Computer Viruses
A Very Real Threat

Bill Glover – December 2004
bill@billglover.co.uk

Table of Contents

Introduction

Most people have heard of a computer virus, but not many people realise what exactly a computer virus is. This report attempts to clarify the definition of a computer virus and how it can be distinguished from other types of malicious software. The report looks at the ongoing battle between virus writers and their anti virus counterparts. We move on to some of the problems of preventing virus infection in the corporate and home environment. It concludes with a look to the future and what it may hold for the computer virus.

What is a Virus?

It is common amongst computer users to use the term "Computer Virus" to describe any form of malicious software. When something goes wrong, it is not uncommon to hear the phrase, "I've got a Virus". This use of the word Virus, while universally understood is technically inaccurate. The computer virus is only one form of many different types malware (malicious software) that exist on computer systems.

There are various different categories of malware. These are classified based on what they do, how they propagate and how they are executed. Some malware, especially some of the recent Internet worms, seems to blur the boundaries of the definitions and make distinct categorisation difficult. An attempt has been made to define the various types of malware, but it is worth noting that conflicting definitions may exist.

Viruses

The first serious discussion on self replicating programs was presented by Von Neumann in 1949 in the University of Illinois Lectures on the Theory and Organization of Complicated Automata [1]. But it wasn't until 1984 that the term "Computer Virus" was first coined by Fred Cohen in his paper *Experiments with Computer Viruses* [2]. We define a computer virus as a program that can infect other programs by modifying them to include a possibly evolved copy of itself. It is important to note that a virus, just like it's biological counterpart, cannot replicate without a host. A virus must infect a host program and this host program must be transferred to a new machine for the virus to spread.

Worms

In contrast to the virus, the computer worm can propagate without the need for a host program. A worm is a completely self contained program that has the ability to replicate without requiring a user to transfer a host between machines [3 4]. The boundaries between the worm and the virus are gradually being broken down. The MyDoom virus/worm exhibits the characteristics of both a virus and a worm at different points in it's execution. This has led to it being classified differently by several anti-virus companies. Symantec claims it is a worm [i] while CERT claims it is virus [ii] although they initially seemed unsure. As the boundaries between viruses and worms continue to blur we will start to see them lumped under one heading, but for the purists out there, the distinction remains that worms, unlike viruses, do not need a host to propagate.

Trojans

Another common form of malware is the Trojan Horse. The Trojan Horse takes its name from the famous wooden horse used by the Greeks during the Trojan war to trick the Trojans into allowing them into the city of Troy. A Trojan Horse tricks the user into executing it by pretending to be a legitimate program. The user unwittingly executes what they believe to be a harmless program only to unleash whatever payload is being carried by the Trojan. Trojan Horses are unable to replicate, but they can be used to trigger a virus or even a worm.

Hoaxes

A virus hoax is as its name suggests, a trick. More often than not, hoaxes come in the form of an email purporting to be from a reputable source. They usually claim in rather over anxious terms, heavily punctuated with lots of exclamation marks, that a new undetectable virus has been discovered and that the message has to be sent to as many people as possible to ensure that everyone is aware of the new threat. A typical example of such a hoax is the Aids Virus Hoax [iii]. The Aids hoax warned of a virus that would "destroy your memory, sound card and speakers, drive and it will infect your mouse or pointing device as well as your keyboards (possibly motherboards) making what you type not able to register on the screen".

There are many such hoaxes sent out but users, oblivious to the fact that they are passing on false information, readily forward these hoax emails to everyone in their address books. Hoaxes have the potential for causing as much damage as a real virus. The increased network traffic caused by people forwarding virus hoaxes can easily slow down or overload a system. Worse still, a hoax-like email could be used to actually transmit a virus around. Hoaxes, while not harmful in themselves, use the threat of computer viruses to try and spread as far as possible.

Other Malware

There are several further classifications of malware that have not been mentioned here, but already it is easy to see the attraction in lumping them all under the common title of "A Computer Virus". the distinction between different types of malware is often hard to make and to the end user is, more often than not, unimportant. But the distinction is of great importance when it comes to detection, prevention and removal.

