A

Seminar report

On

Virus & Anti Viruses

Submitted in partial fulfillment of the requirement for the award of degree
Of CSE

SUBMITTED TO: **SUBMITTED BY:**

www.studymafia.org www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **Virus & Anti Viruses**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to …………..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

CONTENTS

1.INTRODUCTION

In the mid-eighties, so legend has it, the Amjad brothers of Pakistan ran a computer store. Frustrated by computer piracy, they wrote the first computer virus, a boot sector virus called Brain. From those simple beginnings, an entire counter-culture industry of virus creation and distribution emerged, leaving us today with several tens of thousands of viruses. In just over a decade, most of us have been familiar with the term computer virus.

A large portion of modern computing life is to secure the information that we are creating and processing. There are many aspects of information security, ranging from physical access to ensuring that the information has not been changed in any way. One of the most high-profile threats to information integrity is the computer virus. Surprisingly, PC viruses have been around for two-thirds of the IBM PC's lifetime, appearing in 1986. With global computing on the rise, computer viruses have had more visibility in the past two years.

Despite our awareness of computer viruses, how many of us can define what one is, or how it infects computers? This seminar aims to demystify the basics of computer viruses, summarizing what they are, how they attack and what we can do to protect ourselves against them.

2. VIRUSES

THE BASICS OF COMPUTER VIRUSES

Computer viruses are not inherently destructive. The essential feature of a computer program that causes it to be classified as a virus is not its ability to destroy data, but its ability to gain control of the computer and make a fully functional copy of itself. It can reproduce. When it is executed, it makes one or more copies of itself. Those copies may later be executed, to create still more copies, ad infinitum. Not all computer programs that are destructive are classified as viruses because they do not all reproduce, and not all viruses are destructive because reproduction is not destructive. However, all viruses do reproduce. The computer virus overcomes the roadblock of operator control by hiding itself in other programs. Thus it gains access to the CPU simply because people run programs that it happens to have attached itself to without their knowledge. A computer virus attaches itself to other programs earned it the name "virus." However that analogy is wrong since the programs it attaches to are not in any sense alive.

Virus: What exactly is a Virus?

A virus is basically an executable file which is designed such that first of all it should be able to infect documents, then it has to have the ability to survive by replicating itself and then it should also be able to avoid detection. Usually to avoid detection, a Virus disguises itself as a legitimate program which the user would not normally suspect to be a Virus. Viruses are designed to corrupt or delete data on the hard disk i.e. on the FAT (File Allocation Table).

TYPES OF VIRUSES

Computer viruses can be classified into several different types**.**

1. File or program viruses:

Some programs are viruses in disguise, when executed they load the virus in the memory along with the program and perform the predefined steps and infect the system. They infect program files like files with extensions like .EXE, .COM , .BIN , .DRV and .SYS. Some file viruses just replicate while others destroy the program being used at that time.

2. Boot Sector Viruses (MBR or Master Boot Record)

Boot sector viruses can be created without much difficulty and infect either the Master boot record of the hard disk or the floppy drive.

3. Multipartite Viruses

Multipartite viruses are the hybrid variety; they can be best described as a cross between both Boot Viruses and File viruses. They not only infect files but also infect the boot sector.

4. Stealth Viruses

They viruses are stealth in nature and use various methods to hide themselves and to avoid detection.

5. Polymorphic Viruses

They are the most difficult viruses to detect. They have the ability to mutate this means that they change the viral code known as the signature each time it spreads or infects.

6. Macro viruses

In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically users employ macros to automate repetitive tasks and there by save key strokes

THE FUNCTIONAL ELEMENTS OF A VIRUS

Every viable computer virus must have at least two basic parts, or subroutines, if it is even to be called a virus. Firstly, it must contain a *search routine*, which locates new files or new areas on disk which are worthwhile targets for infection. This routine will determine how well the virus reproduces, e.g., whether it does so quickly or slowly, whether it can infect multiple disks or a single disk, and whether it can infect every portion of a disk or just certain specific areas. As with all programs, there is a size versus functionality tradeoff here. The more sophisticated the search routine is, the more space it will take up .So although an efficient search routine may help a virus to spread faster, it will make the virus bigger, and that is not always so good.

Secondly, every computer virus must contain a routine to copy itself into the area which the search routine locates. The *copy routine* will only be sophisticated enough to do its job without getting caught. The smaller it is, the better. How small it can be will depend on how complex a virus it must copy. For example, a virus which infects only COM files can get by with a much smaller copy routine than a virus which infects EXE files. This is because the EXE file structure is much more complex, so the virus simply needs to do more to attach itself to an EXE file.

While the virus only needs to be able to locate suitable hosts and attach itself to them, it is usually helpful to incorporate some additional features into the virus to avoid detection, either by the computer user, or by commercial virus detection software. *Anti-detection routines* can either be a part of the search or copy routines, or functionally separate from them. For example, the search routine may be severely limited in scope to avoid detection. A routine which checked every file on every disk drive, without limit, would take a long time and cause enough unusual disk activity that an alert user might become suspicious. Alternatively, an Anti-detection routine might cause the virus to activate under certain special conditions. For example, it might activate only after a certain date has passed (so the virus could lie dormant for a time).

