

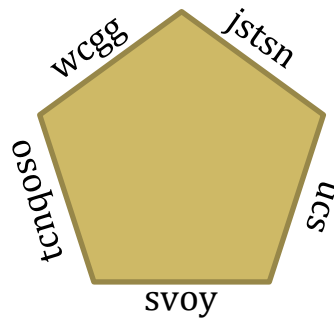
SWANSEA UNIVERSITY

COMPUTER SCIENCE

Research Methodology

Computer **Viruses** and Security threats of the web

A theoretical report in finding what **is** the **key** to solving the computer virus problem



HINT: a keyword cipher

By Dominic Chim

Supervised by Dr Ben Mora

Second Marker: Dr Anton Setzer

07 December 2014

Contents

| | |
|---|----|
| Background Information | 3 |
| Computer Virus | 3 |
| Antivirus | 5 |
| Technical analysis of viruses (Attacker) and antivirus solutions (Defender) | 6 |
| The Attackers | 6 |
| Classification of infection strategies | 6 |
| Classification of in-memory strategies | 8 |
| The Defenders | 9 |
| Antivirus solutions | 9 |
| Memory Scanning and Disinfection | 12 |
| Conclusion | 13 |
| Bibliography | 15 |

1. Overview

In this report we will discuss several aspects of computer security namely, computer viruses and explain what they are, what they can do and what has been done to counter them. We will also discuss all of this and more in a structured manner. Firstly, we will discuss in the Background section, the origins of computer viruses, how they came to be and how they work on a technical level. We will then go through what counter measures have been done to prevent if not limit the amount of damage, the afflicting virus has caused to the system and their users. During both of these stages of the report, we will discuss the technical aspects of both parties and provide an analysis on certain famous examples that has led to the evolution and progression in computer virus research.

Finally, to conclude we will summarise what we have discovered in this report and provide some closing remarks on the potential future of both computer viruses and antiviruses.

2. Background Information

This section serves to provide an insight on a computer virus; from their different forms, how they work and what special properties they have. We will also discuss the approach used to develop anti-virus solutions; this will discuss the forms of defence against computer viruses, what are their characteristics and show some potential improvements on these techniques.

2.1. Computer Virus

Virus-like programs first appeared in computers in the 1980's, however there were two famous examples before the term computer virus was coined, which were Creeper from 1971-72(Szor 2005) and John Walker's "Infective" version of UNIVAC(Szor 2005), a popular ANIMAL game in 1975.

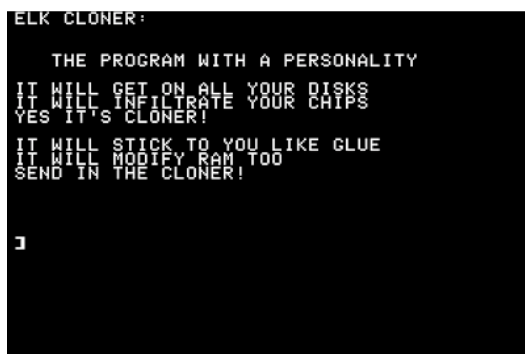
In fact, the Creeper virus and its rival Reaper(Szor 2005), which is the first "antivirus" solution for computers within a network namely, TENEX running on PDP-10s. Were both born during the early development what became "the Internet"(Szor 2005).

The first virus on computers later known as personal computers (PC) were written on the Apple II circa 1982, by Rich Skrenta(Szor 2005), where he wrote a virus called "Elk Cloner" which in time he commented that he is best known for what he called the "stupidest hack" he ever coded(Szor 2005).

The Elk Cloner virus was initially thought to be a broken piece of code by its author who continued with the hack despite this opinion. When the virus was deployed, its intention was for its comedy value however, the teachers whom were the initial victims of the Elk Cloner virus proved to be unamused by this.

The Elk Cloner virus works by activating its payload (subroutine) that displayed the author's poem after every 50th use of the infected disk, when the system was resettled, specifically on the 50th boot Elk Cloner hooked the reset handler meaning the payload would only activate after pressing the reset button. The result is shown in the figure below (Fig 1).

(Fig 1 – Elk Cloner activates (Szor 2005)).



```
ELK CLONER:

THE PROGRAM WITH A PERSONALITY
IT WILL GET ON ALL YOUR DISKS
IT WILL INFILTRATE YOUR CHIPS
YES IT'S CLONER!
IT WILL STICK TO YOU LIKE GLUE
IT WILL MODIFY RAM TOO
SEND IN THE CLONER!


