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
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.