PKOMP 89/5

Analysis of the Bouncing Ball Virus

M S Olivier

H W Teitge

TEGNIESE VERSLAG **TECHNICAL REPORT**
SENTRUM VIR GEVORDERDE REKEN- EN BESLUITNEMINGSONDERSTEUNING
CENTRE FOR ADVANCED COMPUTING AND DECISION SUPPORT
WNNR CSIR

PKOMP 89/5, 8p, Pretoria, July 1989.

CACDS Technical Report PKOMP 89/5
Centre for Advanced Computing and Decision Support
CSIR
P O Box 395
Pretoria
0001

Printed in the
Republic of South Africa
by Scientia Printers, CSIR.
PRETORIA

Analysis of the Bouncing Ball Virus

M S Olivier

H W Teitge

Centre for Advanced Computing and Decision Support,
CSIR, P O Box 395, Pretoria 0001.

Abstract

Computer viruses have become a reality in South Africa and to safeguard computer systems, it is necessary to study every virus as it appears.

This report documents the first such study we undertook, which was of what is generally known as the Bouncing Ball Virus. A description of the major actions of the virus is given and an antidote is discussed.

Keywords: Computer virus, worm, trojan horse, bootstrap code, interrupt vector, bad sector, File Allocation Table.

1. Introduction

The origin of the virus is difficult to determine—[4] gives Italy as the origin. We became aware of it after the Pretoria Technicon Computer Science Department's LAN was reportedly infected and obtained the virus from one of their students for analysis. At that stage it was known that the virus could easily be activated by a game called Sokoban. Once the virus was activated and memory resident, it was only a matter of time before we could locate the offending code and start the dissection. The virus appears as a small ball (ASCII character 7) in the lefthand corner of the screen. It then moves diagonally across the screen and rebounds from the screen edges in the fashion of a billiard ball fashion. The ball is also deflected by some characters on the screen.

2. Classification

2.1 Terminology

There are many differrent classes of "invading" software. The following are the main categories which are not mutually exclusive.

- Class 1 - Worms
  A worm is any program which destroys ("eats") data or memory space (on disk or in primary memory). It does this by writing data to disk (or primary memory)—existing data is overwritten (in order to destroy it) or data is written to available space (in order to degrade system performance).

  In most cases worms are intended to destroy data and are therefore considered malicious.

- Class 2 - Trojan horses
  A trojan horse is a program which has some effect other than (or in addition to) the intended purpose.

  Trojan horses can activate worms, viruses or any other programs—malicious or otherwise.

  Normally trojan horses have appealing names to encourage people to execute them (and then get a "surprise").

- Class 3 - Viruses
  A virus is a program which has the ability to reproduce itself so that it can spread to other computers or disks ("hosts").

  Viruses make their presence known by a message, some other audio and/or visual effect or a worm.

2.2 The Bouncing Ball Virus

The Bouncing Ball Virus falls in the third class—it has an effective method of spreading itself to other machines. It also displays its presence (under certain conditions) by moving the ball around on the screen.

The virus is not intentionally malicious, but problems it introduces are:

- Irritation caused by the bouncing ball.
- The psychological effect caused by the discovery that the system has been invaded by uncontrollable software.
- The ease with which the virus can be adapted to be malicious.
- Customers distrust if it becomes known that a vendor's computer systems are infected.

2.3 Similar viruses

The Bouncing Ball Virus operates in a similar fashion to the well known Brain Virus [3]. Both viruses use the bootstrap sequence of the IBM PC family to install themselves in memory. Both also use "bad" clusters to store their code.

The main differences between the two viruses are the way in which they make their presence known. The Brain Virus changes the volume label while the Bouncing Ball Virus displays the ball. Known versions of the Brain Virus also take up more space (three "bad" clusters) and can only infect floppy disks.

3. Behaviour

In this section the technical aspects of the behaviour of the virus are described. Facts about how disks (both floppy and hard) are infected (contaminated), how the virus imbeds itself in the operating system (activation) and how it spreads (reproduction) are given in the following subsections.

3.1 Contamination

Analysis of the virus code indicates that contamination of a clean system is only possible if the system is booted (started) with a virus infected disk. However, it might be possible that the original infection was initiated via some executable program—a trojan horse which can be down-loaded from bulletin-boards via modems and which causes an initial infection when it is run. No evidence of such a program has as yet been found.

When a computer system is booted, the system loads the boot-sector from the boot disk. The first sector of this disk contains the bootstrap loader, which is used to load the operating system. (If the "boot disk" is a non-system disk, ie. a disk from which the system cannot be booted, this sector contains code to display a message indicating this fact and prompting the user to replace it with a system disk.) After the first sector has been loaded, the computer begins to execute this code.

On an infected disk the virus code is (partly) contained in this sector. Should a system be booted with an infected disk, the viral code is loaded from this sector and executed—this code installs the virus into memory before it loads the original contents of the first sector and transfers control to this original code. This code then causes the operating system to be loaded (or the message indicating a non-system disk to be displayed).

