DYNAMIC DETECTION AND CLASSIFICATION OF COMPUTER VIRUSES USING GENERAL BEHAVIOUR PATTERNS

Morton Swimmer

Virus Test Center, University of Hamburg, Odenwaldstr. 9, 20255 Hamburg, Germany
Tel +49 404 910041 · Fax +49 405 471 5226 · Email swimmer@acm.org

Baudouin Le Charlier and Abdelaziz Mounji

F.U.N.D.P., Institut d'Informatique, University of Namur, Belgium
Email ble@info.fundp.ac.be / amo@info.fundp.ac.be

ABSTRACT

The number of files which need processing by virus labs is growing exponentially. Even though only a small proportion of these files will contain a new virus, each file requires examination. The normal method for dealing with files is still brute force manual analysis. A virus expert runs several tests on a given file and delivers a verdict on whether it is virulent or not. If it is a new virus, it will be necessary to detect it. Some tools have been developed to speed up this process, ranging from programs which identify previously-classified files to programs that generate detection data. Some anti-virus products have built-in mechanisms based on heuristics, which enable them to detect unknown viruses. Unfortunately all these tools have limitations.

In this paper, we will demonstrate how an emulator is used to monitor the system activity of a virtual PC, and how the expert system ASAX is used to analyse the stream of data whicg the emulator produces. We use general rules to detect real viruses generically and reliably, and specific rules to extract details of their behaviour. The resulting system is called VIDES: it is a prototype for an automatic analysis system for computer viruses and possibly a prototype anti-virus product for the emerging 32 bit PC operating systems.

1 INTRODUCTION

Virus researchers must cope with many thousands of suspected files each month, but the problem is not so much the number of new viruses (which number perhaps a few hundred and grows at a nearly exponential rate) as the number of files the researcher receives and must analyse - the glut. Out of perhaps one hundred files, only one may actually contain a new virus. Unfortunately, there are no short cuts. Every file has to be processed.

The standard method of sorting out such files is still brute force manual analysis, requiring specialists. Some tools have been developed to help cope with the problem, ranging from programs which identify and remove previously-classified files and viruses to utilities which extract strings from infected files that aid in identifying the viruses. However, none of the solutions are satisfactory. Clearly, more advanced tools are needed.

In this paper, the concept of dynamic analysis as applied to viruses is discussed. This is based on an idea called VIDES (*Virus Intrusion Detection Expert System*), coined at the Virus Test Center [BFHS91]. The system will comprise of a PC emulation and an IDES-like expert system. It should be capable of detecting viral behaviour using a set of *a priori* rules, as shown in the preliminary work done with Dr. Fischer-Hübner. Furthermore, advanced rules will help in classifying the detected virus.

The present version of VIDES is only of interest to virus researchers; it is not designed to be a practical system for the end-user - its demands on processing power and hardware platform are too high. However, it can be used to identify unknown viruses rapidly and provide detection and classification information to the researcher. It also serves as a prototype for the future application of intrusion detection technology in detecting malicious software under future operating systems, such as OS/2, MS-Windows NT and 95, Linux, Solaris, etc.

The rest of the paper is organized as follows: Section 2 presents the current state of the art in anti-virus technology; Section 3 describes a generic virus detection rule; Section 4 discusses the architecture of the PC auditing system; Section 5 shows how the expert system ASAX is used to analyse the activity data collected by the PC emulator; and finally, Section 6 contains some concluding remarks.

2 CURRENT STATE OF THE ART

For the purpose of discussion it will be necessary to define the term computer virus.

2.1 TERMS

There is still no universally-agreed definition for a computer virus. What is missing is a description which is still general enough to account for all possible implementations of computer viruses. An attempt was made in [Swi95], which is the result of many years of experience with viruses in the Virus Test Center. The following definition for a computer virus is the result of discussion in comp.virus (Virus-L) derived from [Seb]:

Def 1 *A* Computer Virus *is a routine or a program that can 'infect' other programs by modifying them or their environment such that a call to an infected program implies a call to a possibly evolved, functionally similar, copy of the virus.*

A more formal, but less useful, definition of a computer virus can be found in [Coh85]. Using the formal definition, it was possible to prove the virus property undecidable.

We talk of the infected file as the *host program*. System viruses infect system programs, such as the boot or Master Boot Sector, whereas file viruses infect executable files such as EXE or COM files. For an in-depth discussion of the properties of viruses, please refer to literature such as: [Hru92], [SK94], [Coh94] or [Fer92].