```

Interestingly enough the term “Computer Virus” and thus the classification of these programs, was not introduced until in 1984, a mathematician called Dr Frederik Cohen(Cohen 1994) introduced term which consequently, named him the “father” of computer viruses due to his early studies of them. Funnily enough the term “Computer Virus” came from science fiction novels, that was picked up and recommended to Dr Cohen’s by his advisor

Professor Leonard Adleman (Szor 2005).

In Cohen’s work a formal mathematical model for computer viruses was created in 1984(Szor 2005) .

In this study Cohen provided an informal definition of a computer virus: "A virus is a program that is able to infect other programs by modifying them to include a possibly evolved copy of itself." (Szor 2005) – This definition provides the core interesting properties of a computer virus, such as the possibility of evolution i.e. the ability to make a modified copy of a piece of code with some variations.

However, there is no precise definition of a computer virus due to the abundance of various viruses, which there are infinite variations of themselves. An example of such viruses is called companion viruses(Szor 2005)that do not necessarily modify the code of other programs. Companion viruses do not strictly follow Cohen's definition because they do not need to include a copy of themselves within other programs. Instead, they make devious use of the program's environment properties of the operating system by placing themselves with the same name ahead of their victim programs on the execution path(Szor 2005).

This causes problems for other programs that block malicious actions of other viruses. – this assume the principles of these blocker programme developed by their authors strictly follow Cohen’s definition of what virus, thus it only looks for viruses that make unwanted changes to the code of another program, as a result will entirely miss out companion viruses(Szor 2005).

Integrity checker programs also rely on the fact that a program’s source code remains unchanged over time. Such programs rely on a database (created at some initial point in time) which assumes to represent a "clean" state of the programs on a machine (Szor 2005). However, as you can see, companion viruses could easily take advantage of this dependency unless the integrity checker also alerted the user about any new application on the system. This approach to counter companion viruses was tested by Cohen’s own system which properly performed this successfully (Szor 2005). Unfortunately, such persistent method would be unpopular with the public. As members do not like to be bothered each time, a new program is introduced on their system. Overall Cohen's approach is definitely the safest technique to use to counter viruses with characteristics of a companion virus(Szor 2005).

In the real world, behaviour-blocking defence systems often alarm in such a situation. For instance, Norton Commander(Bezroukov n.d.), the popular command shell, might be used to

copy the commander's own code to another hard drive or network resource. This action might be confused with self-replicating code, especially if the folder in which the copy is made has a previous which the program would essentially overwrite itself in order to upgrade itself(Szor 2005).

Taking this example into consideration, a more accurate definition of a computer virus would be the following: *"A computer virus is a program that recursively and explicitly copies a possibly evolved version of it."*(Szor 2005)

Computer viruses are self-automated programs that, against the user's wishes, make copies of themselves to spread themselves to new targets(Szor 2005). Although particular computer viruses ask the user with prompts before they infect a machine, such as, "Do you want to infect another program? (Y/N?)," this obviously does not make the program not viruses (Szor 2005). In the end when it comes to trying to classify a particular program as a virus, we need to ask the important question of whether a program is able to replicate itself recursively and explicitly. A program cannot be classed as a computer virus if it dependant of aids to make further copies of itself, such aid can be in the form of modifying the environment of such a program (for example, manually changing bytes in memory or on a disk)(Szor 2005).

2.2. Antivirus

In the early stages of virus detection and removal, computer viruses were easily managed because there were very few viruses at the time (there were fewer than 100 known strains in 1990)(Szor 2005).

Initial antivirus solution development were not difficult, this is shown in the late 1980's and early 1990's where many users were able to create some sort of antivirus program against a particular strain of computer virus(Szor 2005).

Fredrick Cohen showed that antivirus programs couldn't solve the computer virus problem as no single program can detect all future computer virus in finite time(Szor 2005). Despite this, antivirus programs have been quite successful in dealing the computer virus problem for a while. At the same time, other solutions have been researched and developed, however computer antivirus programs are still the most used form of defence against computer viruses at present regardless of their many drawbacks, including the inability to defend and solve the aforementioned problems (Szor 2005).

Often users do not completely understand how to protect themselves against viruses, nor do they know how virus infection prevention can be applied using proper protocols. Unfortunately, negligence is one of the biggest reasons to the spread of computer viruses (Szor 2005). It was said that the sociological aspects of computer security appear to be more relevant than technology. As careless neglect of the most minimal level of computer maintenance, network security configuration, even failure to clean an infected computer allows further problems to occur in other computers (Szor 2005).