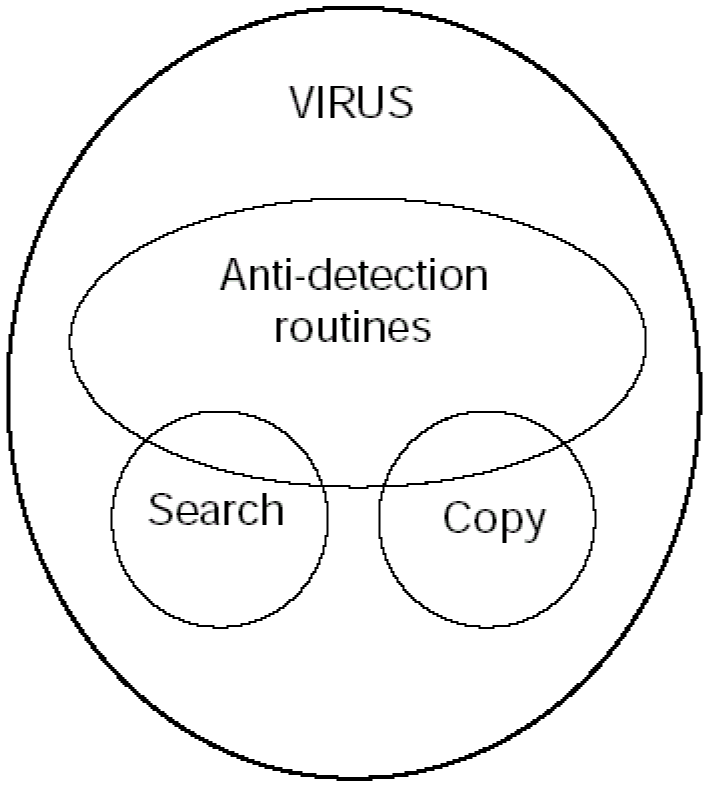

Figure 1. Functional diagram of a virus.

Alternatively, it might activate only if a key has not been pressed for five minutes (suggesting that the user was not there watching his computer). Search, copy, and anti-detection routines are the only necessary components of a computer virus, and they are the components which we will concentrate on in this volume. Of course, many computer viruses have other routines added in on top of the basic three to stop normal computer operation, to cause destruction, or to play practical jokes. Such routines may give the virus character, but they are not essential to its existence. In fact, such routines are usually very detrimental to the virus' goal of survival and self-reproduction, because they make the fact of the virus' existence known to everybody. If there is just a little more disk activity than expected, no one will probably notice, and the virus will go on its merry way. On the other hand, if the screen to one's favorite program comes up saying "Ha! Gotcha!" and then the whole

Computer locks up, with everything on it ruined, most anyone can figure out that they've been the victim of a destructive program. And if they're smart, they'll get expert help to eradicate it right away. The result is that the viruses on that particular system are killed off, either by themselves or by the clean up crew.

TOOLS NEEDED FOR WRITING VIRUSES

Viruses are written in *assembly language*. High level languages like Basic, C, and Pascal have been designed to generate stand-alone programs, but the assumptions made by these languages render them almost useless when writing viruses. They are simply incapable of performing the acrobatics required for a virus to jump from one host program to another. That is not to say that one could not design a high level language that would do the job, but no one has done so yet. Thus, to create viruses, we must use assembly language. It is just the only way we can get exacting control over all the computer system's resources and use them the way we want to, rather than the way somebody else thinks we should.

3.VIRUSES IN DETAIL

FILE OR PROGRAM VIRUSES

Some programs are viruses in disguise, when executed they load the virus in the memory along with the program and perform the predefined steps and infect the system. They infect program files like files with extensions like .EXE, .COM, .BIN, .DRV and .SYS. Some file viruses just replicate while others destroy the program being used at that time. Such viruses start replicated as soon as they are loaded into the memory. As the file viruses also destroy the program currently being used, after removing the virus or disinfecting the system, the program that got corrupted due to the file virus, too, has to be repaired or reinstalled.

A Simple COM File Infector

Some DOS Basics

EXE and COM files are directly executable by the Central Processing Unit. To execute a COM file, DOS must do some preparatory work before giving that program control. Most importantly, DOS controls and allocates memory usage in the computer. So first it checks to see if there is enough room in memory to load the program. If it can, DOS then allocates the memory required for the program. DOS simply records how much space it is making available for such and such a program, so it won't try to load another program on top of it later.

Next, DOS builds a block of memory 256 bytes long known as the *Program Segment Prefix*, or *PSP*.

Once the PSP is built, DOS takes the COM file stored on disk and loads it into memory just above the PSP, starting at offset 100H. Once this is done, DOS is almost ready to pass control to the program. Before it does, though, it must set up the registers in the CPU to certain predetermined values. First, the segment registers must be set properly, or a COM program cannot run.