Today, anti-virus technology can be divided into two approaches: the *virus specific* and the *generic* approach. In principle, the former requires knowledge of the viruses before they can be detected. Due to advances in technology, this prerequisite is no longer entirely valid in many of the modern anti-virus products. This type of technology is known to us as a *scanner*. The latter attempts to detect a virus by observing attributes characteristic of all viruses. For instance, integrity checkers detect viruses by checking for modifications in executable files; a characteristic of many (although not all) viruses.

2.2 VIRUS SPECIFIC DETECTION

Virus specific detection is by far the most popular type of virus protection used on PCs. Information from the virus analysis is used in the so-called scanner to detect it. Usually, a scanner uses a database of virus identification information which enable it to detect all viruses previously analysed.

The term *scanner* has become increasingly incorrect terminology. The term comes from *lexical scanner*, i.e. a pattern matching tool. Traditionally scanners have been just that. The information extracted from viruses were strings which were representative of that particular virus. This means that the string has to:

- differ significantly from all other viruses, and
- differ significantly from strings found in *bona fide* anti-virus programs.

Finding such strings was the entire art of anti-virus program writing until polymorphic viruses appeared on the scene.

Encrypted viruses were the first minor challenge to string searching methods. The body of the virus was encrypted in the host file, and could not be sought, due to its variable nature. However, the body was prepended by a decryptor-loader which must be in plain text (unencrypted code); otherwise it would not be executable. This decryptor can still be detected using strings, even if it becomes difficult to differentiate between viruses.

Polymorphic viruses are the obvious next step in avoiding detection. Here, the decryptor is implemented in a variable manner, so that pattern matching becomes impossible or very difficult. Early polymorphic viruses were identified using a set of patterns (strings with variable elements). Moreover, simple virus detection techniques are made unreliable by the appearance of the so-called *Mutation Engines* such as MtE and TPE (Trident Polymorphic Engine). These are object library modules generating variable implementations of the virus decryptor. They can easily be linked with viruses to produce highly polymorphic infectors. Scanning techniques are further complicated by the fact that the resulting viruses do not have any scan strings in common even if their structure remains constant. When polymorphic technology improved, statistical analysis of the opcodes was used.

Recently, the best of the scanners have shifted course from merely detecting viruses to attempting to identify the virus. This is often done with added strings, perhaps position dependent, or checksums, over the invariant part of the virus. To support this, many anti-virus products have implemented machine-code emulators so that the virus' own decryptor can be used to decrypt the virus. Using these enhancements, the positive identification of even polymorphic viruses poses no problem.

The next shift many scanners are presently experiencing is away from known virus only detection to detection of unknown viruses. The method of choice is *heuristics*. Heuristics are built into an anti-virus product in an attempt to deduce whether a file is infected or not. This is most often done by looking for a pattern of certain code fragments that occur most often in viruses and hopefully not in *bona fide* programs.

Heuristics analysis suffers from a moderate to high false-positive rate. Of course, a manufacturer of a heuristic scanner will improve the heuristics both to avoid false positives and still find all new viruses, but both cannot be achieved completely. Usually, a heuristic scanner will contain a 'traditional' pattern-matching component, so that viruses can be identified by name.

2.3 GENERIC VIRUS DETECTION

Computer viruses must replicate to be viruses. This means that a virus must be observable by its mechanism of replication.

Unfortunately, it is not as easy to observe the replication as it may seem. DOS, in it various flavours, provides no process isolation, or even protection of the operating system from programs. This means that any monitoring program can be circumvented by a virus which has been programmed to do so. There used to be many anti-virus programs which would try to monitor system activity for viruses, but were not proof against all viruses. This problem led to the demise of many such programs. Later in the paper, we shall discuss how we avoided the problem when implementing VIDES.

A more common approach is to detect symptoms of the infection such as file modifications. This type of program is usually called an *integrity checker* or *checksummer*.

When programs are installed on the PC, checksums are calculated over the entire file, or over portions of the file. These checksums are then used to verify that the programs have not been modified. The shortcoming of this method is that the integrity checker can detect a modification in the file, but cannot determine whether the modification is due to a virus or not. A legitimate modification to, for instance, the data area of a program will cause the same alarm as a virus infection.