In the computer virus research field, computer virus analysis has some common patterns that can be learned easily, depending on efficiency of the analysis process. There are several techniques that computer virus researchers can use in order to acquire a precise understanding of viral programs, which can be utilised to provide appropriate prevention and to provide a response to computer virus outbreaks, which can be controlled to achieve damage limitation(Szor 2005).

Despite what some virus writers may call themselves “experts” in their field, due to creating a code that replicates itself(Gordon 1996). This assumption is far from the truth, in fact several masterminds at various times in the history of computer virus writing represented the pinnacle in the art of computer virus making, such mastermind would go by alias such as: Dark Avenger(Gordon 1993). (Vecna, Jacky Qwerty, Murkry, Sandman, Quantum, Spanska, GriYo, Zombie, roy g biv, and Mental Driller.) All who come from the infamous underground virus writing group 29A (editor note – trustworthy source are difficult to impossible to find).

3. Technical analysis of viruses (Attacker) and antivirus solutions (Defender)

3.1. The Attackers

This section will show that there are two classifications of virus by how the viruses infection strategies we will first talk about the common types of virus that target various file formats and system areas. Afterwards we will discuss another common types of virus that use some form of memory residency strategy that depending on the strategy used can become more virulent than others.

3.1.1. Classification of infection strategies

Boot virus

The first known successful viruses were boot sector viruses [1]. An example of the first boot sector virus is called brain that was created by two Pakistani brothers in 1986 using an IBM PC [1].

Boot Sector viruses take advantage of the boot process of systems, as most computers do not have their operating system (OS) in their read-only memory (ROM), instead they require loading the operating system from a different destination such as local disks or from the network via network cables.

Typically, PC's load the OS from the hard drive, however in older system the boot order could not be defined, thus the machine's default boot location was from the diskette, which was a great entry point for computer viruses to load before the OS. the general concept work by when the ROM-BIOS reads the first sector of the specified boot disk according to the boot order settings in the BIOS setup, stores it in the memory at 0:0x7C00 when successful, and runs the loaded code(Szor 2005).

In newer PC's, there is a record call the master boot record (MBR (that would be located at the root of the first sector of a hard disk. In the MBR there is a general purpose code to locate active boot partitions called a *boot strap loader*(Szor 2005). Because of the use of general-purpose code, a computer virus needs to target only the MBR code to infect the system, and remain in memory depending on the installed operating system. An example comes in the form MS-DOS viruses that can easily remain in memory and infect other inserted media on the fly. A particular famous example of a DOS virus is the Exebug(Szor 2005), which would force the system to load the virus first before completing the other boot processes. Exebug would modify the CMOS setting of the BIOS; in order to trick the system into thinking it has no floppy drives. As a result the system would be force to boot from the infected MBR first, achieved this. Once executed Exebug would check if there is a diskette in drive A: and if there is one, it will load the boot sector of the diskette and transfer control to it. Thus, when the user tries to boot from a boot diskette, the virus would trick the user into believing that the system has booted from the diskette when in reality, it has not(Szor 2005) (Fig 2) shows an example of what the user would see if their system was infected with a boot sector virus.

(Fig 2) Result of Boot Sector Virus

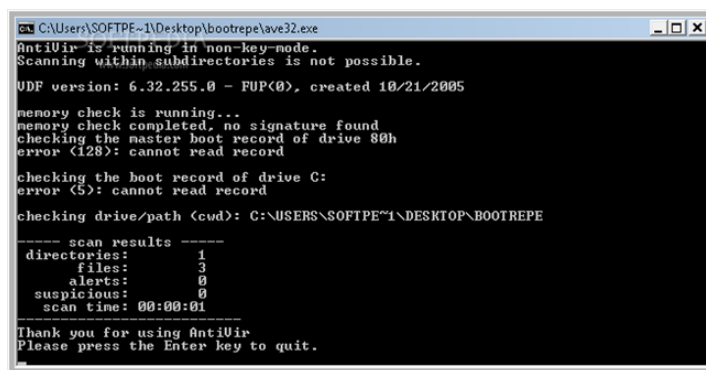
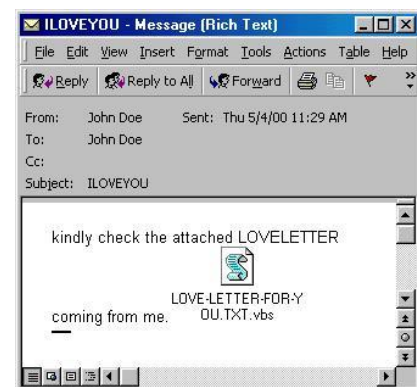


Fig 3.1: how the LoveLetter Virus appears to users.