COM files are designed to operate with a very simple, but limited segment structure. Namely they have one segment, **cs**=**ds**=**es**=**ss**. All data is stored in the same segment as the program code itself, and the stack shares this segment.

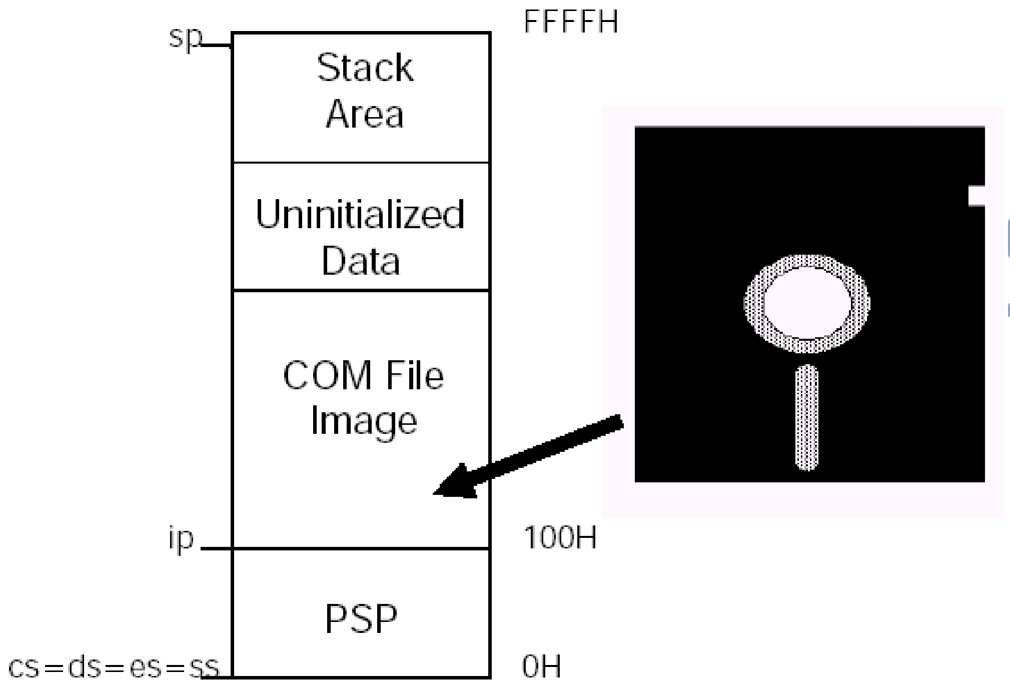

Figure 2. Memory map just before executing a COM file.

An Outline for a Virus

In order for a virus to reside in a COM file, it must get control passed to its code at some point during the execution of the program. The easiest point to take control is right at the very beginning, when DOS jumps to the start of the program.

At this time, the virus is completely free to use any space above the image of the COM file which was loaded into memory by DOS. Since the program itself has not yet executed, it cannot have set up data anywhere in memory, or moved the stack, so this is a very safe time for the virus to operate. To gain control at startup time, a virus infecting a COM file must replace the first few bytes in the COM file with a jump to the virus code, which can be appended at the end of the COM file.

Then, when the COM file is executed, it jumps to the virus, which goes about looking for more files to infect, and infecting them. When the virus is ready, it can return control to the host program. The problem in doing this is that the virus already replaced the first few bytes of the host program with its own code. Thus it must restore those bytes, and then jump back to offset 100 Hex, where the original program begins.

Step by step, it might work like this:

1. An infected COM file is loaded into memory and executed. The viral code gets control first.
2. The virus in memory searches the disk to find a suitable COM file to infect.
3. If a suitable file is found, the virus appends its own code to the end of the file.
4. Next, it reads the first few bytes of the file into memory, and writes them back out to the file in a special data area within the virus' code. The new virus will need these bytes when it executes.
5. Next the virus in memory writes a jump instruction to the beginning of the file it is infecting, which will pass control to the new virus when its host program is executed.
6. Then the virus in memory takes the bytes which were originally the first bytes in its host, and puts them back (at offset 100H).
7. Finally, the viral code jumps to offset 100 Hex and allows its host program to execute. Ok. So let's develop a real virus with these specifications. We will need both a search mechanism and a copy mechanism.

