitors through S/PDIF [20] over a coaxial cable. The studio monitors performed the D/A conversion and amplification (Fig. 6.a). We repeated the experiment with another setup of professional studio equipment, using Antelope Zen Studio audio interface with AKG K240 Studio earphones (Fig. 6.b). We set all volumes to maximum, including the volume of the device driver, of the monitor's amplifier and of the audio interface's earphones amplifier.

Results: The minimum audible ε found is $3.051\, x\, 10^{-5}$.

To check for possible detection by hardware, we repeated the experiment with a Shure SM57 microphone [23] as the listener, connected to Antelope Zen Studio audio interface, located 5 centimeters from the studio monitor (Fig. 6.c). We measured the signal detected by the microphone.

Results: Signal perceived by the microphone at $\varepsilon = 3.814\, x\, 10^{-6}$, roughly one order of magnitude more sensitive than the human perception.

This may be audio equipment dependent; however, as this result is 13 orders of magnitude louder than the range we actually use for Sound Covert, we consider Sound Covert to be completely undetectable by audio sensing equipment.

B. *Using Sound Covert for Communication*

To prove the feasibility of Sound Covert and assess its data rate, we implemented two applications, one acting as the sender and one as the receiver. The applications were implemented and tested under Microsoft Windows 7, but the implementation for Linux is similar.

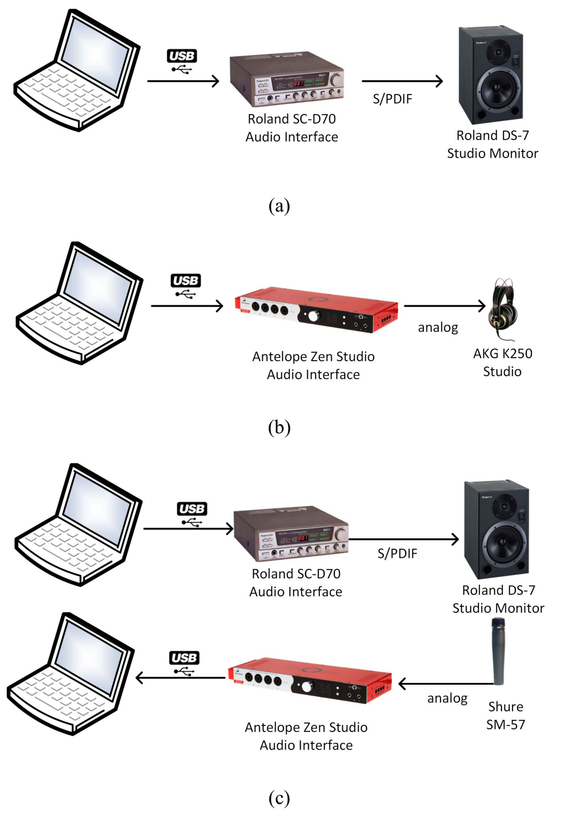
USB
Roland SC-D70 Audio Interface
S/PDIF
Roland DS-7 Studio Monitor
(a)
USB
Antelope Zen Studio Audio Interface
analog
AKG K250 Studio
(b)
USB
Roland SC-D70 Audio Interface
S/PDIF
Roland DS-7 Studio Monitor
USB
Antelope Zen Studio Audio Interface
analog
Shure SM-57
(c)

Fig. 6. Audibility test using: (a) a studio monitor; (b) studio earphones; (c) a closed loop with microphone.