Overwriting viruses

Some viruses simple locate another file and overwrite it with its own copy, although very primitive it is certainly the easiest approach. These kinds of viruses can be fatal if these simple viruses overwrite files on the entire disk. Because of the nature of these viruses, it is not possible to disinfect the infected files from the system; instead, infected viruses must be deleted and restored from backup. (Fig 3) illustrates what becomes of a program code when an overwriting virus infects it.

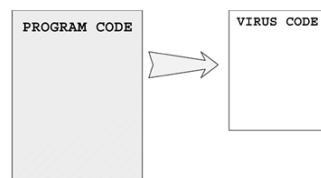


Fig 3 – illustrate a overwritting virus attack

Normally, overwriting viruses are not very successful threats because the effects of these viruses are easily detectable even to the user. However, such viruses have better potential when this technique is combined with network-based propagation. For instance, the VBS/LoveLetter.A@mm(Chien 2002) virus mass mails itself to other computers. When executed, it will overwrite with its own copy any local files with the many extensions such as: .vbs, .vbe, .css, .jpg, .jpeg, .wav, .txt, .gif, .doc etc.

3.1.2. Classification of in-memory strategies

Direct action viruses

5.1. Direct-Action Viruses

Some of the simpler computer viruses do not actively manifest themselves in computer memory. The very first file infector viruses on the IBM PC, such as Virdem and Vienna(Szor 2005), belong to this category.

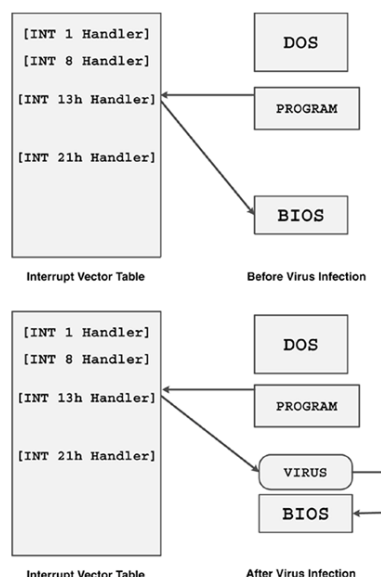
Direct-action viruses load the host program into computer memory. Upon getting control, they look for new objects to infect by searching for new files. As a result, this trait, Direct-action viruses has become one of the most common kinds of computer virus due to relative ease in crafting them by the attacker on a variety of platforms, in binary or in script languages(Szor 2005).

Memory resident viruses

A much more efficient class of computer viruses remains in memory after the initialization of virus code. Such viruses typically follow these steps(Szor 2005):

1. The virus gets control of the system.
2. It allocates a block of memory for its own code.
3. It relocates its code to the allocated block of memory.
4. It activates itself in the allocated memory block.
5. It hooks the execution of the code flow to itself.
6. It infects new files and/or system areas.

Fig 3.2 illustrates how boot sector viruses such as the brain virus install itself via hooking the BIO disk handler



A method that this kind of viruses use, is the manipulation of an operating system's interrupt via hooking(Szor 2005), for example, DOS programs use DOS and BIOS interrupts for system services. In the past, on older computers, programmers typically transferred control to a BIOS-based entry point, so programmers needed to keep such entry points in mind(Szor 2005). The interrupt vector table (IVT) simplifies the programmer's task on the IBM PC for several reasons. By allowing programs can refer to functions by their interrupt number and service number via the IVT, As a result, hard-coded addresses to services need not be compiled into the program code. Instead, the INT x instruction can be used to transfer control to a service via the IVT. (Fig 3.2 shows how the brain virus hooks installs itself by hooking the BIOS disk handler.)

Honourable mentions related to memory resident virus are the *stealth viruses*. Stealth viruses always intercept a single function or set of functions in such a way that the caller of the function will receive manipulated data from the hook routine of the virus. Therefore, computer virus researchers only call virus "stealth" if the virus is active in memory and manipulates the data being returned(Szor 2005).

one of the first-known viruses on the PC, Brain (a boot virus)(Leyden 2006), was already stealth. Brain showed the original boot sector whenever an infected sector was accessed and the virus was active in memory, hooking the disk interrupt handling. This was in the golden days when Alan Solomon (author of one of the most widely used virus-scanning engines)(Szor 2005) was challenged to figure out what exactly was going on in Brain-infected systems.

3.2. The Defenders

This section will discuss different forms of defence against the computer virus problem, from antivirus defence protection, memory scanning and disinfection technique to antivirus defences used to prevent worms and network work intrusions.

