SCANNERS OF THE YEAR 2000: HEURISTICS

Dmitry O. Gryaznov

S&S International Plc, Alton House, Gatehouse Way, Aylesbury, Bucks, HP19 3XU, UK
Tel +44 1296 318700 · Fax +44 1296 318777 · Email grdo@sands.co.uk

INTRODUCTION

At the beginning of 1994, the number of known MS-DOS viruses was estimated at around 3,000. One year later, in January 1995, the number of viruses was estimated at about 6,000. By the time this paper was written (July 1995), the number of known viruses exceeded 7,000. Several anti-virus experts expect this number to reach 10,000 by the end of the year 1995. This large number of viruses, which keeps growing fast, is known as the glut and it does cause problems to anti-virus software – especially to scanners.

Today, scanners are the most frequently used type of anti-virus software. The fast-growing number of viruses means that scanners should be updated frequently enough to cover new viruses. Also, as the number of viruses grows, so does the size of the scanner or its database, and in some implementations the scanning speed suffers.

It was always very tempting to find a final solution to the problem; to create a generic scanner which can detect new viruses automatically without the need to update its code and/or database. Unfortunately, as proven by Fred Cohen, the problem of distinguishing a virus from a non-virus program is algorithmically unsolvable as a general rule.

Nevertheless, some generic detection is still possible, based on analysing a program for features typical or not typical of viruses. The set of features, possibly together with a set of rules, is known as heuristics. Today, more and more anti-virus software developers are looking towards heuristical analysis as at least a partial solution to the glut problem.

Working at the Virus Lab, *S&S International Plc*, the author is also carrying out a research project on heuristic analysis. The article explains what heuristics are. Positive and negative heuristics are introduced and some practical heuristics are represented. Different approaches to a heuristical program analysis are discussed and the problem of false alarms is explained and discussed. Several well-known scanners employing heuristics are compared (without naming the scanners) both virus detection and false alarms rate.

1 WHY SCANNERS?

If you are following computer virus-related publications, such as the proceedings of anti-virus conferences, magazine reviews, anti-virus software manufacturers' press releases, you read and hear mainly 'scanners, scanners, scanners'. The average user might even get the impression that there is no anti-virus software other than scanners. This is not true. There are other methods of fighting computer viruses – but they are not

as popular or as well known as scanners; and anti-virus packages based on non-scanner technology do not sell well. Sometimes people who are trying to promote non-scanner based anti-virus software even come to the conclusion that there must be some kind of an international plot of popular anti-virus scanner producers. Why is this? Let us briefly discuss existing types of anti-virus software. Those interested in more detailed discussion and comparison of different types of anti-virus software can find it in [*Bontchev1*], for example.

1.1 SCANNERS

So, what is a scanner? Simply put, a scanner is a program which searches files and disk sectors for byte sequences specific to this or that known virus. Those byte sequences are often called *virus signatures*. There are many different ways to implement a scanning technique; from the so-called 'dumb' or 'grunt' scanning of the whole file, to sophisticated virus-specific methods of deciding which particular part of the file should be compared to a virus signature. Nevertheless, one thing is common to all scanners: they detect only *known* viruses. That is, viruses which were disassembled or analysed and from which virus signatures unique to a specific virus were selected. In most cases, a scanner cannot detect a brand new virus until the virus is passed to the scanner developer, who then extracts an appropriate virus signature and updates the scanner. This all takes time – and new viruses appear virtually every day. This means that scanners have to be updated frequently to provide adequate anti-virus protection. A version of a scanner which was very good six months ago might be no good today if you have been hit by just one of the several thousand new viruses which have appeared since that version was released.

So, are there any other ways to detect viruses? Are there any other anti-virus programs which do not depend so heavily on certain virus signatures and thus might be able to detect even new viruses? The answer is yes, there are: *integrity checkers* and *behaviour blockers (monitors)*. These types of anti-virus software are almost as old as scanners, and have been known to specialists for ages. Why then are they not used as widely as scanners?