A Pseudo Code Virus

```
program virus := {
virusID;

subroutine infec-executable := {
    loop: file = randomExecutable;
    if (firstLineOFFile == virusID) then goto loop;
    prepend virus to file;
}

subroutine do-damage := {
    do damage;
}

subroutine trigger-pulled := {
    return true on desired conditions;
}

main-program := {
    infect-executable;
    if (trigger-pulled) then do-damage;
    goto next;
}

next:
}
```

This simple pseudo code virus [2] gives a clear indication of the key components of a virus. There are three main routines that make up this simplified virus model. There is an infection routine which contains all the code required for the virus to replicate. There is the damage, or payload, routine. The payload is normally independent of the infection routine and dictates the effect the virus has on a system. The trigger routine defines the circumstances necessary for the payload to be executed. This allows the virus writer to determine the precise moment at which damage will be inflicted. All three of these main components are discussed later.

The Growth of Viruses

Although there is some debate as to the first computer virus, Elk Cloner [iv] is widely credited with being the first virus seen in the wild. Elk Cloner was released around 1982 and was written by a 15 year old boy named Richard Skrenta. It was designed to infect Apple II computer systems. The virus was not designed to inflict any damage. It merely copied itself to the boot sector on all floppy disks used on an infected system and displayed a short poem every 50th time it was executed.

The first PC virus was not released until 1986. It came in the form of the Brain [v] virus from two brothers in Pakistan. Now extinct, the Brain virus was thought to be an advert for the computing business run by the two brothers. The original variants of the Brain virus were harmless and simply changed the volume label of the hard disk to "(C) Brain".

Viruses first started to earn their place in the media during 1998. Computer viruses featured in articles in several popular newspapers and magazines including Time, News Week and Working Woman [5]. Initially the media tended to predict the end of the world every time a successful virus was released. But in recent years the media has tended to be more informative in its reporting rather than try and spread panic. See [6] for an example of a recent article on BBC News Online regarding the Sober-I Virus [vi].

In the early 1990s several anti virus scanners were released. People started to realise that there was big money in the anti virus industry. The competitive nature of the anti virus industry led to the takeover of many of the smaller firms to form a much shorter list of key anti virus corporations.

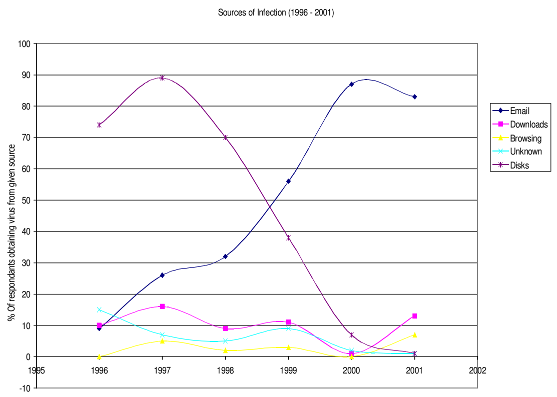
1995 saw a new twist in the history of the computer virus. The first macro virus was found in the wild. The virus was named Concept [vii]. Concept exploited vulnerabilities in the scripting language that was part of Microsoft's Office suite to create a virus that would infect and spread as part of a MS Word Document. While Concept was merely released to show what could be done with MS Word Macros, a descendant of Concept was far more devastating. Melissa [iix] took the Concept virus one stage further. It would email a copy of the infected document to the first 50 people in the MS Outlook address book and then infect the normal.dot template so that all other word documents would become infected. Viruses were now no longer restricted to executable files.

Until 1999 it was common to advise people that they were safe from email viruses as long as they did not open infected attachments. The BubbleBoy [ix] virus proved this advice to be incorrect. BubbleBoy was the first virus that could execute when the user simply read the email. It was only effective on Microsoft email clients but was proof that virus writing was not standing still.

Viruses have continued to grow in complexity in an ongoing battle to circumvent anti virus software. Viruses are starting to include several replication strategies to ensure they reach as many machines as possible. It is not uncommon for a virus to include it's own email server or ftp client to ensure that hardening existing clients does not stem the speed of infection.