3.2.1. Antivirus solutions

1st /2nd gen scanners

Antivirus scanners as simple programs that look for sequences of bytes extracted from computer viruses in files and in memory to detect them. As a result of its simplicity Virus scanner have become one of the most popular methods to detect computer viruses, due to it being reasonably effective(Szor 2005). In modern times state of the art anti-virus software use a lot more complex features to detect complex viruses, which are beyond the capabilities of first gen scanners,

In contrast 2nd generation scanners use *near exact* and *exact* identification alongside other methods which helps to refine the detection of computer viruses and other malicious programs

An approach used by 2nd generations scanners is an approach called *smart scanning*(Szor 2005). Which was introduced about the same time as computer virus mutator kits appeared, these kits were typically worked with assembly source files and tried to insert junk instructions such as the *do-nothing* instructions (NOP)(Szor 2005). Smart scanning skipped these NOP instruction in the host program and did not store these instructions in the virus signature(Szor 2005). Instead, an effort was put into selecting the areas of the virus body that had no references to data or other subroutines, which as a result increased the likelihood of detecting a closely related variant of the virus.

Another method used in 2nd generation scanners is skeleton detection which was invented by Eugene Kaspersky(Kramer & Perlroth 2012) , it proved to be especially useful in detecting macro virus families(Szor 2005). This was achieved by the scanner parsing the macro statements line by line and drops all nonessential statements, as well as aforementioned white spaces. This resulted in a skeleton of the macro body that has only the essential macrocode that commonly appear in macro viruses. The scanner would then use this information to detect the viruses, enhancing variant detection of the same family(Szor 2005).

Nearly exact identification is used to detect computer viruses more accurately. For example, the following secondary string could be selected from offset 0x7CFC in the previous disassembly to detect a Stoned virus nearly exactly(Szor 2005):

```
0700 BA80 00CD 13EB 4990 B903 00BA 0001
```

The scanner can detect a Stoned virus variant if one of the strings is detected and refuses disinfection of the virus because it could be a unknown variant that as a result of improper analysis could possibly lead to disinfecting the variant incorrectly(Szor 2005). Whenever both strings are found, the virus is nearly exactly identified. It could be still a virus variant, but at least the repair of the virus is more likely to be proper(Szor 2005). On a side note 2nd generation scanner could also achieve nearly exact identification without the use of search string of any kind by relying on the cryptographic checksums(Lammer 1990) or some sort of hash function.

Exact identification(Szor 2005) is the only way to guarantee that the scanner precisely identifies virus variants. This method is usually combined with first-generation techniques. Unlike *nearly exact identification*, *exact identification* uses as many ranges as necessary to calculate a checksum of all constant bits of the virus body(Szor 2005). To achieve this level of accuracy, the variable bytes of the virus body must be eliminated to create a map of all constant bytes.

Consider the stoned virus and its code shown in (fig 4), there are two jump instructions that finally lead the execution flow to the real start of virus code, which are displayed before the front of the code at the zero byte of the virus body

Fig 4- illustrates the code of the Stoned virus(Szor 2005)

```
seg000:7CD3 B9 88 01      mov     cx, 108h      ; 440 bytes
seg000:7CD6 0E          push    cs
seg000:7CD7 1F          pop     ds
seg000:7CD8 33 F6      xor     si, si        ; copy virus code to memory
seg000:7CD9 8B FE      mov     di, si
seg000:7CDC FC          cld
seg000:7CDD F3 A4      rep movsb
seg000:7CDF 2E FF 2E 00 00  jmp     dword ptr cs:00h
```

after the second jump(jmp) instructions is where the data area of the virus actual is. The variables are flag, int13off, int13seg, and virusseg. These are the true variables which values can be changed according to

the environment of the virus. The constants are jumpstart, bootoff, and bootseg; as by the definition of a constant, these values will not change, just like the rest of the virus code(Szor 2005).

Once the variable bytes are all identified, there is only one more important item remaining to be checked: the size of the virus code. It can be deduced that a Stoned virus fits in a single sector(Szor 2005); however, the virus copies itself into existing boot and master boot sectors. To find the real virus body size, the scanner need to identify the code that copies the virus to the virus segment, this can be shown in the disassembly in (Fig4.1) , once this final piece of information is found, all the required parts are gathered for the scanner to accurately calculate the map of the virus

```
seg000:7C00      bodyzero:
seg000:7C00 EA 05 00 C0 07  jmp     far ptr 7C0h:5
seg000:7C05 E9 99 00      jmp     start
seg000:7C05      ; -----
seg000:7C08 00      Flag db 0 ; hard disk or diskette?
seg000:7C09 51 02      int13off dw 251h ; DATA XREF: seg000:7CAFjw
seg000:7C0B 00 C8      int13seg dw 0C800h ; DATA XREF: seg000:7CB5jw
seg000:7C0D E4 00      jumpstart dw 0E4h ; offset to make
seg000:7C0D      ; inter segment jump
seg000:7C0F 00 9F      virusseg dw 9F80h ; DATA XREF: seg000:7CC6jw
seg000:7C11 00 7C      bootoff dw 7C80h ; to boot
seg000:7C13 00 00      bootseg dw 0 ; segment
seg000:7C15      ; -----
seg000:7C15 1E      push    ds
seg000:7C16 50      push    ax
```

Fig 4.1 Shows the disassembly of a Stoned virus, and locates the virus body size.