1.2 BEHAVIOUR BLOCKERS

A behaviour blocker (or a monitor) is a memory-resident (TSR) program which monitors system activity and looks for virus-like behaviour. In order to replicate, a virus needs to create a copy of itself. Most often, viruses modify existing executable files to achieve this. So, in most cases, behaviour blockers try to intercept system requests which lead to modifying executable files. When such a suspicious request is intercepted, a behaviour blocker, typically, alerts a user and, based on the user's decision, can prohibit such a request from being executed. This way, a behaviour blocker does not depend on detailed analysis of a particular virus. Unlike a scanner, a behaviour blocker does not need to know what a new virus looks like to catch it.

Unfortunately, it is not that easy to block all the virus activity. Some viruses use very effective and sophisticated techniques, such as tunnelling, to bypass behaviour blockers. Even worse, some legitimate programs use virus-like methods which could trigger a behaviour blocker. For example, an install or setup utility is often modifying executable files. So, when a behaviour blocker is triggered by such a utility, it's up to the user to decide whether it is a virus or not – and this is often a tough choice: you would not assume that all users are anti-virus experts, would you?

But even an ideal behaviour blocker (there is no such thing in our real world, mind you!), which never triggers on a legitimate program and never misses a real virus, still has a major flaw. To enable a behaviour blocker to detect a virus, the virus must be run on a computer. Not to mention the fact that virtually any user would reject the very idea of running a virus on his/her computer, by the time a behaviour blocker catches the virus attempting to modify executable files, the virus could have triggered and destroyed some of your valuable data files, for example.

1.3 INTEGRITY CHECKERS

An integrity checker is a program which should be run periodically (say, once a day) to detect all the changes made to your files and disks. This means that, when an integrity checker is first installed on your system, you need to run it to create a database of all the files on your system. During subsequent runs, the integrity checker compares files on your system to the data stored in the database, and detects any changes made to the files. Since all viruses modify either files or system areas of disks in order to replicate, a good integrity checker should be able to spot such changes and alert the user. Unlike a behaviour blocker, it is much more difficult for a virus to bypass an integrity checker, provided you run your integrity checker in a virus clean environment – e.g. having booted your PC from a known virus-free system diskette.

But again, as in the case of behaviour blockers, there are many possible situations when the user's expertise is necessary to decide whether changes detected are the result of virus activity. Again, if you run an install or setup utility, this normally results in modifications to your files which can trigger an integrity checker. That is, every time you install new software on your system, you have to tell your integrity checker to register these new files in its database.

Also, there is a special type of virus, aimed specifically at integrity checkers – so-called *slow infectors*. A slow infector only infects objects which are about to be modified anyway; e.g. as a new file being created by a compiler. An integrity checker will add this new file to its database to watch its further changes. But in the case of a slow infector, the file added to the database is infected already!

Even if integrity checkers were free of the above drawbacks, there still would be a major flaw. That is, an integrity checker can alert you only **after** a virus has run and modified your files. As in the example given while discussing behaviour blockers, this might be well too late...

1.4 THAT'S WHY SCANNERS!

So, the main drawbacks of both behaviour blockers and integrity checkers, which prevent them from being widely used by an average user, are:

1. Both behaviour blockers and integrity checkers, by their very nature, can detect a virus only **after** you have run an infected program on your computer, and the virus has started its replication routine. By this time it might be too late – many viruses can trigger and switch to destructive mode **before** they make any attempts to replicate. It's somewhat like deciding to find out whether these beautiful yet unknown berries are poisonous by eating them and watching the results. Gosh! You would be lucky to get away with just dyspepsia!
2. Often enough, the burden to decide whether it is a virus or not is transferred to the user. It's as if your doctor leaves **you** to decide whether your dyspepsia is simply because the berries were not ripe enough, or it is the first sign of deadly poisoning, and you'll be dead in few hours if you don't take an antidote immediately. Tough choice!

On the contrary, a scanner can and should be used to detect viruses **before** an infected program has a chance to be executed. That is, by scanning the incoming software prior to installing it on your system, a scanner tells you whether it is safe to proceed with the installation. Continuing our berries analogy, it's like having a portable automated poisonous plants detector, which quickly checks the berries against its database of known plants, and tells you whether or not it is safe to eat the berries.