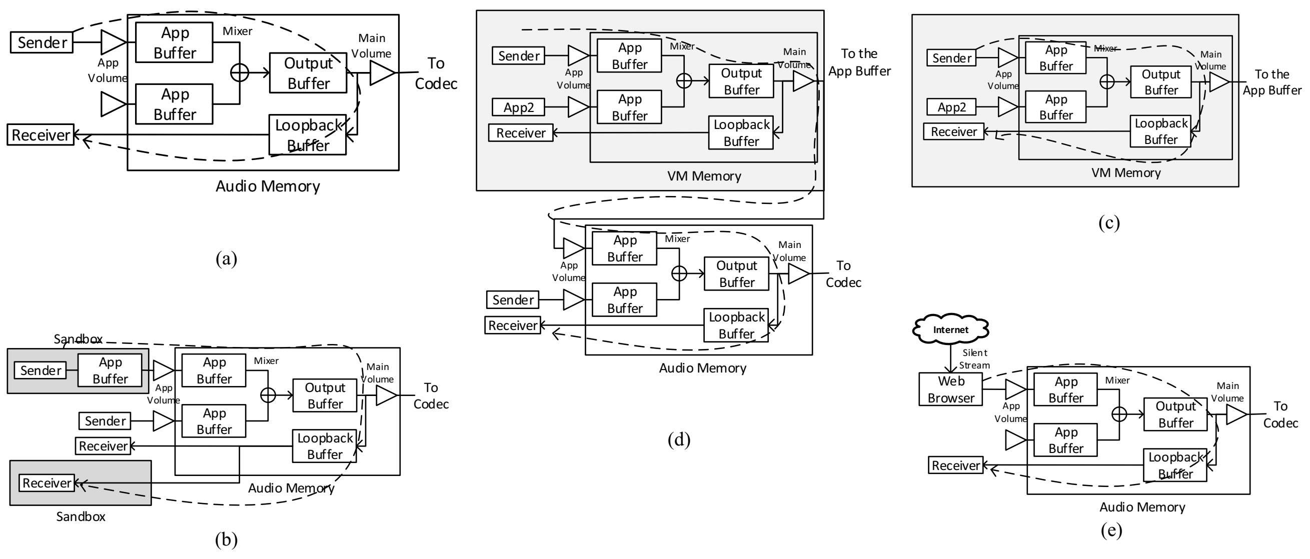

Fig. 7. Sound Covert through: (a) host OS; (b) sandbox and the host OS; (c) inside a VM; (d) Between a VM and the host OS (e) Using the web browser

The sender reads a file from disk and sends it using Sound Covert. The receiver monitors the audio loopback buffer and receives the file when detected. The sender uses start and finish markers, thereby enabling the receiver to distinguish a file sent to it from any other audio being played. The markers are 256 bit random numbers (chosen once and agreed upon between the sender and receiver), so the probability of false detection of start or finish markers is extremely low.

We then tested the file copy through Sound Covert in several different settings, in which we sent files of various sizes (in the range of 1MB to 10MB) from a sender application to a receiver application. The data rate was calculated by the receiver by simply measuring the time it took to receive the file. We used Windows 7 SP1 64 bit operating system as the bare-metal (sometimes host) and VM operating system, and tested it both with the Intel High Definition Audio [24] on-board audio interface, with Roland SC-D70, and with Antelope Zen Studio.

Settings. The settings that were tested:

A. Two applications running on the same host OS. (Fig. 7.a)
B. An application running natively on the host OS and an application running inside a sandbox, using Sandboxie [27], bidirectional communication. (Fig. 7.b)
C. Two applications running inside the same virtual machine, using both Oracle VM VirtualBox [28]. (Fig. 7.c)
D. An application running inside a virtual machine, and an application running on the host machine, single direction (Fig. 7.d). Although in some cases the host OS is able to directly spy into the VM running on it, this generally requires high permissions from the host OS. In other cases [21,19], it may not be possible due to hardware based virtual compartments or encryption. Sound Covert is able to communicate with the VM internals with no added permissions, and on top of currently available security compartments.

Results: In settings A-C, the file was copied without errors at a measured data rate of 2.6Mb/s, regardless of its size, nearly equal to the theoretical maximum. We limited the value range used for data encoding to $1.084\ x\ 10^{-19}$ (per Section III.B), so no sound was heard during operation.

In setting D, as the VM application downscales its output audio stream to 24 bits, it reduced the number of bits we can actually use in the audio stream to 1.9Mb/s. Unlike the previous settings, this setting only allows unidirectional communication between the applications, from an application running within a VM to one running directly on the host OS.

Native Receiver and Web Browser. Next, instead of two applications running on the same physical machine, the receiving application ran on the (attacked) target machine; the sender was placed remotely, embedding the file to be sent in the form of an audio stream as content of a web page.