Algorithmic scanning methods

The term algorithmic scanning is a bit misleading but nonetheless widely used(Szor 2005). Whenever the algorithm a scanner cannot deal with a virus, new detection code must be introduced to implement a virus-specific detection algorithm. The result is then called algorithmic scanning, however virus-specific detection algorithm could be a better term(Szor 2005).

Because of these new iteration of algorithmic scanning code, segments of these algorithms contain special detection routines that were hard to port to new platforms, thus stability issues aware a common issue, to resolve this the underlying problem had to be resolved which was the virus scanning language (Szor 2005).

To eliminate this problem, modern algorithmic scanning is implemented as a Java-like p-code (portable code)(Wirth & Weber 1966)(Nori et al. 1975) using a virtual machine. Norton AntiVirus (product of the technology company Symantec) uses this technique. The main advantage of this method is that the detection routines are highly portable. Such that there is no need to port each virus-specific detection routine to new platforms(Szor 2005).

The disadvantage of such scanners is the relatively slow p-code execution, compared to real run-time code, as interpreted code can often be hundreds of times slower than real machine code(Szor 2005). Despite this trait, these routines provide scan functions to look for a group of strings with a single search or convert virtual and physical addresses of executable files.

In the future, it is expected that algorithmic scanners will implement a JIT (just-in-time)(Aycok 2003) system to compile the p-code-based detection routines to real architecture code, similarly to the .NET framework of Microsoft Windows(Szor 2005). For example, when the scanner is executed on an Intel platform, the p-code-based detections are compiled to Intel code on the fly, enhancing the execution speed of the p-code often by more than a hundred times(Szor 2005). This method eliminates the problems of real-code execution on the system as part of the database, and the execution itself remains under control of the scanner because the detection routines consist of managed code(Szor 2005).

Code Emulation

Code emulation is an extremely powerful virus detection technique. A virtual machine is implemented to simulate the CPU and memory management systems to mimic the code execution. As a result, malicious code is simulated in the virtual machine of the scanner, rather than the real processor thus preventing any damage to the actual system executes no actual virus code(Szor 2005).

Integrity Checking

The wide variety of scanning techniques clearly shows how difficult computer virus detection based on the identification of known viruses can be. Thus, it might make sense to use an approach to virus detection that has better control via more generic methods that detect and prevention of changes to the file and other executable objects based on their integrity.

Frederick Cohen(Cohen 1994) demonstrated that integrity checking of computer files is the best generic method. For example, on-demand integrity checkers can calculate the checksums of each file using some known algorithm, such as MD4(Rivest 1991), MD5, or a simple CRC32. even simple CRC algorithms work would effectively by changing the generator polynomial(Szor 2005)(Radai 1992).

On-demand, integrity checkers use a checksum database that is created on a protected system or elsewhere, such as an online location(Szor 2005). This database is used each time the integrity checker is run to see whether any object is new to the system or whether any objects have changed checksums.

3.2.2. Memory Scanning and Disinfection

Memory scanning is essential for all operating systems. This is because after a virus has been executed and is active in memory, it has the potential to hide itself from scanners by using stealth techniques(Szor 2005). Even if the virus does not use a stealth technique, removing the virus from the system becomes more difficult when the virus is active in memory because such a virus can infect previously disinfected objects again and again(Szor 2005). In addition, a file cannot be deleted from the disk as long as it is loaded in memory as a process. Similar to how a Registry key related to a malicious program cannot be deleted if the malicious code puts the same key back into the Windows Registry as soon as the antivirus program removes the keys(Szor 2005).

An important example of memory scanning was in 2001 when the W32/CodeRed Worm followed by W32/Slammer, which used a similar approach. As traditionally, memory scanning would be done on-access however there was a theory by the author, Mikko Hypponen and Ismo Bergroth (Szor 2005) of next generation viruses that would never hit the disk, thus rendering these on-access scanner useless as no file was created upon the execution of these viruses(Szor 2005). It can be said then without memory scanning , such threats cannot be detected by antivirus software, however this statement could be countered with that antivirus solutions were not the right technology but rather, a different technology would be needed such as intrusion prevention(Szor 2005).