But what if the berries are not in the database of your portable detector? What if it is a brand new species? What if a software package you are about to install is infected with a new, very dangerous virus unknown to your scanner? Relying on your scanner only, you might find yourself in big trouble. This is where behaviour blockers and integrity checkers might be helpful. It's still better to detect the virus while it's trying to infect

your system, or even after it has infected but before it destroys your valuable data. So, the best anti-virus strategy would include all three types of anti-virus software:

- a scanner to ensure the new software is free of at least known viruses **before** you run the software
- a behaviour blocker to catch the virus **while** it is trying to infect your system
- an integrity checker to detect infected files **after** the virus has propagated to your system but not yet triggered.

As you can see, the scanners are the first and the most simply implemented line of anti-virus defence. Moreover, most people have scanners as **the only** line of defence.

2 WHY HEURISTICS?

2.1 GLUT PROBLEM

As mentioned above, the main drawback of scanners is that they can detect only **known** computer viruses. Six or seven years ago, this was not a big deal. New viruses appeared rarely. Anti-virus researchers were literally hunting for new viruses, spending weeks and months tracking down rumours and random reports about a new virus to include its detection in their scanners. It was probably during these times that a most nasty computer virus-related myth was born that anti-virus people develop viruses themselves to force users to buy their products and profit this way. Some people believe this myth even today. Whenever I hear it, I can't help laughing hysterically. Nowadays with two to three hundred new viruses arriving monthly, it would be total waste of time and money for anti-virus manufacturers to develop viruses. Why should they bother if new viruses arrive in dozens virtually daily, completely free of charge? There were about 3,000 known DOS viruses at the beginning of 1994. A year later, in January 1995, the number of viruses was estimated at least 5,000. Another six months later, in July 1995, the number exceeded 7,000. Many anti-virus experts expect the number of known DOS viruses to reach the 10,000 mark by the end of 1995. With this tremendous and still fast-growing number of viruses to fight, traditional virus signature scanning software is pushed to its limits [*Skulason, Bontchev2*]. While several years ago a scanner was often developed, updated and supported by a single person, today a team of a dozen skilled employers is only barely sufficient. With the increasing number of viruses, R&D and Quality Control time and resource requirements grow. Even monthly scanner updates are often late, by one month at least! Many formerly successful anti-virus vendors are giving up and leaving the anti-virus battleground and market. The fast-growing number of viruses heavily affects scanners themselves. They become bigger, and sometimes slower. Just few years ago a 360Kb floppy diskette would be enough to hold half a dozen popular scanners, leaving plenty of room for system files to make the diskette bootable. Today, an average good signature-based scanner alone would occupy at least a 720Kb floppy, leaving virtually no room for anything else.

So, are we losing the war? I would say: not yet – but if we get stuck with just virus signature scanning, we will lose it sooner or later. Having realised this some time ago, anti-virus researchers started to look for more generic scanning techniques, known as *heuristics*.

2.1 WHAT ARE HEURISTICS?

In the anti-virus area, heuristics are a set of rules which should be applied to a program to decide whether the program is likely to contain a virus or not. From the very beginning of the history of computer viruses different people started looking for an ultimate generic solution to the problem. Really, how does an anti-virus expert know that a program is a virus? It usually involves some kind of reverse engineering (most often disassembly) and reconstructing and understanding the virus' algorithm: what it does and how it does it. Having analysed hundreds and hundreds of computer viruses, it takes just few seconds for an experienced anti-virus researcher to recognise a virus, even it is a new one, and never seen before. It is

almost a subconscious, automated process. Automated? Wait a minute! If it is an automated process, let's make a program to do it!

Unfortunately (or rather, fortunately) the analytic capabilities of the human brain are far beyond those of a computer. As was proven by Fred Cohen [*Cohen*], it is impossible to construct an algorithm (e.g. a program) to distinguish a virus from a non-virus with 100 per cent reliability. Fortunately, this does not rule out a possibility of 90 or even 99 per cent reliability. The remaining one per cent, we hope to be able to solve using our traditional virus signatures scanning technique. Anyway, it's worth trying.