Transmission of Viruses

In their infancy computer viruses were traditionally slow to spread between machines. They required a user to place a disk in an infected machine and physically transfer it to a new machine where the infection could continue to spread. Viruses could only spread as fast as users moved information between computers. Over their relatively short life time, viruses have improved both in the speed of propagation and in complexity of the code [4]. The rapid growth of the Internet as a communications medium has changed the way viruses propagate. Viruses no longer need to be physically carried between machines as they can use high speed computer networks to do the same job.



A survey done in 2001 asked users to identify how they were infected with their most recent virus [7]. The trend clearly indicates that email has replaced disks as the medium of transport for viruses. Other sources of infection have remained pretty much constant over the same time period.

One of the fastest growing viruses released to date was Nimda [x]. It managed to spread worldwide in just half an hour. But astonishingly the title of the fastest growing virus of all time [8] goes to MyDoom [xi]. This is astonishing for the simple reason that MyDoom required users to open an infected email attachment. Users have still not picked up on one of the essential messages in the fight against viruses: Never open attachments you don't know to be safe. Viruses that spread this quickly are thankfully few and far between, but when they are released the consequences can be devastating.

The Trigger

The nature of a virus is determined just as much by the trigger function as it is by the replication and the payload. The introduction of a trigger mechanism allowed virus writers to delay the destructive effects of their viruses. By delaying the damage, a virus writer can ensure that a virus has been given maximum chance to spread unnoticed. This ensures the damage caused by the payload is as extensive as possible. This works because users are simply unlikely to notice anything is amiss if the payload does not execute.

Delayed triggering allows infected files to work their way into scheduled backups. This ensures that when users restore infected or damaged files from their backups, they are also restoring the virus. The fist virus to employ this technique was Dark Avenger.1800 [xii] released in 1989.

A more recent use of the triggering mechanism is to synchronise the launch of a Distributed Denial of Service attack on some computer system or website. By launching an attack simultaneously from millions of different computers the effects are likely to be devastating.

The trigger does not have to be date dependant. A payload could be triggered on any number of events including, program execution, a fixed number of infections, key combinations and of course a calendar date.

The Payload

Although strictly not required under the definition of the virus given earlier, most viruses carry some form of payload in addition to the ability to replicate. The payload is some code that is executed outside of the normal replication process. It is the actions of the payload that most computer users associate with being a computer virus. When asked to describe a virus, most people would list some of the behaviour of a typical payload. This misunderstanding of the difference between a payload and a virus is not unreasonable since it is the actions of the payload that most users actually experience.

Payloads come in varying forms with dramatically different effects and consequences. Some payloads are harmless and classed as a mere annoyance while others are highly destructive. Some could even be said to be beneficial.

The Simile.D [xiii] virus is notoriously hard to detect due to its highly complex replication engine but its payload is relatively simple [9]. If the date is March 17 or September 17 (May 17 on Linux) then the payload simply displays a message box with a short message to the user. The payload is more of an irritation than a catastrophic nightmare. This is in contrast to the catastrophic effects of a virus such as CIH [xiv]. The CIH virus contains two payloads, both activated on the 26th of each month. The first payload overwrites all sectors on the hard disk. This is destructive enough in itself, but the second payload tries to take destruction to a new level. It tries to damage the computer hardware by modifying the computer's BIOS.

Today viruses and worms tend to carry payloads with a more global view of destruction. Virus writers tend to use payloads which create a back door into the infected system allowing them to use that system to launch further attacks. Another trend is to use virus payloads to launch distributed denial of service attacks against one or more websites [4]. Microsoft and the White House are two high profile targets of such DDoS attacks launched from the payload of a virus. The Microsoft Windows Update site was attacked by the MsBlast payload [xv] and the White House was attacked by CodeRed [x].

The release of CodeRed into the wild had such a devastating effect that it saw a radical new approach by one company to counter it. HP Labs used the same exploit as CodeRed but replaced the payload with a program that cleaned infected systems. They claimed that the virus spread so quickly that the only way to try and clean systems was to use the same replication method as the virus itself [10]. The concept of a good virus opens up a whole new discussion not covered here, but the CodeRed virus and the HP-fix demonstrates how important the payload is. Two viruses can replicate and spread in exactly the same manner but have completely different payloads.