The web browser acts as a pseudo-sender, an uncompromised application that is capable of playing a wave file (received by network), and therefore acts as a sender on the target machine. The (malicious) receiver does not communicate with the network directly, so it cannot be blocked by a firewall and goes unnoticed by anti-malware software. This setting is useful for distributing data to trojan receivers by breaking into a commonly accessed website. (Fig. 7.e)

By simply embedding the wave file into a website (e.g. by using HTML5 <audio> [36]), the web browser reduced the audio stream's data rate, so we attempted a different approach. Instead, we used a web page with JavaScript [30] to load the wave file, and Mozilla Developer Network's (MDN) Audio Buffer Web API [31] for setting exact values to the audio stream. This enabled us to control the 32-bit audio stream in a bit-accurate manner. Audio Buffer Web API is commonly used in most of the modern web browsers [31]. We used Mozilla Firefox [32] v.44.0.2 as the browser.

This setting allows unidirectional communication through Sound Covert, as the web browser cannot submit audio information remotely without asking for the user's permission.

Results: The file was copied correctly, at 2.6Mb/s.

Remark. In a real setting, a user of the target machine should direct its browser to a compromised page. The attacker should thus try to compromise as many pages as possible, and to make use of any available information pertaining to the target machine users' habits. A compromised proxy server could also be used to direct the user to such a page, even temporarily, merely to allow the sender to send the desired data to the target machine. Even if the user notices something strange, the phenomenon would go away and if the sender immediately removes the redirection from the proxy it will be impossible to reconstruct the apparent transient abnormality.

Table 1 summarizes all the implementations tested, detailing the measured data rate if successful or 'No' if unsuccessful. (The browser runs inside the target (receiver's) VM; each sandbox runs only a single application.)

TABLE I. CAPABILITIES AND MEASURED PERFORMANCE

To / From	HostOS	Sandbox	VM	Web Browser
HostOS	2.6Mb/s	2.6 Mb/s	No	No
Sandbox	2.6Mb/s	2.6Mb/s	No	No
VM	1.9Mb/s	1.9Mb/s	2.6Mb/s	No
Web Browser	2.6 Mb/s	2.6 Mb/s	2.6Mb/s	No

V. MORE COMPLEX SCENARIOS

Sound Covert generally entails using the audio loopback feature to covertly transfer data, while the data coding method is orthogonal to that. So far, we assumed that no other audio source is active in the system, so the sender may simply play raw data encoded using our Silent Stream technique. Yet, there are cases in which the output audio buffer (which mixed together all the sound streams) is used by others as well. We wish to explore additional use cases of Sound Covert.

A. *Multiple Sound Sources*

The Silent Stream technique, presented in Section III.B., is unusable with another audible audio source played simultaneously on the device, due to the nature of floating point number addition (performed at the mixing stage), which adjusts the result's accuracy according to the value of the result. Audible audio sources must have far greater amplitude than a Silent Stream, causing the Silent Stream to be completely masked when mixed together.

Waiting. One solution is to wait until the audio device is idle, and then use Sound Covert with a Silent Stream. Although this is a naïve approach, the high data rate of Sound Covert (2.6Mbps) along with the fact that audio interfaces are rarely used extensively, makes this suitable for most cases.

If Sound Covert is used and a foreign audio stream is suddenly activated, Sound Covert will not interfere with the foreign audio stream. Both streams are mixed together, and the Silent Stream will simply disappear in the mixed audio. Error detection codes may be embedded in the data for detecting communication errors.

Ultrasonic Frequencies. The use of ultrasonic frequencies to encode data silently was discussed in some past works [13,14,39], and it is applicable for Sound Covert as well. It is usable when simultaneous audio streams are played together, though it allows small bandwidth compared to our previously suggested scheme. [25] discusses the amount of energy required for encoding data using ultrasonic waves, and implications on the resulting data rate and added audible noise.

We do, however, find this method useful with Sound Covert for broadcasting. Assuming a compromised website (or a proxy server) that broadcasts audio or video (e.g. internet radio or streaming service). With ultrasonic frequencies, secret data may be added to the streamed data without requiring to analyze it first (unlike the case in steganography). This content-unaware approach enables on-the-fly stream manipulation.

B. *Audio Steganography and Sound Covert*

Although Sound Covert achieves top data rate with our Silent Stream technique, it may also be used with traditional audio steganography. An example scenario may be using a web browser (being a pseudo-sender) that accesses a compromised web site or a compromised video or audio stream uploaded to an uncompromised web site. It then plays the media, and the receiver (using audio loopback) decodes the message without ever accessing the network directly. This is not limited to any audio steganography technique, and may also use lossy compressed media (such as [35]), commonly used on the web.