2.2 SIMPLE HEURISTICS

So, how do they do it? How does an anti-virus expert recognise a virus? Let us consider the simplest case: a parasitic non-resident appending COM file infector. Something like Vienna, but even more primitive. Such a virus appends its code to the end of an infected program, stores a few (usually just three) first bytes of the victim file in the virus body and replaces those bytes with a code to pass control to the virus code. When the infected program is executed, the virus takes control. First, it restores the original victim's bytes in its memory image. It then starts looking for other COM files. When found, the file is opened in Read_and_Write mode; then the virus reads the first few bytes of the file and writes itself to the end of the file. So, a primitive set of heuristical rules for a virus of this kind would be:

1. The program immediately passes control close to the end of itself
2. It modifies some bytes at the beginning of its copy in memory
3. Then it starts looking for executable files on a disk
4. When found, a file is opened
5. Some data is read from the file
6. Some data is written to the end of the file.

Each of the above rules has a corresponding sequence in binary machine code or assembler language. In general, if you look at such a virus under DEBUG, the favourite tool of anti-virus researchers, it is usually represented in a code similar to this:

```
START:                          ; Start of the infected program
          JMP VIRUSCODE         ; Rule 1:  the control is passed
                                ; to the virus body
                                ;

          <victim's code>

VIRUS:                          ; Virus body starts here

SAVED:                          ; Saved original bytes of the
                                  victim's code

MASK:     DB '*.COM',0          ; Search mask

VIRUSCODE:                      ; Start of the virus code
          MOV DI,OFFSET START   ; Rule 2: the virus restores
          MOV SI,OFFSET SAVED   ; victim's code
          MOVSW                 ; in memory
          MOVSB                 ;

        MOV DX,OFFSET MASK    ; Rule 3: the virus
```

```
MOV AH,4EH              ; looks for other
INT 21H                 ; programs to infect

MOV AX,3D02H            ; Rule 4: the virus opens a file
INT 21H ;

MOV DX,OFFSET SAVED     ; Rule 5: first bytes of a file
MOV AH,3FH              ; are read to the virus
INT 21H                 ; body

MOV DX,OFFSET VIRUS     ; Rule 6: the virus writes itself
MOV AH,40H              ; to the file
INT 21H                 ;
```

Figure 1. A sample virus code

When an anti-virus expert sees such code, it is immediately obvious that this is a virus. So, our heuristical program should be able to disassemble a binary machine-language code in a similar manner to DEBUG, and to analyse it, looking for particular code patterns in a similar manner to an anti-virus expert. In the simplest cases, such as the one above, a set of simple wildcard signature string matching would do for the analysis. In this case, the analysis itself is simply checking whether the program in question satisfies rules 1 through 6; in other words, whether the program contains pieces of code corresponding to each of the rules.

In a more general case, there are many different ways to represent one and the same algorithm in machine code. Polymorphic viruses, for example, do this all the time. So, a heuristic scanner must use many clever methods, rather than simple pattern-matching techniques. Those methods may involve statistical code analysis, partial code interpretation, and even CPU emulation, especially to decrypt self-encrypted viruses: but you would be surprised to know how many real life viruses would be detected by the above six simple heuristics alone! Unfortunately, some non-virus programs would be 'detected' too.

2.3 FALSE ALARMS PROBLEM

Strictly speaking, heuristics do not detect viruses. As behaviour blockers, heuristics are looking for virus-like behaviour. Moreover, unlike the behaviour blockers, heuristics can detect not the behaviour itself, but just *potential ability* to perform this or that action. Indeed, the fact that a program contains a certain piece of code does not necessarily mean that this piece of code is ever executed. The problem of discovering whether this or that code in a program ever gets control is known in the theory of algorithms as the Halting Problem, and is in general unsolvable. This issue was the basis of Fred Cohen's proof of the impossibility of writing a perfect virus detector. For example, some scanners contain pieces of virus code as the signatures for which to scan. Those pieces might correspond to each and every one of the above six rules. But they are never executed – the scanner uses them just as its static data. Since, in general, there is no way for heuristics to decide whether these code pieces are ever executed or not, this can (and sometimes does) cause *false alarms*.