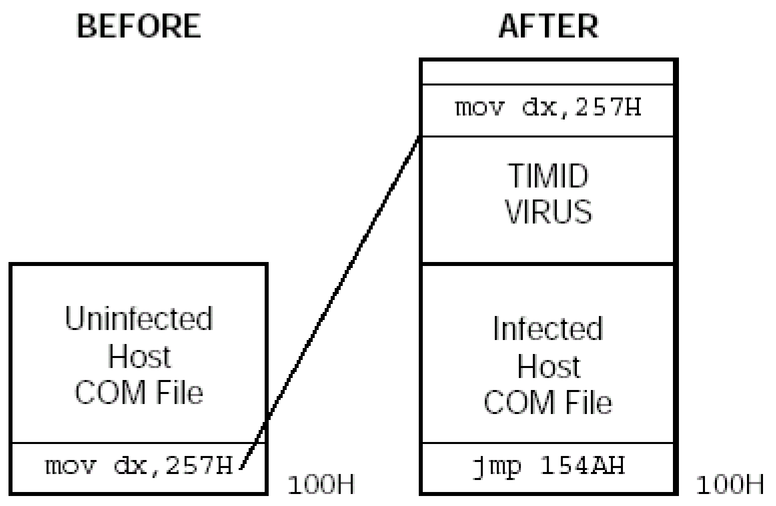

Figure 3. Replacing the first bytes in a COM file**.**

AN EXECUTABLE VIRUS

The simple COM file infector which we just developed it only attacks COM files in the current directory, it will have a hard time proliferating. In this chapter, we will develop a more sophisticated virus that will overcome these limitations. . . . a virus that can infect EXE files and jump directory to directory and drive to drive. Such improvements make the virus much more complex, and also much more dangerous.

The structure of an exe file

The EXE file is designed to allow DOS to execute programs that require more than 64 kilobytes of code, data and stack. All of this information is stored in the EXE file itself, in the *EXE Header* at the beginning of the file. This header has two parts to it, a fixed-length portion, and a variable length table of *pointers* to *segment references* in the *Load Module*, called the *Relocation Pointer Table.* Since any virus which attacks EXE files must be able to manipulate the data in the EXE Header.

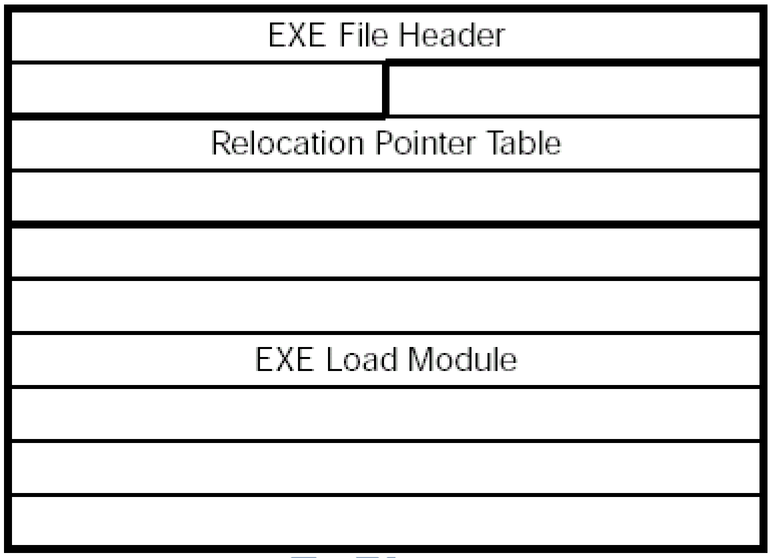

Figure 4. The layout of an EXE file.

Infecting an EXE File

A virus that is going to infect an EXE file will have to modify the EXE Header and the Relocation Pointer Table, as well as adding its own code to the Load Module. The EXE file virus will attach itself to the end of an EXE program and gain control when the program first starts. This will require a routine similar to that in COM File, which copies program code from memory to a file on disk, and then adjusts the file.

To set up segments for the virus, new initial segment values for **cs** and **ss** must be placed in the EXE file header. All the initial segment values must be calculated from the size of the load module which is being infected. Also, the old initial segments must be stored somewhere in the virus, so it can pass control back to the host program when it is finished executing. We will have to put two pointers to these segment references in the relocation pointer table, since they are relocatable references inside the virus code segment.

A Persistent File Search Mechanism

As in the TIMID virus, the search mechanism and determine whether it can be infected and make sure it has not already been infected. The only two criteria for determining whether an EXE file can be infected are whether the **Overlay Number** is zero, and whether it has enough room in its relocation pointer table for two more pointers. To determine whether the virus has already infected a file, we put an ID word with a pre-assigned value in the code segment at a fixed offset (say 0).

The procedure in COM file virus could only search for files in the current directory to attack. a good virus should be able to leap from directory to directory, and even from drive to drive. To search more than one directory, we need a *tree search routine*. For each subdirectory found, search routine will recursively call itself using the new subdirectory as the directory to perform a search on.

Passing Control to the Host

The final step the virus must take is to pass control to the host program. To do that, all the registers should be set up the same as they would be if the host program were being executed without the virus. Except for these, only the **ax** register is set to a specific value by DOS, to indicate the validity of the drive ID in the FCB's in the PSP. The DTA must also be moved when the virus is first fired up, and then restored when control is passed to the host.

A BOOT SECTOR VIRUS

The boot sector virus can be the simplest or the most sophisticated of all computer viruses. Since the boot sector is the first code to gain control after the ROM startup code, it is very difficult to stop before it loads. If one writes a boot sector virus with sufficiently sophisticated anti-detection routines, it can also be very difficult to detect after it loads, making the virus nearly invincible.