VI. COUNTERMEASURES

If one is expecting this kind of attack, Sound Covert can be detected simply by an appropriate monitor looking for abnormal traffic in the sound buffer. This, however, is true for nearly every attack, which can be detected if known and anticipated. Sound Covert can also be blocked by simply disabling the audio interface. However, while simple and useful for some settings (e.g. a public cloud), for many others this defense is unacceptable. We next discuss additional countermeasures.

The Audio Signal Path. VM host applications (e.g. [28]) implement audio device emulation internally, and then their resulting audio is aggregated together like any other application; however, audio loopback is implemented inside the VM itself. As a result, audio played by any VM, sandbox, or the host operating system, is accessible by any other sandbox and the host operating system running on the same physical machine, and Sound Covert exploits that.

We suggest a separate per-VM audio channel, holding a separate output audio buffer for each VM and the host operating system, and mix these buffers by hardware just before being played, so audio loopback is only accessible to the applications running under the same host operating system or VM. Once available by hardware, sandboxes should use the same mechanisms.

Signal Integrity. The operating system or some special process may intentionally add "silent noise" to the audio buffer. So doing would prevent bit-accurate covert transmission, thereby at least dramatically slow the covert channel. Data rate would rely on the level of "silent noise" added (optimally, at the limit of non-audibility amplitude), and the error detection and correction mechanism in use.

VII. CONCLUSIONS

We presented Sound Covert, a novel technique for forming a covert communication channel between applications running on the same physical machine. Sound Covert uses the audio buffer for transferring data between sender and receiver programs, even if they do not have permission to communicate directly. Sound Covert works across the host operating system/VM/sandbox on the machine (with the exception of not being able to go between VMs), and across different operating systems. Although it uses the machine's audio interface, Sound Covert does not produce any sound. By exploiting the huge range of the 32-bit floating point representation, we create a silent audio stream (Silent Stream), constructed of our data. Sound Covert can also provide covert communication from an external source (e.g., a compromised web page) to an application in the target machine without requiring network access by this application; this is done in the form of a silent audio stream that is placed in the audio buffer of the relevant VM in the target machine by an uncompromised and unaware browser running in the target machine (in the same VM as the receiver) and accessing the compromised web page. Similarly, an innocent, uncompromised and unaware web server running on the target machine, either in the same VM as the malicious sender or directly under the host OS, can be used with Sound Covert to send data out of the target machine. We implemented Sound Covert and showed that it may reach at least 2.6Mb/s, a dramatic (~10,000X) improvement over any published covert channel.

REFERENCES

[1] Brouchier, J., Kean, T., Marsh, C. and Naccache, D., 2009. Temperature attacks. Security & Privacy, IEEE, 7(2), pp.79-82.

[2] Percival, C., 2005. Cache missing for fun and profit.

[3] Loopback Recording, https://msdn.microsoft.com/en-us/library/windows/desktop/dd316551%28v=vs.85%29.aspx

[4] Matrix:Module-aloop, http://www.alsa-project.org/main/index.php/Matrix:Module-aloop

[5] IEEE, IEEE Standard for Floating-Point Arithmetic, 2008.

[6] Hunger, C., Kazdagli, M., Rawat, A., Dimakis, A., Vishwanath, S. and Tiwari, M., 2015, February. Understanding contention-based channels and using them for defense. In High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on (pp. 639-650). IEEE.

[7] Evtyushkin, D., Ponomarev, D. and Abu-Ghazaleh, N., 2015, June. Covert channels through branch predictors: a feasibility study. In Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy (p. 5). ACM.

[8] Saltaformaggio, B., Xu, D. and Zhang, X., 2013. Busmonitor: A hypervisor-based solution for memory bus covert channels. Proceedings of EuroSec.

[9] Wang, Y., Ferraiuolo, A. and Suh, G.E., 2014, February. Timing channel protection for a shared memory controller. In High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on (pp. 225-236). IEEE.

[10] Wu, Z., Xu, Z. and Wang, H., 2012. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In Presented as part of the 21st USENIX Security Symposium (USENIX Security 12) (pp. 159-173).

[11] Wang, Z. and Lee, R.B., 2006, December. Covert and side channels due to processor architecture. In null (pp. 473-482). IEEE.