Unfortunately memory scanning is subject to several possible attacks, the following illustrates a number of possible attacks and notes some solutions(Szor 2005).

- Encryption is the main problem, even under other operating systems such as DOS. Viruses might decrypt themselves in such a way that only a tiny window of decrypted code is available at a time.
- Attackers can use in-memory polymorphic code to confuse scanners. For example, viruses such as Whale and DarkParanoid(Szor 2005) used this method on DOS, and W32/Elkern(Szor 2005) variants used it on 32-bit Windows systems. Such viruses can be detected only by algorithmic in-memory scanning.
- A worm can make and run multiple copies of itself, each one keeping an eye on its other copies. Alternatively, a single thread is injected into another process that keeps an eye on the worm process. An example of the first scenario is a variant of W32/Chiton(Szor 2005). An example of the second scenario is W32/Lovegate@mm(Szor 2005). (The first variation of this attack is based on the self-protection mechanism of the "Robin Hood and Friar Tuck"(Szor 2005) programs that, according to anecdotes, were developed at Motorola in the mid-1970s(Szor 2005)(Raymond 2003).

3.2.3. Memory Disinfection

A good memory scanner should work closely with an on-access virus scanner and should be always be aware of the same set of viruses that are known by the file scanner components of the antivirus product.

The easiest way to deactivate a virus in memory would be to kill the particular task in which the virus code is detected by the memory scanner. This is can done easily using some TerminateProcess() API along with the appropriate rights(Szor 2005). However this come with some risks, and should use with great case as active virus code is most likely attached to a user application thus import user data can be lost if the infected process were simply killed. With this in mind it is advisable to use TerminateProcess() in situation in which the virus code is active as a separate process such as the WNT/RemEx or W32/Parvo viruses(Szor 2005).

The most difficult case of deactivation comes in the form when the virus code is active as a part of a loaded EXE or DLL image or if the virus allocates pages for itself on a per process basis, which then it hooks some imports of the hosts application to itself(Szor 2005). In these situations, the active code must be patch in memory so that the virus is deactivated. A carefully developed procedure must be implemented, as an incorrect path of the virus could lead to the creation of new variants as a direct result from the memory disinfection itself(Szor 2005).

4. Conclusion

This conclusion will be split in two parts, with each part concluding on the potential future of both computer viruses and antivirus solutions respectively. Afterwards there will be some final words from the author to the reader

Firstly the mentioned viruses are merely a fragment of the over abundant source of computer viruses that have been created in the past and the infinite combinations of various complexities of computer virus that can be created in the future. With that said the trend in which the intention of these computer viruses being used for has shifted from what was once intellectual curiosity and comedy value to much graver possibilities such as aiding in digital crimes such as identity fraud and the destruction of commercial intellectual property. Which agrees with Peter Szor's prediction from 2005 that the payloads of viruses have shifted from being a "joke" to more criminalising payloads (Szor 2005).

On the other hand, antivirus solutions, specifically memory scanning and disinfection techniques are much harder to predict. As these two techniques are very challenging task under NT-based computers, due to the multitasking multithreaded environment. This makes the operating system much more complex than DOS, which therefore makes any infections from computer viruses equally as complex if not more complex because of the virus writer's design. As such with the trend for more advance and ever increase in complexity of personal computers and other systems, ever more complex solution will be required which may lead to more innovative techniques to approach the computer virus problem. However as it stand as said by Dr Alan Solomon (Dunn 2014), antiviruses are not the solution as no single antivirus solution can evolve in such a way that viruses can. Thus, users are left with antivirus solutions that are fine for the current generation however it will always be vulnerable to never generations of different strains of computer viruses that is until the next update, which at that point the game of cat (antivirus) and mouse (virus) will start all over again.

4.1 Final Words

Our journey in computer virus research has come to an end. Unfortunately due there is a lot of material that was not included due to the level of scope that this report provides.

To conclude, this report attempted to offer useful information related to the computer virus problem, with various topic related to this problem being discussed to the best of the author's ability. However, it must be mentioned that due to ubiquitous yet mysterious nature of the computer virus problem, many sources contain many technical errors that are based on "well known facts" and anecdotes unrelated to technical realities. As viruses evolve, the research is expected to evolve, thus may be accepted in the past as "known facts", can be completely invalid in modern times, as a result it's also acceptable to question these "known facts" if one is to come about new realisations, that would ultimately contribute to the evolution of the art of computer virus research.

Thanks for reading.