Specifically, let's look at a virus which will carefully hide itself on both floppy disks and hard disks, and will infect new disks very efficiently, rather than just at boot time. Such a virus will require more than one sector of code, so we will be faced with hiding multiple sectors on disk and loading them at boot time.

Additionally, if the virus is to infect other disks after boot-up, it must leave at least a portion of itself memory-resident. The mechanism for making the virus memory resident cannot take advantage of the DOS Keep function (Function 31H) like typical TSR programs.

Basic Structure of the Virus

Our new boot sector virus, named STEALTH, will have three parts. First, there is a new boot sector, called the *viral boot sector*. This is the sector of code that will replace the original boot sector at Track 0, Head 0, Sector 1. Secondly, there is the *main body of the virus*, which consists of several sectors of code that will be hidden on the disk. Thirdly, there is the *old boot sector*, which will be incorporated into the virus.

When the viral boot sector is loaded and executed at startup, it will go out to disk and load the main body of the virus and the old boot sector. The main body of the virus will execute, possibly infecting the hard disk, and installing itself in memory (as we will discuss in a moment) so it can infect other disks later. Then it will copy the original boot sector over the viral boot sector at 0000:7C00H, and execute it. The last step allows the disk to boot up in a normal fashion without having to bother writing code for startup.

It simply gobbles up the code that's already there and turns it to its own purposes. This strategy provides the added benefit that the boot sector virus will be completely operating system independent.

The Copy Mechanism

The biggest part of designing the copy mechanism is deciding how to hide the virus on disk. One tricky way of making the virus code totally invisible to the user is to store the data on disk in an area that is completely outside of anything that DOS (or other operating systems) can understand. In the case of floppies, an alternative is to tell DOS to reserve a certain area of the disk and stay away from it. Then the virus can put itself in that area and be sure that DOS will not see it or overwrite it. This can be accomplished by manipulating the File Attribute Table. Let's examine the 3 1/2" 720 kilobyte diskette format in detail to see how STEALTH approaches hiding itself. This kind of diskette has 80 tracks, two sides, and nine sectors per track. The virus will hide the body of its code in Track 79, Side 1 and Sectors 4 through 9. Those are the last six sectors on the disk, and consequently, the sectors least likely to contain data.

STEALTH puts the main body of its code in sectors 4 through 8, and hides the original boot sector in sector 9. However, since DOS normally uses those sectors, the virus will be overwritten unless it has a way of telling DOS to stay out. Fortunately, that can be done by modifying the FAT table to tell DOS that those sectors on the disk are bad.

If a cluster is empty, the corresponding FAT entry is 0. If it is in the middle of a file, the FAT entry is a pointer to the next cluster in the file; if it is at the end of a file, the FAT entry is FF8 through FFF. A cluster may be marked as bad by placing an FF7 Hex in its FAT entry. In the event that the diskette is full of data, the virus should ideally be polite, and avoid overwriting anything stored in the last clusters. This is easily accomplished by checking the FAT first, to see if anything is there before infecting the disk.

There are non-DOS areas on every disk. In particular, the first boot sector, which contains the partition table, is not a part of DOS. Hence finding a single area on any hard disk that does not belong to DOS is not too difficult. Although the first boot sector is located at Track 0, Head 0, Sector 1, FDISK (for all the versions I've tested) does not place the start of the first partition at Track 0, Head 0 and Sector 2. Instead, it always starts at Track 0, Head 1, and Sector 1. That means that all of Track 0, Head 0 (except the first sector) is free space.

Once a strategy for hiding the virus has been developed, the copy mechanism follows quite naturally. To infect a disk, the virus must:

1) Determine which type of disk it is going to infect, a hard disk or one of the four floppy disk types.

2) Determine whether that disk is already infected, or if there is no room for the virus. If so, the copy mechanism should not attempt to infect the disk.

3) Update the FAT tables (for floppies) to indicate that the sectors where the virus is hidden are bad sectors.

4) Move all the virus code to the hidden area on disk.

5) Read the original boot sector from the disk and write it back out to the hidden area in the sector just after the virus code.

6) Take the disk parameter data from the original boot sector (and the partition information for hard disks) and copy it into the viral boot sector. Write this new boot sector to disk as the boot sector at Track 0, Head 0 and Sector 1.

The Search Mechanism

Searching for uninfected disks is not very difficult. We could put an ID byte in the viral boot sector so when the virus reads the boot sector on a disk and finds the ID; it knows the disk is infected. Otherwise it can infect the disk. Infecting floppy disks and hard disks are entirely different matters. Then if a user leaves an infected diskette in drive A and turns on his machine, his hard drive is infected immediately.