[12] Lucian Constantin, "Utterly crazy hack uses long-distance lasers to send malware commands via all-in-one printers" http://www.pcworld.com/article/2834972/allinone-printers-can-be-used-to-control-infected-airgapped-systems-from-far-away.html

[13] Hanspach, M. and Goetz, M., 2013. On Covert Acoustical Mesh Networks in Air. Journal of Communications, 8(11).

[14] Lynch, E., Li, Y. and Zhao, W., 2014. Covert Acoustic Channels.

[15] Deshotels, L., 2014. Inaudible sound as a covert channel in mobile devices. In 8th USENIX Workshop on Offensive Technologies.

[16] High Definition Audio Specification, June 17, 2010. www.intel.com/content/dam/www/public/us/en/documents/product-specifications/high-definition-audio-specification.pdf

[17] Lampson, B.W., 1973. A note on the confinement problem. Communications of the ACM, 16(10), pp.613-615.

[18] Nittala, A.S., Yang, X.D., Bateman, S., Sharlin, E. and Greenberg, S., 2015, June. PhoneEar: interactions for mobile devices that hear high-frequency sound-encoded data. In Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems.

[19] Chen, X., Garfinkel, T., Lewis, E.C., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J. and Ports, D.R., 2008, March. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In ACM SIGARCH Computer Architecture News (Vol. 36, No. 1, pp. 2-13). ACM.

[20] Sony/Philips Digital Interface Format, http://www.hardwarebook.info/S/PDIF

[21] Zhang, F., Chen, J., Chen, H. and Zang, B., 2011, October. CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (pp. 203-216). ACM.

[22] Petitcolas, F.A., 1998. mp3stego. http://www.petitcolas.net/steganography/mp3stego/

[23] Malik, H.M., Ansari, R. and Khokhar, A.A., 2007. Robust data hiding in audio using allpass filters. IEEE Transactions on Audio, Speech, and Language Processing, 15(4), pp.1296-1304.

[24] Intel High Definition Audio, http://www.intel.com/content/www/us/en/chipsets/high-definition-audio.html

[25] Santosa, R.A. and Bao, P., 2005, June. Audio-to-image wavelet transform based audio steganography. In 47th International Symposium ELMAR, 2005. (pp. 209-212). IEEE.

[26] Audio File Format Specifications, http://www-mmsp.ece.mcgill.ca/documents/audioformats/wave/wave.html

[27] Sandboxie, http://www.sandboxie.com/

[28] VirtualBox, https://www.virtualbox.org/

[29] Matsuoka, H., 2006, December. Spread spectrum audio steganography using sub-band phase shifting. In 2006 International Conference on Intelligent Information Hiding and Multimedia (pp. 3-6). IEEE.

[30] Jayaram, P., Ranganatha, H.R. and Anupama, H.S., 2011. Information hiding using audio steganography–a survey. The International Journal of Multimedia & Its Applications (IJMA) Vol, 3, pp.86-96.

[31] Mozilla Development Network Audio Buffer Web API, https://developer.mozilla.org/en-S/docs/Web/API/AudioBuffer

[32] Firefox, https://www.mozilla.org/en-US/firefox/new/

[33] Kirovski, D. and Malvar, H.S., 2003. Spread-spectrum watermarking of audio signals. Signal Processing, IEEE Transactions on, 51(4).

[34] Radhakrishnan, R., Shanmugasundaram, K. and Memon, N., 2002, December. Data masking: a secure-covert channel paradigm. In Multimedia Signal Processing, 2002 IEEE Workshop on (pp. 339-342).

[35] Raissi, R., 2002. The theory behind MP3. MP3'Tech.

[36] HTML5 Audio, http://www.w3schools.com/html/html5_audio.asp

[37] Trithemius, J. and Heidel, W.E., 1721. Johannis Trithemii, Steganographia. WE.

[38] Sridevi, R., Damodaram, A. and Narasimham, S., 2009. Efficient Method of Audio Steganography by Modified LSB Algorithm and Strong Encryption Key with Enhanced Security. In Journal of Theoretical and Applied Information Technology.

[39] Wheeler, D., Johnson, D., Yuan, B. and Lutz, P., 2012, January. Audio Steganography Using High Frequency Noise Introduction. In Proceedings of the International Conference on Security and Management (SAM)