A false alarm is when an anti-virus product reports a virus in a program, which in fact does not contain any viruses at all. Different types of false alarms, as well as most widespread causes of false alarms, are described in [*Solomon*] for example. A false alarm might be even more costly than an actual virus infection. We all keep saying to users: 'The main thing to remember when you think you've got a virus – **do not panic**!' Unfortunately, this does not work well. The average user will panic. And the user panics even more if the anti-virus software is unsure itself whether it is a virus or not. In the case, say, where a scanner definitely detects a virus, the scanner is usually able to detect all infected programs, and to remove the virus. At this point, the panic is usually over; but if it is a false alarm, the scanner will not be able to remove the virus, and most likely will report something like: 'This file seems to have a virus', naming just a single file as infected. This is when the user really starts to panic. 'It must be a new virus!' – the user thinks. 'What do

I do?!' As a result, the user well might format his/her hard disk, causing himself a far worse disaster than a virus could. Formatting the hard disk is an unnecessary and un-justified act, by the way; even more so as there are many viruses which would survive this act, unlike legitimate software and data stored on the disk.

Another problem a false alarm can (and did) cause is negative impact on a software manufacturing company. If an anti-virus software falsely detects a virus in a new software package, the users will stop buying the package and the software developer will suffer not only profit losses, but also a loss of reputation. Even if it was later made known that it was a false alarm, too many people would think: 'There is no smoke without fire', and would treat the software with suspicion. This affects the anti-virus vendor as well. There has already been a case where an anti-virus vendor was sued by a software company whose anti-virus protection mistakenly reported a virus.

In a corporate environment, when a virus is reported by anti-virus software, whether it is a false alarm or not, the normal flow of operation is interrupted. It takes at best several hours to contact the anti-virus technical support and to ensure it was a false alarm before normal operation is resumed – and, as we all know, time is money. In the case of a big company, time is big money.

So, it is not at all surprising that, when asked what level of false alarms is acceptable (10 per cent? 1 per cent? 0.1 per cent?), corporate customers answer: 'Zero per cent! We do not want any false alarms!'

As previously explained, by its very nature heuristic analysis is more prone to false alarms than traditional scanning methods. Indeed, not only viruses but many scanners as well would satisfy the six rules we used as an example: a scanner does look for executable files, opens them, reads some data and even writes something back when removing a virus from a file. Can anything be done to avoid triggering a false positive on a scanner? Let's again turn to the experience of a human anti-virus expert. How does one know that this is a scanner, and not a virus? Well, this is more complicated than the above example of a primitive virus. Still, there are some general rules too. For example, if a program relies heavily on its parameters or involves an extensive dialogue with a user, it is highly unlikely that the program is a virus. This leads us to the idea of *negative heuristics*; that is, a set of rules which are true for a non-virus program. Then, while analysing a program, our heuristics should estimate the probability of the program to be a virus using both positive heuristics, such as the above six rules, and negative heuristics, typical for non-virus programs and rarely used by real viruses. If a program satisfies all our six positive rules, but also expects some command-line parameters and uses an extensive user dialogue as well, we would not call it a virus.

So far so good. Looks like we found a solution to the virus glut problem, right? Not really! Unfortunately, not all virus writers are stupid. Some are also well aware of heuristic analysis, and some of their viruses are written in a way which avoids the most obvious positive heuristics. On the other hand, these viruses include otherwise useless pieces of code, the only aim of which is to trigger the most obvious negative heuristics, so that such a virus does not draw the attention of a heuristical analyser.

2.4 VIRUS DETECTION VS. FALSE ALARMS TRADE-OFF

Each heuristic scanner developer sooner or later comes to the point when it is necessary to make a decision: 'Do I detect more viruses, or do I cause less false alarms?' The best way to decide would be to ask users what do they prefer. Unfortunately, the users' answer is: 'I want it all! 100 per cent detection rate and no false alarms!' As mentioned above, this cannot be achieved. So, a virus detection versus false alarms trade-off problem must be decided by the developer. It is very tempting to build the heuristic analyser to detect almost all viruses, despite false alarms. After all, reviewers and evaluators who publish their tests results in magazines read by thousands of users world-wide, are testing just the detection rate. It is much more difficult to run a good false alarms test: there are gigabytes and gigabytes of non-virus software in the world, far more than there are viruses; and it is more difficult to get hold of all this software and to keep it for your tests. 'Not enough disk space' is only one of the problems. So, let's forget false alarms and negative heuristics and call a virus each and every program which happens to satisfy just some of our