The Anti Virus

In order to understand how we can go about protecting a system from the effects of computer viruses we need to examine some of the methods used by virus writers to avoid detection.

Before the days of sophisticated anti virus software, viruses often had little or no defence against detection. Early viruses started off by ensuring that the last modified date of an infected file was left unchanged after the infection process. Viruses then started to infect files without changing the size of the file. This is only possible because most files contain unused space within the executable. If a virus can overwrite these unused spaces it can then embed itself within the file without changing the file size. Both these techniques are not enough to make a virus undetectable but they do ensure that a casual glance at a directory listing will not give any clues as to the presence of a virus.

Stealth

As the complexity of viruses increased, increasing emphasis was placed on ensuring that an individual virus remained undetected for as long as possible. The first stealth viruses started to appear in the wild. As their name suggests, stealth viruses try and hide their existence from a user or anti virus program. When a request is made to scan or read an infected file, the virus intercepts the request before it reaches the operating system and returns a copy of the uninfected file. The file then appears clean when in fact the copy sitting on the disk is actually infected with the virus. In order that the virus can intercept this request to read a file, it has to be resident in memory at the time of the request. Such stealth tactics are avoided by ensuring that there are no memory resident viruses at the time of scanning. The easiest way to ensure this is to boot from a known clean boot disk.

Self Modification

Early anti virus programs used to search for a virus by looking for patterns of commands that would uniquely identify a virus. These tell tale commands were known as a virus fingerprint. It wasn't long before virus writers came up with a new trick to prevent detection. Self modifying viruses started to appear. Each replication of the virus had now become a mutation. This had implications for the anti virus writers in that each virus could now have any number of fingerprints. The first self-modifying viruses simply swapped subroutines around or inserted random commands between two known points to try and destroy its fingerprint. These self modifying viruses didn't change much from their original form and anti virus writers soon learnt to give each virus multiple signatures or to simply ignore the random statements.

```
virusStatement1;
virusStatement2;
goto    $ + 5;
   randomCommand;     // These commands are never executed and so
   randomCommand;     // they can be replaced with random commands
   randomCommand;     // to disrupt the signature.
   randomCommand;
virusStatement3;
```

Encryption

The self encrypting virus tried to take self modification to the next level. With each mutation, the virus would encrypt itself with a random key. Now the entire virus code was modified at each mutation into any number of possible forms. However the virus had to be able to decrypt itself in order to function and so every encrypted virus had to have an unencrypted decryption engine. This decryption engine would remain constant with each mutation. The fingerprint of the decryption engine was found in every infected file and so it could be used to indicate the presence of a virus. It was still possible for anti virus programs to locate viruses even when they were encrypted with a random key.

Polymorphism

The first real threat to anti virus software designers came with the invention of polymorphic code. A polymorphic virus combines the concept of self modifying code with the self-encrypting virus. The main body of the virus is encrypted with a random key and then the decryption engine is modified so that no part of the code remains constant between mutations. This meant that each virus could have any number of fingerprints. A new approach was needed to keep anti virus software one step ahead of viruses.

Heuristics

As the rate at which new viruses rocketed, it became increasingly difficult for anti virus writers to ensure their fingerprint databases were up to date. The databases could only be updated with the fingerprint of a new virus after it had been discovered and analysed. With polymorphic engines ensuring that each virus had any number of possible fingerprints the size of these databases started to increase rapidly. There was no garuantee that anti virus packages could detect all known viruses by simply searching their database of known fingerprints. This, coupled with the dramatic increase in the speed of replication, meant that it was no longer possible to update anti virus software even if the fix was discovered at the same time as the virus was released. A new form of anti virus software was desperately needed [11]. What was needed was a computer algorithm that could detect any virus by simply examining its behaviour. Unfortunately, Fred Cohen proved that such an algorithm was an impossibility [2].

```
// This program is called virus.exe

if(isVirus('virus.exe'){
    originalProgram();
} else {
    replicate();
    doVirusStuff();
    originalProgram();
}
```

This simple circular proof shows that for any algorithm `isVirus()` it is possible to write a virus as shown that defeats the algorithm. It is not possible to write and algorithm that will guarantee the detection of all viruses. Despite this proven impossibility it did not stop anti virus researchers trying to produce a detection method that came close to this holy grail of the anti virus community.