5. Bibliography

- Aycock, J., 2003. A Brief History of Just-In-Time. *ACM*, pp.97–113. Available at: <http://www.cs.tufts.edu/comp/150IPL/papers/aycock03jit.pdf> [Accessed December 6, 2014].
- Bezroukov, N., The History of Development of Norton Commander. *Web Page*. Available at: http://www.softpanorama.org/OFM/Paradigm/Ch03/norton_commander.shtml [Accessed December 5, 2014].
- Chien, E., 2002. VBS.LoveLetter.Var | Symantec. *Web Page*. Available at: http://www.symantec.com/security_response/writeup.jsp?docid=2000-121815-2258-99 [Accessed December 6, 2014].
- Cohen, F.B., 1994. *A Short Course on Computer Viruses*, Wiley. Available at: http://books.google.co.uk/books/about/A_short_course_on_computer_viruses.html?id=LqNQAAAAMA-AJ&pgis=1 [Accessed December 4, 2014].
- Dunn, J.E., 2014. Anti-virus pioneer Alan Solomon thinks anti-virus is dead. He uses Linux instead - Blogs - Techworld.com. *Tech World*. Available at: <http://www.techworld.com/blogs/war-on-error/antivirus-pioneer-alan-solomon-thinks-antivirus-is-dead-he-uses-linux-instead-3537985/> [Accessed December 7, 2014].
- Gordon, S., 1993. Inside the mind of Dark Avenger. *PERSONAL COMPUTER WORLD*. Available at: <http://molar.crb.ucp.pt/cursos/1%C2%BA e 2%C2%BA ciclos - lics e lics com mests/Inform%C3%A1tica de Gest%C3%A3o/2%C2%BA Semestre/%C3%89tica e Deontologia/Papers para %C3%89tica/The Bulgarian Dark Avenger.pdf> [Accessed December 6, 2014].
- Gordon, S., 1996. The Generic Virus Writer II (VX heaven). *website*. Available at: <http://vxheaven.org/lib/asg04.html> [Accessed December 6, 2014].
- Kramer, A.E. & Perlroth, N., 2012. Cyberweapon Warning From Kaspersky, a Computer Security Expert - NYTimes.com. *The New York Times*. Available at: http://www.nytimes.com/2012/06/04/technology/cyberweapon-warning-from-kaspersky-a-computer-security-expert.html?pagewanted=all&_r=0 [Accessed December 6, 2014].
- Lammer, P., 1990. Cryptographic Checksums - Virus Bulletin. *Virus Bulletin*, pp.10–12. Available at: <https://www.virusbtn.com/pdf/magazine/1990/199010.pdf> [Accessed December 6, 2014].
- Leyden, J., 2006. PC virus celebrates 20th birthday • The Register. *The Register*. Available at: http://www.theregister.co.uk/2006/01/19/pc_virus_at_20/ [Accessed December 6, 2014].
- Nori, K.V. et al., 1975. The Pascal P Compiler Implementation Notes. *ACM*, pp.57–58. Available at: http://www.moorecad.com/standardpascal/The_Pascal_P_Compiler_implementation_notes.pdf [Accessed December 6, 2014].
- Radai, Y., 1992. Checksumming Techniques for Anti-Viral Purposes. In *Education and Society - Information Processing '92, Volume 2, Proceedings of the IFIP 12th World Computer Congress, Madrid, Spain, 7-11 September 1992*. pp. 511–517. Available at: http://www.researchgate.net/publication/221330679_Checksumming_Techniques_for_Anti-Viral_Purposes [Accessed December 6, 2014].
- Raymond, E., 2003. The Meaning of “Hack.” *Web Page*. Available at: <http://catb.org/~esr/jargon/html/meaning-of-hack.html> [Accessed December 6, 2014].

Rivest, R.L., 1991. The MD4 Message Digest Algorithm. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*. Springer-Verlag, pp. 303–311. Available at: <http://portal.acm.org/citation.cfm?id=646755.705223>.

Szor, P., 2005. *The art of computer virus research and defense*, Available at: http://books.google.co.uk/books?hl=en&lr=&id=XE-ddYF6uhYC&oi=fnd&pg=PT3&dq=computer+virus&ots=GgFNinYVG_&sig=ZWCR91EjUxX14v9ZlthGjL-30Ys [Accessed October 19, 2014].

Wirth, N. & Weber, H., 1966. EULER: a generalization of ALGOL, and its formal definition: Part II. *Communications of the ACM*, 9(2), pp.89–99. Available at: <http://dl.acm.org/citation.cfm?id=365170.365202> [Accessed December 6, 2014].