ATTACKS ON WIN32 – PART II

Péter Ször
Symantec (SARC Division), 2500 Broadway, Suite 200, Santa Monica, CA 90404-3036, USA
Tel +1 310 453 4600 • Fax +1 310 453 0636 • Email pszor@symantec.com

ABSTRACT

In 1998 several anti-virus companies introduced heuristic scanning for 32-bit Windows *viruses. As a result the number of anti-heuristic viruses is on the rise. In my paper I will introduce infection methods with special attention to the anti-heuristic infection techniques. I will also provide results achieved by testing old Win32 viruses and worms on* Windows 2000. *This provides a better understanding of the impact of old Win32 viruses on* Windows 2000 *and vice versa.*

The current number of 32-bit Windows *viruses is almost 400. We have reached the point when automatic replication, detection and repair for the simple* Windows *viruses is becoming mandatory.*

PE repair is becoming very difficult if not impossible. Unfortunately, we have seen examples of new binary virus variants that were the result of inexact PE repair. This is a valid scenario since several products use CRCs to identify Trojans and worms. When a networked worm gets infected and repaired the result can be an unknown Windows *virus. In this section I will describe the inexact nature of PE file repair.*

Furthermore, I will introduce the common techniques used by 32-bit Windows *email worms and find out the possibilities, if any, of detecting them more generically.*

In the last part of the presentation I would like to talk about possible new virus models we are likely to see in the near future.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

PREFACE

The first part of this paper 'Attacks on Win32'[1] was published back in 1998 at the eighth *Virus Bulletin* conference in Munich. At the time of writing the original paper the number of different 32-bit *Windows* virus variants was less than 40.

Despite the small number of *Windows 95* viruses, the impact was greater than expected. Several *Windows 95* viruses got into the wild and the need for a heuristic detector became very clear. Two years have passed and the number of 32-bit creations has jumped. In April 1999 we saw about 116 such viruses. A year later the number of 32-bit *Windows* virus variants passed 500.

Probably the most important Win32 development by virus writers was in Win32 worms. Several such creations made it to the wild. Win32/Ska.A[2] was one of the most successful creations of the last decade. Currently (May 2000) Win32/PrettyPark variants are the most reported viruses to *SARC*. Certainly 32-bit binary viruses gained attention during the last two years and they represent a significant danger for corporations worldwide.

Modern virus writing is moving from the decade of viruses to the decade of networked worms. In this paper I will try to see if more generic detection methods could be used against at least some of the Win32 worm classes. Additionally, I will describe some of the incompatibility problems that exist between Win32 viruses and *Windows 2000*. The impact of *Windows 2000* on Win32 viruses is not major, although some 25% of old Win32 viruses will not replicate under *Windows 2000*.

In this paper I will cover the most interesting virus infection techniques developed by 32-bit *Windows* virus writers which made the job of heuristic scanners and emulators more difficult. In some cases I will assume that the reader is familiar with the first part of this paper in order to avoid repeating myself unnecessarily.

INTRODUCTION

> *'There is every reason to believe that, as software technology evolves over the next century or so, there will plenty of important and interesting new problems that must be solved in this field.'*
>
> Dr. Steve R White, IBM Thomas J. Watson Research Center[3]

In 1998 *Windows* virus development was at a relatively early stage. This is why a wide variety of different infection methods were introduced which made it possible to contemplate using heuristics as a defence against 32-bit *Windows* viruses. Careful analysis of different infection types led to the development of first generation Win32 heuristic detectors that used static heuristics. Static heuristics are capable of pinpointing suspicious Portable Executable (PE) file structures. As a result of this, they are able to catch first generation 32-bit *Windows* viruses with a very high detection rate.

Until the end of 1999 many new virus replication methods were already developed, including those of Kernel mode viruses such as WinNT/Infis.4608[4]. Moreover, a major part of 32-bit *Windows* viruses used some sort of encryption, polymorphism or metamorphic techniques. Encrypted viruses are more difficult to detect and very dangerous when we look forward to the future of scanners.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

Scanners are relatively slow in the case of modern 32-bit PE entry point obscuring viruses such as Win95/SK, Win32/CTX or Win32/Dengue. Some scanners do survive by using cryptographic detection against such viruses but it is only a matter of complezity and the detection of a single virus can become extremely slow. (Cryptographic methods detect the virus without emulation/or partial emulation only using the X-RAY technique.)

A possible solution is the reuse of polymorphic decryptor loop detection. That method was originally used to detect polymorphic viruses at the time when generic decryptors (using code emulation) did not ezist. Unfortunately, that technique causes more false positives as well as false negatives.

Modern anti-virus software needs to combine all of these techniques and perform heuristic scanning for certain file viruses in order to keep the virus scanner speed fast enough for the users. Certainly more generic methods need to be introduced for the public to deal with the viruses of the next decade.

Static heuristics were extremely successful against PE file viruses. Even virus writers were surprised at the situation, but as always we did not have to wait too long until they came up with attacks against heuristic detectors. Several anti-heuristic infection types were introduced over the last two years. Most of these methods were already listed in my original paper in 1998 as possible infection techniques of the future. In this paper I will introduce the most important anti-heuristic infection techniques that virus writers came up with.

Some virus writers also introduced anti-emulation techniques against the strongest component of the anti-virus product: the emulator. This paper will also introduce the most important anti-heuristic tricks of virus writers that were developed over the last two years.