positive heuristics. This way we shall score top most points in the reviews. But what about the users? They normally run scanners not on a virus collection but on a clean disks. Thus, they won't notice our almost perfect detection rate, but are very likely to notice our not-that-perfect false alarms rate. Tough choice. That's why some developers have at least two modes of operation for their heuristical scanners .The default is the so-called 'normal' or 'low sensitivity' mode, when both positive and negative heuristics are used and a program needs to trigger enough positive heuristics to be reported as a virus. In this mode, a scanner is less prone to false alarms, but its detection rate might be far below what is claimed in its documentation or advertisement. The often-used (in advertising) figures of 'more than 90 per cent' virus detection rate by heuristic analyser refer to the second mode of operation, which is often called 'high sensitivity' or 'paranoid' mode. It is really a paranoid mode: in this mode, negative heuristics are usually discarded, and the scanner reports as a possible virus any program which happens to trigger just one or two positive heuristics. In this mode, a scanner can indeed detect 90 per cent of viruses, but it also produces hundreds and hundreds of false alarms, making the 'paranoid' mode useless and even harmful for real-life everyday use, but still very helpful when it comes to a comparative virus detection test. Some scanners have a special command-line option to switch the paranoid mode on; some others switch to it automatically whenever they detect a virus in the normal low sensitivity mode. Although the latter approach seems to be a smart one, it takes just a single false alarm out of many thousands of programs on a network file server to produce an avalanche of false virus reports.

2.5 HOW IT ALL WORKS IN PRACTICE: DIFFERENT SCANNERS COMPARED

Being myself an anti-virus researcher and working for a leading anti-virus manufacturer, I have developed a heuristic analyser of my own. And of course, I could not resist comparing it to other existing heuristic scanners. We believe the results will be interesting to other people. They underscore what was said about both virus detection and false alarms rates. As the products tested are our competitors, we decided not to publish their names in the test results. So, only FindVirus of *Dr Solomon's AntiVirus Toolkit* is called by its real name. All the other scanners are referred to with letters: Scanner_A, Scanner_B, Scanner_C and Scanner_D. The latest versions of the scanners available at the time of the test were used. For FindVirus, it was version 7.50 – the first version to employ a heuristic analyser.

Each scanner tested was run in heuristics-only mode, with normal virus signature scanning disabled. This was achieved by either using a special command-line option, where available, or using a special empty virus signature database in other cases.

The test consisted of two parts: virus detection rate and false alarms rate. For the virus detection rate *S&S International Plc* ONE OF EACH virus collection was used, containing more than 7,000 samples of about 6,500 different known DOS viruses. For the false alarms test the shareware and freeware software collection of SIMTEL20 CD-ROM (fully unpacked), all utilities from different versions of MS-DOS, IBM DOS, PC-DOS and other known files were used (current basic *S&S* false alarms test set).

When measuring false alarms and virus detection rate, all files reported were counted; reported either as 'Infected' or 'Suspicious'. Separate figures for the two categories are given where applicable.

In both parts of the test, the products were run in two heuristic sensitivity modes, where applicable: normal or low sensitivity mode, and paranoid or high sensitivity mode. The automatic heuristic sensitivity adjustment was prohibited, where applicable.

The results of the tests are as follows:

Virus Detection Test

	Files scanned	Files triggered (infected + suspicious) Normal			Paranoid	
FindVirus	7375	5902	(N/A)	80.02%	N/A	
Scanner_D	7375	5743	(0+5743)	77.87%	6182 (0+6182)	83.54%
Scanner_C	7375	5692	(0+5692)	77.18%	N/A	
Scanner_A	7375	4250	(N/A)	57.63%	6491 (N/A)	87.74%
Scanner_B	7392(*)	3863	(2995+868)	52.38%	6124 (2992+3132)	82.68%

(*) Scanner_B was tested couple of days later, when 17 more infected files were added to the collection.

Table 1. Virus detection test results.

False alarms test

	Files scanned(*)	Files triggered (infected + suspicious) Normal			Paranoid	
FindVirus	13603	0	(N/A)	0.000%	N/A	
Scanner_A	13428	11	(N/A)	0.082%	371 (N/A)	2.746%
Scanner_B	13471	17	(0+17)	0.126%	382 (0+382)	2.836%
Scanner_D	13840	24	(0+24)	0.173%	254 (0+254)	1.824%
Scanner_C	13603	28	(0+28)	0.206%	N/A	

(*) Different number of files reported as scanned is due to the fact different products treat somewhat different sets of file extensions as executables.