A heuristic approach is adopted to try and get round the unsolvable nature of the problem. A heuristic approach effectively relaxes the requirements of the original algorithm. In this case, the ideal algorithm would detect all viruses without any false positive detections. A heuristic algorithm would detect *most* viruses with a *minimal* number of false positives.

In a simplified heuristic approach two sets of rules are developed. One set of rules can be referred to as the positive rule set and the other, the negative rule set. The positive rule set is a list of rules that define virus-like behaviour. The negative rule set defines rules that indicate non virus-like activity. A simple example [12] is explored below.

Positive Rule Set:

- Very soon after it is run, a program jumps to somewhere near the end
- The program replaces the first few bytes of its image in memory
- It searches for executable files on disk
- Reads a few bytes from the beginning of a program file
- Overwrites the beginning and end of a program file

Using the simple rule set above we can look at the process of identifying virus-like programs. When an infected program is run, execution jumps to near the end of the file. This is a result of the infection process placing the bulk of the virus code at the end of a file. The virus then overwrites the infected sections of the file in the memory image to stealth it from detection by anti virus software. As the virus starts to try and spread, it searches the disk looking for other executable files to infect. All of these actions satisfy the rules for virus-like activity defined in the positive rule set. The heuristic scanner would then positively identify the program as infected.

If the same rule set is considered when looking at the actions of an anti virus scanner we start to see some of the problems associated with the heuristic approach. An anti virus scanner actually exhibits several of the characteristics associated with a virus. It begins it's execution by scanning code in memory to check for memory resident viruses. It then starts to scan the disk for executable files, opening each one in turn and reading the first few bytes of a file looking for tell tale virus signatures. This triggers some of the rules listed under the positive rule set. The program which is actually an anti virus scanner could easily be mistaken for an infected file using the positive rule set alone. This is known as a *false positive* detection. In order to reduce the number of false positives the negative rule set is used to try and eliminate programs that are not viruses themselves but do show some of the characteristics of a virus.

Negative Rule Set

- A program requires or accepts command line arguments
- The program prompts user for action during it's execution
- The program displays information for the user to see

An anti virus scanner would typically prompt the user before taking any action. It usually keeps the user informed of progress through the scan by providing constant feedback. It is also more than likely that the anti virus program requires or at least accepts command line arguments to change the nature of the scan. These are all unlikely to be properties of a real virus they are listed in the negative rule set. The heuristic scanner would then identify the anti virus program as unlikely to be a virus and not produce the false positive we saw before.

The rule sets used in this simple heuristic example are tiny. Anti virus vendors have spent considerable time and money on defining their heuristic rule sets to try and optimise the trade off between the best detection rate and the minimal number of false positives. The heuristic approach to virus detection allows anti virus software to detect some new and previously unseen viruses but it does introduce the concept of the false positive.

The Problem of False Positives

We have already seen how heuristic scanning introduces the concept of false positives to virus detection, but these false positives introduce a greater problem than is immediately obvious. When asked what they want from a virus scanner users will without a doubt demand that the virus checker will identify 100% of viruses without any false positives [12]. Anti virus software has to balance the number of false positives against the successful detection rate. It is not uncommon for anti virus products to be judged on the number of viruses they can detect. Products are marketed on their detection rates. The higher the detection rate, the better the product will sell. However simply shifting the balance of the heuristic scanner to favour detection at all costs has its downsides.

When a clean program is identified as being a possible virus the first reaction of the user is panic. They have a virus on their system. "Viruses are bad." Drastic action is taken and the user could possibly loose data. This is not beneficial to the user and does not instil confidence in the anti virus product. If users don't have confidence in a product then they are unlikely to continue to purchase it.

A second consequence of false positives is significantly more serious. When a legitimate program is identified as a virus, the user is immediately doubtful of the program. This can result in significant mistrust and loss of business for the vendor of the legitimate program. In some cases legal action has been taken against ant virus companies. The case of Imageline vs McAfee Associates was settled out of court and no details made public [12 13]. The potential however exists for significant legal challenges.