On the other hand, once a hard disk has the virus on it, In order to infect the floppy disk the virus must be present in memory when the diskettes are in the floppy drive. That means when the virus is loaded from a hard drive, it must become memory-resident and stay there. If the virus were to trigger when the boot sector itself is read, the disk would be infected immediately, since the boot sector on a newly inserted floppy drive is read before anything else is done. It will go into the infection sequence any time that the boot sector is read. That means that when the virus is active, any time you so much as insert a floppy disk into the drive, and do a directory listing (or any other operation that reads the disk), it will immediately become infected. To implement this search mechanism, the STEALTH virus must intercept Interrupt 13H, the BIOS disk service, at boot time,

Installing the Virus in Memory

Before the virus passes control to the original boot sector, which will load DOS, it must set itself up in memory somewhere where it won't get touched. The basic idea involved here is that DOS uses a number stored at 0040:0013 Hex, which contains the size of available memory in kilobytes. This number is set up by the BIOS before it reads the boot sector. It may have a value ranging up to 640 = 280H. When the BIOS set this parameter up, it looks to see how much memory is actually installed in the computer, and reports it here. However, something could come along before DOS loads and change this number to a smaller value. In such a situation, DOS will not use all the memory that is available in the system, but only what it's told to use by this memory size variable. Memory above that point will be reserved, and DOS won't touch it.

The two responsibilities of the viral boot sector are to load the main body of the virus into memory, and then to load and execute the original boot sector. When the BIOS loads the viral boot sector (and it loads whatever is placed at Track 0, Head 0, Sector 1), that sector first moves itself into the highest 512 bytes of memory (within the 640 kilobyte limit). In a machine with 640K of memory, the first unoccupied byte of memory is at A000:0000. The boot sector will move itself to the first 512 bytes just below this. Since that sector was compiled with an offset of 7C00 Hex, it must relocate to 9820:7C00 Hex (which is right below A000:0000), as desired. Next, the viral boot sector will read the 6 sector long main body of the virus into memory just below this, from 9820:7000 to 9820:7BFF. The original boot sector occupies 9820:7A00 to 9820:7BFF (since it is the sixth of six sectors loaded).

The viral boot sector then subtracts 4 from the byte at 0040:0013H to reserve 4 kilobytes of memory for the virus. Next, the viral boot sector reroutes Interrupt 13H to the virus. Finally, it moves the original boot sector from 9820:7A00 to 0000:7C00 and executes it. The original boot sector proceeds to load DOS and get the computer up and running, oblivious to the fact that the system is infected.

MULTIPARTITE VIRUSES

Multipartite viruses are the hybrid variety; they can be best described as a cross between both Boot Viruses and File viruses. They not only infect files but also infect the boot sector. They are more destructive and more difficult to remove. First of all, they infect program files and when the infected program is launched or run, the multipartite viruses start infecting the boot sector too. Now the interesting thing about these viruses is the fact that they do not stop, once the boot sector is infected. Now after the boot sector is infected, when the system is booted, they load into the memory and start infecting other program files. Some popular examples would be Invader and Flip etc.

STEALTH VIRUSES

They viruses are stealth in nature and use various methods to hide themselves and to avoid detection. They sometimes remove themselves from the memory temporarily to avoid detection and hiding from virus scanners. Some can also redirect the disk head to read another sector instead of the sector in which they reside. Some stealth viruses like the Whale conceal the increase in the length of the infected file and display the original length by reducing the size by the same amount as that of the increase, so as to avoid detection from scanners. For example, the whale virus adds 9216 bytes to an infected file and then the virus subtracts the same number of bytes i.e. 9216 from the size given in the directory. They are somewhat difficult to detect.

POLYMORPHIC VIRUSES

They are the most difficult viruses to detect. They have the ability to mutate this means that they change the viral code known as the signature each time it spreads or infects. Thus Antiviruses which look for specific virus codes are not able to detect such viruses. Now what exactly is a Viral Signature? Basically the Signature can be defined as the specific fingerprint of a particular virus which is a string of bytes taken from the code of the virus. Antiviral softwares maintain a database of known virus signatures and look for a match each time they scan for viruses. As we see a new virus almost everyday, this database of Virus Signatures has to be kept updated. This is the reason why the Antivirus vendors provide updates.

How does a Polymorphic Virus Strike?

1. The User copies an infected file to the disk.
2. When the infected file is run, it loads the Virus into the memory or the RAM.
3. The new virus looks for a host and starts infecting other files on the disk.
4. The virus makes copies of itself on the disk.
5. The mutation engines on the new viruses generate a new unique encryptic code which is developed due to a new unique algorithm.

Thus it avoids detecting from Check summers.

MACRO VIRUSES

In essence, a macro is an executable program embedded in a word processing document or other type of file. Typically users employ macros to automate repetitive tasks and there by save key strokes. The macro language is some type of basic programming language. A user might define a sequence of key strokes in a macro and set it up so that a macro is invoked when a function key is invoked. Common auto executing events are opening a file, closing file etc. Once a macro is running it can copy itself to other documents, deleting files etc.

How does a Macro Virus strike?

1. The user gets an infected Office Document by email or by any other medium.
2. The infected document is opened by the user.
3. The evil Macro code looks for the event to occur which is set as the event handler at which the Virus is set off or starts infecting other files.