Table 2. False alarms test results

3 WHY 'OF THE YEAR 2000'?

Well, first of all simply because I could not resist the temptation of splitting the name of the paper into three questions and using them as the titles of the main sections of his presentation. I thought it was funny. Maybe I have a weird sense of humour. Who knows...

On the other hand, the year 2000 is very attractive by itself. Most people consider it a distinctive milestone in all aspects of human civilisation. This usually happens to the years ending with double zero; still more to the end of a millennium, with its triple zero at the end. The anti-virus arena is not an exclusion. For example, during the EICAR'94 conference there were two panel sessions discussing 'Viruses of the year 2000' and 'Scanners of the year 2000' respectively. The general conclusion made by a panel of well-known anti-virus researchers was that, at the current pace of new virus creation by the year 2000, we well might face dozens (if not hundreds of thousands) of known DOS viruses. As I tried to explain in the second section of this paper (and other authors explained elsewhere [*Skulason, Bontchev2*]), this might be far too much for a current standard scanners' technique, based on known virus signature scanning. More generic anti-virus tools, such as behaviour blockers and integrity checkers, while being less vulnerable to the growing number of viruses and the rate at which the new viruses appear, can detect a virus only **when** it is already running on a computer or even only **after** the virus has run and infected other programs. In many cases, the risk of allowing a virus to run on your computer is just not affordable. Using a heuristic scanner, on the other hand, allows detection of most of new viruses with a regular scanner safe manner: **before** an infected program is copied to your system and executed. And very much like behaviour blockers and integrity checkers, a heuristic scanner is much more generic than a signature scanner, requires much rare updates, and provides an instant response to a new virus. Those 15-20 per cent of viruses which a heuristic

scanner cannot detect could be dealt with using current well-developed signature scanning techniques. This will effectively decrease the virus glut problem five fold, at least.

Yet another reason for choosing the year 2000 and not, say, 2005 is that I have strong doubts whether the current computer virus situation will survive the year 2000 by more than a couple of years. With new operating systems and environments appearing (Windows NT, Windows'95, etc.) I believe DOS is doomed. So are DOS viruses. So is the modern anti-virus industry. This does not mean viruses are not possible for new operating systems and platforms – they are possible in virtually any operating environment. We are aware of viruses for Windows, OS/2, Apple DOS and even UNIX. But to create viruses for these operating systems, as well as for Windows NT and Windows'95, it requires much more skill, knowledge, effort and time than for the virus-friendly DOS. Moreover, it will be much more difficult for a virus to replicate under these operating systems. They are far more secure than DOS, if it is possible to talk about DOS security at all. Thus, there will be far fewer virus writers and they will be capable of writing far fewer viruses. The viruses will not propagate fast and far enough to represent a major problem. Subsequently, there will be no virus glut problem. Regrettably, there will be a much smaller anti-virus market, and most of today's anti-virus experts will have to find another occupation...

But until then, DOS lives, and anti-virus developers still have a lot of work to do!

REFERENCES

[*Bontchev1*] Vesselin Bontchev, 'Possible Virus Attacks Against Integrity Programs And How To Prevent Them', *Proc. 2nd Int. Virus Bulletin Conf.*, September 1992, pp. 131-141.

[*Skulason*] Fridrik Skulason, 'The Virus Glut. The Impact of the Virus Flood', *Proc. 4th EICAR Conf.*, November 1994, pp. 143-147.

[*Bontchev2*] Vesselin Bontchev, 'Future Trends in Virus Writing', *Proc. 4th Int. Virus Bulletin Conf.*, September 1994, pp. 65-81.

[*Cohen*] Fred Cohen, 'Computer Viruses – Theory and Experiments', Computer Security: A Global Challenge, Elsevier Science Publishers B. V. (North Holland), 1984, pp. 143-158.

[*Solomon*] Alan Solomon, 'False Alarms', *Virus News International*, February 1993, pp. 50-52.