As a solution to the problem of false positives, modern virus packages allow you to adjust the balance of the heuristic scanners. This usually comes in the form of a slider that allows you to change the "paranoia" level of the scanner. The improvement of the heuristic rule sets is an ongoing challenge for anti virus vendors.

Virus Prevention

With the threat of infection from a computer virus growing ever stronger, the need for some form of defence from attack is paramount. While the only way to guarantee complete immunity from infection is to prevent the sharing of information [2], this is often counter productive. The reason computers, email and the Internet are so popular is the relative ease with which they allow people to share vast volumes of information. Removing this ability to share information is effectively removing the one reason people rely on computers and the Internet.

There are two main computing environments we need to consider when trying to prevent viruses. There is the corporate environment where computing resources are centrally managed by skilled personel and there is the home environment where anyone and everyone is in charge of the family PC. In some cases different approaches are needed in these two unique environments but there are some measures that could be applied to both.

The first measure in the defence against computer viruses is user education. We see that users still have not got the message that opening strange email attachments is a major security risk [8]. Users need to be informed of how serious computer virus infections can be. The belief that "It won't happen to me" is a luxury that no user can afford. The damage viruses cause extends way beyond the loss of a few hours work.

To improve the security of the Home computing environment there are several steps that could be taken over and above simple user education. Computer software is becoming overly complex. The average home user does not require their email or word processing packages to have scripting functionality. Computer software has traditionally been sold with a wide range of advanced features available by default. Advanced networking, file sharing, scripting, web servers are all some of the things enabled on the average home computer. There is no need for these features to be made available from the word go. If users need to use such features then they can enable them after they have been informed of the risks. Security from the word go should become a priority. Microsoft have made an attempt to do this with SP2 for their Windows XP operating system but more still needs to be done to ensure that an out of the box Windows install is highly secure against even the most uneducated of users. As of August 2004 the average time it took for a clean, un-patched installation of Windows to become infected is around 16 minutes [14]. This is not even long enough to download the patches to secure the system.

In a corporate environment security policies can be forced upon users by system administrators. While this is no substitute for user education it is widely becoming necessary to ensure the corporate computing networks stay secure. Email attachments can be blocked at the server, firewalls can be installed at the barrier with the outside world and update policies can be automated. However the reality is that systems remain un-patched and new exploits are discovered almost daily. When a patch is released it has to be rolled out to every single desktop in the corporation. With OS updates coming out monthly and new virus updates released weekly it is no wonder that a large number of corporate IT systems remain un-patched.

The Future

We are unable to predict the future, but we can speculate as to the next moves of computer virus writers. The high popularity of P2P file sharing programs such as bittorrent, winmx and direct connect opens up a whole new world to virus writers. People are freely exchanging software, music and movies, most of it illegal. There is no guarantee as to the source of the software and people are prepared to risk being infected with a virus to get the use of some cracked software package they have downloaded. The full potential of P2P as a virus distribution mechanism has yet to be tapped. But it could have devastating consequences when it is.

Instant messaging is becoming increasingly popular. With an ongoing battle by IM client vendors to capture the greatest market share the pressure is on to add an increasing number of features to what was once a simple program. File transfer (with the ability to automatically accept incoming files) is a common feature. Users are likely to trust anything they receive from people on their buddy lists as they usually "know" the person who sent the file [15]. This is another, as yet untapped, virus distribution medium.

Evidence has started to surface that virus writers are starting to target the mobile phone industry. There are no known mobile phone viruses in the wild, but but the Cabir [xvi] virus has been shown to infect phones using the Symbian operating system. The virus spreads by searching for other vulnerable phones using the Bluetooth network [16]. The threat is so credible that Nokia even include a virus checker in some of their latest handsets. With most modern handsets supporting Bluetooth technology the possibility for infection exists.