Macro viruses include "Concept," "Melissa," and "Have a Nice Day."

4. ANTIVIRUS APPROACHES

The ideal solution to the threat of viruses is prevention. Do not allow a virus is get into the system in first place. This goal is in general difficult to achieve, although prevention can reduce the no: of successful viral attacks. The next best approach is to be able to do the following.

- **Detection:** Once the infection has occurred, determine that it has occurred and locate the virus.
- **Identification:** Once detection has been achieved, identify the specific virus has infected a program.
- **Removal:** Once the specific virus has been identified, remove all traces of the virus from the infected program and restore it to its original state.

Advances in viruses and antivirus technology go hand in hand. As the virus arms race has evolved, both viruses and antivirus software have grown more complex and sophisticated. There are three main kinds of anti-virus programs [McAfee]. Essentially these are scanners, monitors and integrity checkers.

SCANNERS

Scanners are programs that scan the executable objects (files and boot sectors) for the presence of code sequences that are present in the known viruses. Currently, these are the most popular and the most widely used kind of anti-virus programs. There are some variations of the scanning technique, like virus removal programs (programs that can "repair" the infected objects by removing the virus from them), resident scanners (programs that are constantly active in memory and scan every file before it is executed), virus identifiers (programs that can recognize the particular virus variant exactly by keeping some kind of map of the non-modifiable parts of the virus body and their checksums), heuristic analyzers (programs that scan for particular sequences of instructions that perform some virus-like functions), and so on.

The reason that this kind of anti-virus program is so widely used nowadays is that they are relatively easy to maintain. This is especially true for the programs which just report the infection by a known virus variant, without attempting exact identification or removal. They consist mainly of a searching engine and a database of code sequences (often called virus signatures or scan strings) that are present in the known viruses. When a new virus appears, the author of the scanner needs just to pick a good signature (which is present in each copy of the virus and in the same time is unlikely to be found in any legitimate program) and to add it to the scanner's database. Often this can be done very quickly and without a detailed disassembly and understanding of the particular virus.

Furthermore, scanning of any new software is the only way to detect viruses before they have the chance to get executed. Having in mind that in most operating systems for personal computers the program being executed has the full rights to access and/or modify any memory location (including the operating system itself), it is preferable that the infected programs do not get any chance to be executed.

At last, even if the computer is protected by another (not virus-specific) defense, a scanner will still be needed. The reason is that when the non virus-specific defense detects a virus-like behavior, the user usually wants to identify the particular virus, which is attacking the system - for instance, to figure out the possible side-effects or intentional damage, or at least to identify all infected objects.

Unfortunately, the scanners have several very serious drawbacks. The main one is that they must be constantly kept up-to-date. Since they can detect only the known viruses, any new virus presents a danger, because it can bypass a scanner-only based protection. In fact, an old scanner is worse than no protection at all - since it provides a false sense of security.

Simultaneously, it is very difficult to keep a scanner up-to-date. In order to produce an update, which can detect a particular new virus, the author of the scanner must obtain a sample of the virus, disassemble it, understand it, pick a good scan string that is characteristic for this virus and is unlikely to cause a false positive alert, incorporate this string in the scanner, and ship the update to the users. This can take quite a lot of time. And new viruses are created every day - with a current rate of up to 100 per month. Very few anti-virus producers are able to keep up-to-date with such a production rate. One can even argue that the scanners are somehow responsible for the existence of so many virus variants. Indeed, since it is so easy to modify a virus in order to avoid a particular scanner, lots of "wannabe" virus writers are doing it.

However, the fact that the scanners are obsolete as a single line of defense against the computer viruses became obvious only with the appearance of the polymorphic viruses. These are viruses, which use a variable encryption scheme to encode their body and which even modify the small decryption routine, so that the virus looks differently in each infected file. It is impossible to pick a simple sequence of bytes that will be present in all infected files and use it as a scan string. Such sequence simply does not exist. Some polymorphic viruses can be detected using a wildcard scan string, but more and more viruses appear today, which cannot be detected even if the scan string is allowed to contain wildcard bytes.

The only possible way to detect such viruses is to understand their mutation engine in detail. Then one has to construct an algorithmic "scanning engine" specific to the particular virus. However, this is a very time-consuming and effort-expensive task, so many of the existing scanners have problems with the polymorphic viruses. And we are going to see more such viruses in the future. The Bulgarian virus writer known under the handle Dark Avenger has even released a "mutating engine" - a tool for building extremely polymorphic viruses... Very few scanners are able to detect the viruses, which are using it, with 100 reliability.

One last drawback of the scanners is that scanning for lots of viruses can be very time-consuming. The number of currently existing viruses is about 1,600 and is expected to reach 3,000 at the end of 1992. Indeed, some scanners use clever scanning methods like fixed-point scanning, top-and-tail scanning, hashing and so on. The detailed description of these methods is outside the scope of this paper, but as has been proved in [Cohen90], scanning is not cost-effective in the long run, despite the scanning method used.