The contaminating code first loads itself into into RAM and stays resident. The resident code hides itself in the top 2K of RAM and informs the operating system that its available RAM is now 2K smaller: This is accomplished by subtracting two from the system variable indicating memory size. (This variable is a word situated at hex address 40:13.) The virus then copies itself (the part loaded from sector one) to this "reserved" 2K. Interrupt vector 13h is changed to point to the virus code (see next paragraph). The virus then proceeds to load its second part from disk, before it executes the normal bootstrap code as described above.

On IBM PC's and compatibles interrupt 13h is used when calling BIOS for any disk operation. The virus changes interrupt vector 13h to point to an interrupt handler in the virus code. This interrupt handler can cause a activation of the ball or reproduction of the virus. Whether activation or reproduction is caused or not, control is transfered to the original interrupt handler for interrupt 13h so that the original functions are still available. (The viral interrupt handler is thus transparent to the user and is executed whenever a disk is accessed.)

3.2 Reproduction

Whenever a disk is accessed, the viral code is executed first. It checks whether this call accesses a different drive than the previous call. If a different drive is accessed, it infects that disk (if it is not already infected). If the same drive is accessed the virus may activate the bouncing ball (see next section).

The new disk is infected by "lifting" the real boot code from sector 0 and replacing it with the first section of the virus. The File Allocation Table (FAT) is then inspected to find the first available cluster on disk. (Available clusters are marked with 000h or 0000h.) This virus will only infect disks which have at least two sectors per cluster. This available cluster is marked as bad (by writing FF7h or FFF7h to its FAT entry). The second section of the virus is written to the first sector of this cluster and the original bootstrap code is written to the second sector of this cluster. A variable in the part written to sector 0 contains the address of the other two sectors so that these two sectors can easily be located.

The virus code in sector 0 also contains a "signature" and version number. The signature is the value 1357h at a certain address in this sector. It is used by the virus to determine whether a disk has been infected, so that it is not infected more than once. The version number of the virus is also checked. If the inpecting virus has a version number higher than that of the virus on the disk, the old version on the disk is merely replaced by the later version.

3.3 Activation

When interrupt 13h occurs and the virus is not reproduced, the bouncing ball may be activated. The system timer is consulted and the virus will only become active if the time is within a small window of specific time. This allows the virus to infect many disks before it makes its presence known. This check also "randomizes" the appearance of the ball which makes tracking of the virus difficult.

System time is kept in a 64 bit variable which is incremented 18.2 times a second. The ball will only be activated if the disk is accessed when bits 28 to 5 are zero (and bits 64 to 29 and 4 to 1 have any value). This means that (unless the system time is changed) a possibility of activating the ball occurs approximately twice every hour.

Activation of the dormant virus is accomplished by pointing interrupt vector 8 (system timer) to a character displaying routine. This routine is executed a rate of 18.2 times a second. The active virus routine determines a position to display a character on the screen. It then saves the character currently displayed on the screen at that position and displays its own characteristic dot (ASCII character 7) at that position. The character overwritten by the ball during the previous interrupt 8 is also restored. Since the "ball's" position is changed once every 18th of a second, the illusion of a moving ball is created. After moving the "ball" one position, control is transferred to the normal handler for interrupt 8—usual system functions may therefore proceed normally.

4. Antidote

The antidote has four parts -

- Determination of infection
  An antidote can determine whether a disk has been infected by looking at the virus's "signature". It can verify this finding by inspecting the FAT entry where the second part of the virus and the original bootstrap code is located. The location of these sectors is stored in a pointer in sector 0. This sector is marked bad in the FAT if the disk is infected.

  After these two checks have been done, it is safe to assume that the disk has been infected. (If more versions of the virus come into circulation, it will be a good idea to check that the second sector of the bad cluster actually contains bootstrap code.)

- Determination of the virus location
  The virus code in sector 0 has a pointer to the “bad” cluster. This pointer can be used to determine the location of the rest of the virus and the original bootstrap code.
- Restoring the FAT-table (killing the virus)
  Using the sector address of the first “bad” sector and the diskette parameters, the location of the FAT entry for the “bad” cluster can be determined. (This entry will contain the value $FF7$ or $FFF7$ to indicate that it is bad.) To mark it as available, the value 000 or 0000 is written at this location in the FAT.
- Repairing the boot sector
  Using the pointer to the “bad” sector and adding one, the old boot sector can be read and written to sector 0. This completes the restoration of the disk.

5. Concluding remarks

The virus is only compatible with IBM XTs or clones. This means that only Intel 8088 and 8086 machines seems to be susceptible to the virus. (We could not infect AT clones or 386 based machines.)

Although the virus is not destructive it could easily be used as a vehicle for malicious software, such as disk formatting routines.

The only real way to protect oneself from the virus is to buy original software and not to use disks on any untrusted systems.

As a preventative measure against loss of data caused by any virus, backups should be made as regularly as possible.

Reference

1. MICROSOFT, 1985. “Macro Assembler”, *(Reference manual)* , MicroSoft
2. IBM, 1987. “Personal System/2 and Personal Computer BIOS Interface Technical Reference”, International Business Machines Corporation
3. CSSA, 1988. “CSSA Virus Information Service”, Computer Society of South Africa
4. S&S SOFTWARE, 1988. “Dr. Solomon’s Anti-virus Toolkit”,