Computer viruses look like they are here to stay. They may not stick around in their current form and the current technology we use to defend against them will probably be inadequate against the next generation, but the threat remains constant. The best form of protection is user education. Users need to be aware of the threat. If users are not aware of the threat that viruses pose then there is little we can do to prevent the spread of malicious code. It is only after a user is aware of the problem that anti virus software can become really effective.

Bibliography

References:

[1] Von Neumann, J. (1949) - Theory of Self-Reproducing Automata

[2] Cohen, Fred (1984) – Experiments with Computer Viruses

[3] Wikipedia – Malware

[4] Kee, Rich (2002) – Evolution of the Computer Virus

[5] Wells, Joe (1996) - Timeline - IBM Antivirus Research (http://www.research.ibm.com/antivirus/timeline.htm)

[6] BBC News Online (2004) - Lazarus-like virus hits computers (http://news.bbc.co.uk/1/hi/technology/4026541.stm)

[7] ICSA Labs Virus Prevalence Survey (2001)

[8] Munro, Jay (2004) - MyDoom.A:Fastest Spreading Virus in History (http://www.pcmag.com/article2/0,1759,1485719,00.asp)

[9] Schreiner, Keri (2002) – New Viruses Up the Stakes on Old tricks

[10] Norman, Andy & Williamson, Matthew (2002) – Hitting Back at CodeRed

[11] Leyden, John – The Register (2002) - The trouble with anti-virus

[12] Gryaznov, Dmitry (1999) - Scanners of the Year 2000 – Heuristics

[13] Bontchev, Vesselin (1997) – Future Trends in Virus Writing

[14] SANS Internet Storm Center (2004) - Survival Time History (http://isc.sans.org/survivalhistory.php)

[15] Williamson, Matthew; Parry, Alan & Byde, Andrew (2004) - Virus Throttling for Instant Messaging

[16] Rohde, Laura (2004) – Nokia Phone Gets Virus Protection (http://www.pcworld.com/news/article/0,aid,117904,pg,1,RSS,RSS,00.asp)

Viruses:

[i] W32.Mydoom.A@mm - *Symantec Security* Response
(http://securityresponse.symantec.com/avcenter/venc/data/w32.mydoom.a@mm.html)

[ii] W32/Novarg.A - *CERT Incident Note*
(http://www.cert.org/incident_notes/IN-2004-01.html)

[iii] Aids Virus Hoax – *Symantec Security Response*
(http://www.symantec.com/avcenter/venc/data/aids.virus.hoax.html)

[iv] Elk Cloner - *The program with a personality*
(http://www.skrenta.com/cloner/)

[v] Brain - *F-Secure Virus Descriptions*
(http://www.f-secure.com/v-descs/brain.shtml)

[vi] W32.Sober.I@mm – *Symantec Security Response*
(http://securityresponse.symantec.com/avcenter/venc/data/w32.sober.i@mm.html)

[vii] WM.Concept – *Symantec Security Response*
(http://securityresponse.symantec.com/avcenter/venc/data/wm.concept.html)

[iix] W97.Melissa.A – *Symantec Security Response*
(http://www.symantec.com/avcenter/venc/data/mailissa.html)

[ix] VBS/BubbleBoy FAQ – *Sophos*
(http://www.sophos.com/virusinfo/articles/bubbleboy.html)

[x] CodeRed – *Symantec Security Response*
(http://securityresponse.symantec.com/avcenter/venc/data/codered.worm.html)

[xi] W32.Mydoom.M@mm – *Symantec Security Response*
(http://securityresponse.symantec.com/avcenter/venc/data/w32.mydoom.m@mm.html)

[xii] DarkAvenger.1800 – *McAfee Virus Information*
(http://vil.nai.com/vil/content/v_406.htm)

[xiii] {Win32,Linux}/Simile.D – *Symantec Security Response*
(http://securityresponse.symantec.com/avcenter/venc/data/linux.simile.html)

[xiv] W95.CIH – *Symantec Security Response*
(http://www.symantec.com/avcenter/venc/data/cih.html)

[xv] WORM_MSBLAST.A – *Trend Micro Security Response*
(http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_MSBLAST.A&VSect=T)

[xvi] Cabir – *Fsecure Virus Descriptions*
(http://www.f-secure.com/v-descs/cabir.shtml)