MONITORS

The monitoring programs are memory resident programs, which constantly monitor some functions of the operating system. Those are the functions that are considered to be dangerous and indicative for virus-like behavior. Such functions include modifying an executable file, direct access of the disk bypassing the operating system, and so on. When a program tries to use such a function, the monitoring program intercepts it and either denies it completely or asks the user for confirmation.

Unlike the scanners, the monitors are not virus-specific and therefore need not to be constantly updated. Unfortunately, they have other very serious drawbacks - drawbacks that make them even weaker than the scanners as an anti-virus defense and almost unusable today.

The most serious drawback of the monitors is that they can be easily bypassed by the so-called tunneling viruses. The reason for this is the total lack of memory protection in most operating systems for personal computers. Any program that is being executed (including the virus) has full access to read and/or modify any area of the computer's memory - including the parts of the operating system. Therefore, any monitoring program can be disabled because the virus could simply patch it in the memory. There are other clever techniques as interrupt tracing, DOS scanning, and so on, which allow the viruses to find the original handlers of any operating system function. Afterwards, this function can be called directly, thus bypassing any monitoring programs, which watch for it.

Another drawback of the monitoring programs is that they try to detect a virus by its behavior. This is essentially impossible in the general case, as proven in [Cohen84]. Therefore, they cause many false alarms - since the functions that are expected to be used by the computer viruses usually have pretty legitimate use by the normal programs. And if the user gets used to the false alerts, s/he will be likely to oversee a real one.

The monitoring programs are also completely useless against the slow viruses, described later in this paper.

INTEGRITY CHECKING PROGRAMS.

Therefore, in order to be a virus, a program must be able to infect. And, in order to infect, the program must cause modifications to the programs that are infected. Therefore, a program, which can detect that the other executable objects have been modified, will be able to detect the infection. Such programs are usually called integrity checkers.

The integrity checkers compute some kind of checksum of the executable code in a computer system and store it in a database. The checksums are re-computed periodically and compared with the stored originals. Several authors point out that in order to avoid forging attempts from the part of the virus, the checksums must be cryptographically strong. This can be achieved by using some kind of trap-door one-way function, which is algorithmically difficult to be inverted. Such functions include DES, MD4, MD5, and so on. But, as has been shown by [Radai], this is not mandatory. A simple CRC is sufficient, if implemented correctly.

There are several kinds of integrity checkers. The most widely used ones are the off-line integrity checkers, which are run to check the integrity of all the executable code on a computer system. Another kind is the integrity modules, which can be attached (with the help of a special program) to the executable files, so that when the latter started will check their own integrity. Unfortunately, this is not a good idea, since not all executable objects can be "immunized" this way. Additionally, the "immunization" itself can be easily bypassed by stealth viruses, as described later in this paper. The third kind of integrity software is the integrity shells. They are resident programs, similar to the resident scanners, which check the integrity of an object only at the moment when this object is about to be executed. These are the least widespread anti-virus programs today, but the specialists predict them a bright future [Cohen90].

The integrity checking programs are not virus-specific and therefore do not need constant updating like the scanners. They do not try to block virus replication attempts like the monitoring programs and therefore cannot be bypassed by the tunneling viruses. In fact, as demonstrated by [Cohen90], they are currently the most cost-effective and sound line of defense against the computer viruses.

They also have some drawbacks. For instance, they cannot prevent an infection - they are able only to detect and report it after the fact. Second, they must be installed on a virus-free system; otherwise they will compute and store the checksums of already infected objects. Therefore, they must be used in a combination with a scanner at least before installation. This is needed, in order to ensure that the system they are being installed on is virus-free. Third, they are prone to false positive alerts. Since they detect changes, not viruses, any change in the programs (like updating the software with a new version), is likely to trigger the alert. Sometimes this can be avoided or at least reduced by using some intelligent heuristics and educating the users. Fourth, while the integrity checkers are able to detect the virus spread and identify the newly infected objects, they usually cannot determine the initially infected object, i.e., the source of the infection.

Despite the drawbacks mentioned, the integrity checking programs are the currently most powerful line of defense against computer viruses and are likely to be used more widely in the future. Therefore, we should expect that new viruses will appear which will target the integrity programs in the same way as the polymorphic viruses are targeting the scanners and the tunneling viruses are targeting the monitors. Let's see what kinds of attacks are possible against the integrity checking programs and how these programs can be improved to avoid them.

CONCLUSION

Computer viruses are not evil and that programmers have a right to create them, posses them and experiment with them. But we should never support those people who writing viruses with destructive nature. If you do create a virus, though, be careful with it. Make sure you know it is working properly or you may wipe out your own system by accident. And make sure you don't inadvertently release it into the world.

In order to deal with the viruses it is necessary to have a deep knowledge of the way in which different viruses exploits our system's weakness, there by causing destruction of data or hampering of security. Furthermore, it is also impossible to create antivirus against a particular virus with out knowing the way it affects our system.

REFERENCES

- www.google.com
- www.wikipedia.com
- www.studymafia.org