Another problem is virus technology aimed specifically against anti-virus products. Advances in stealth and tunnelling technology have made updates necessary. There have also been direct attacks against particular integrity checkers, rendering them useless. Again, the lack of support from the operating system makes the prevention of such attacks very difficult. As a consequence, the acceptance of such products is low.

The non-specific nature of the detection has little appeal for many of the users. Even generic repair facilities in the anti-virus products do not help, despite these methods effectively rendering identification unnecessary. The problem is partly understandable. The user is concerned with his data. Merely disinfecting the programs is not enough if data has been manipulated. Only if the virus has been identified and analyzed can the user determine if his data was threatened.

Generic virus detection technology should not be dismissed. It is just as valid as virus-specific technology. The problems so far have stemmed from the permissiveness of the underlying operating system, DOS, and from the limits in the programs. Both problems can be addressed.

3 DYNAMIC DETECTION RULES

Before we can attempt to detect a virus using ASAX, we need to model the virus attack strategy. This is then translated into RUSSEL, the rule-based language which ASAX uses to identify the virus attack.

3.1 REPRESENTING INFECTION PATTERNS USING STATE TRANSITION DIAGRAMS

State transition diagrams are eminently suitable for representing virus infection scenarios. In this model of representation, we distinguish two basic components: a node in a state transition diagram represents some aspects of the computing system state. Arcs represents actions performed by a program in execution. Given a (current) state s_i, the action a takes the system from the state s_i to the state s_f as shown in Figure 1. The infection process played by a virus can be viewed as a sequence of actions which drives the system from an initial *clean* state to a final *infectious* state, where some files are infected. In order to get a complete description of the actual scenario, a state is adorned by a set of *assertions*, characterizing the objects as affected by actions.

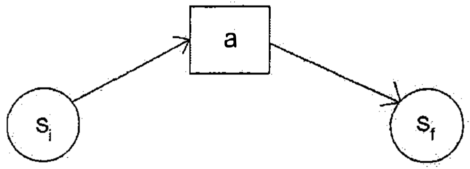

Figure 1: State transition diagram

In practice, we only represent those actions relevant to the infection scenario. As a result, many possible actions may occur between adjacent states, but are not recorded because they do not entail a modification in the current state. In terms of auditing, irrelevant audit records may be present in the sequence of audit records representing the infection signature.

For the sake of simplicity, discussion of the generic detection rules are based on the state transition diagrams described above.

3.2 BUILDING THE RULES

VIDES uses three types of detection rules: *generic detection rules*, *virus specific rules*, *other rules*. As its name implies, generic rules are used to detect all viruses which use a known attack pattern. For this, models of virus behaviour are needed for the target system (in our case MS-DOS). Virus-specific rules use information from a previous analysis to detect that specific virus, or direct variants. These rules are similar to virus-specific detection programs, except for the fact that they analyze the dynamic behaviour of the virus instead of its code. Finally, there are the 'other rules' for gleaning other information from the virus which can be used in its classification.

We will not go into the virus-specific rules or the 'other' rules, concentrating instead on the generic rules.

In developing a generic rule for detecting viruses, we need to have a model for the virus attack. No one model will do, because MS-DOS viruses can use choose from many effective strategies. This is compounded by the diversity of executable file types for MS-DOS. Fortunately for us, the majority of viruses have chosen one particular strategy, and infect only two types of executable files. This means that we can detect most viruses with very few rules. On the other hand, a virus which uses an unknown attack strategy will not be detected. For this reason, the prototype analysis system contains an auxiliary static analysis component to detect such problems.

In the following, we will develop a generic rule which detects file infectors that modify the file directly to gain control over that file. We will concentrate on COM file infectors. EXE file infectors are detected in an analogous way.

We must make two assumptions about the behaviour of DOS viruses to help us build the rule.

Assumption 1: *A file-infecting virus modifies the host file in such a way that it gains control over the host file when the host file is run.*

This is a specific version of the virus definition (Def 1). However, it doesn't specify when the virus gains control over the host file.

Assumption 2: *The virus in an infected file receives control over the file before the original host program.*

That is, when the infected file is run, the virus is run before the host program.

Discussion: If the virus never gains control over the host file, it would not fulfil the definition of a virus. This observation leads to Assumption 1. However, there is no reason (in the definition) why the virus must gain control before the host does.

We make an additional assumption that the virus *does* gain control before the host program does. The reason we do this is to avoid very blatant false positives. However, it should be noted that Assumption 2 does not result from the virus definition, and will cause some viruses to be missed. For these cases, other rules are used.

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.