1 ATTACKS AGAINST FIRST GENERATION WIN32 HEURISTICS

In this section I describe the attacks against first generation Win32 heuristics that virus writers introduced over the last two years. Examination of these new methods enables us to enhance heuristic scanners to deal with these new tricks. Please use the first part of the paper as a reference to the PE file format whenever necessary.

New PE File Infection Techniques

Many PE file viruses add a new section or append to the last section of PE files. Even today many viruses can be detected with the simplest possible PE file heuristic. This heuristic has the benefit that it can easily be performed by users. The heuristic checks if the entry point of the PE file points into the last section of the application.

This heuristic can cause false positives, but additional checks can be performed to see if the file is likely to be compressed (compressed PE files cause the most significant false alarm rate since the actual extractor code is very often patched into the last section of the application). A large number of commercial PE file 'on-the-fly' packers were introduced for Win32 platforms such as Neolite, Petite, Shrink32, ASPack, etc… The fact is that such packers were already used by virus writers and other Trojan code creators to hide their creations. Many people can perform heuristic

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England. Tel +44 1235 555139.** No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

examination and a file can be extremely suspicious if it contains certain words or phrases. Therefore virus writers use packers in order to hide these suspicious matters from the heuristics of scanners and humans. Unfortunately certain worm or Trojan code can be changed with packers. Win32/ExploreZip had several variants that were created by using 32-bit PE packing.

Thus, modern anti-virus software needs to deal with new 32-bit packers in its decomposer. It should not matter for the scanner if a *Word* document is placed in a ZIP archive that contains another embedded document that also has an embedded executable object that is packed with a 32-bit packer. The scanner needs to get the last possible decomposed object and perform its scanning task on that without dealing the unpacking as part of the scanning engine. Obviously, the decomposer needs to have a recognizer module that can sort out the packed object easily. This way the false positive rate of 32-bit *Windows* virus heuristics will decrease significantly, while the detection rate of regular scanning will be much better. Virus writers have realized that infection of the last section is a far too obvious method and they have come up with new infection techniques to avoid heuristic detection.

(i) Viruses that do not append to the Last Section

There are various ways to avoid infecting the last section. In this section I will describe the most common tricks used in modern 32-bit *Windows* viruses.

More than one Virus Section

Several Win32 viruses append not just one section but many sections at a time. For instance Win32/Resure virus appends four sections ('.text', '.rdata', '.data' and '.reloc' respectively) to each PE host. Since the entry point of the application will be changed to point into the new '.text' section of the virus, the heuristic is easily fooled. This is because the last section seemingly does not receive any control. Furthermore, the virus will maintain very high compatibility factor to most Win32 systems. This is because the virus is written in C instead of assembly language. The file can be suspicious to heuristics. A human could easily see if the same four sections are added to the section table of a few PE files. However, a heuristic scanner has more difficulty. This is because some linkers create such suspicious structures in some circumstances. Thus, a heuristic developed for these viruses ('multiply-section names') would need to be considered a 'low value' heuristic.

Prepending Viruses that Encrpyt the Host File Header

Another very easy anti-heuristic trick was included in HLL-written prepender viruses. These viruses do not need to deal with PE file formats. First generation heuristics can look for a PE file header at the purported end of the PE file, according to the prepended virus file header. Again, a human can perform such a heuristic easily, even if the end of the file is somewhat encrypted (such as Win32/HLLP.Cramb). However, it is more difficult to create a program for this case, since the encryption of the original application can be performed in various ways, and thus the program heuristic will likely fail.

First Section Infection of Slack Area

Some viruses such as Win95/Invir do not modify the entry point of the application to point to the last section. Rather the virus overwrites the slack area of the first section and jumps from there to

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England. Tel +44 1235 555139.** No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

the start of virus code that is placed somewhere else, such as the last section. This technique is often combined with anti-emulation tricks, as we will see later on. This is because second generation, dynamic heuristics use emulation to see if the actual application jumps from one section to another.

First Section Infection by Shifting the File Sections

The first known virus utilizing this method was the Win32/IKX virus. The virus body of Win32/IKX is longer than would fit into a single section's slack area. Therefore the virus makes a 'hole' for itself by shifting each section of the PE host after its body and inserts itself in between the code '.text' section and the following sections. The virus adjusts the raw data offset of each section by 512 bytes. Obviously, dynamic heuristics are necessary in such a case; that is, performed on the code instead of the structure of the application.

First Section Infection with Packing

Probably one of the most interesting anti-heuristic viruses was Win95/Aldabera. This virus does not infect the last section of PE files. The virus cannot infect small files; rather, it looks for larger PE files and tries to pack their code section. In case the code section of the application can be packed such that the actual packed code and the virus code fit in the very same place, the virus will shrink the code section and place the result together in the code section. After that it will unpack the code section on execution similarly to an 'on-the-fly' packer.

Obviously Win95/Aldabera causes problems to heuristic scanners as well as regular scanners. This is because the virus is rather big and it decrypts/unpacks itself and the code section very slowly during emulation taking many emulator iterations. (The virus also uses the RDA decryption technique.) Moreover, the virus accesses and modifies too much virtual memory. The emulator must keep more than 64 KB of dirty pages in order to decrypt the code properly, and this exceeds some of the early 32-bit code emulators' maximum – those still trying to support the XT platform (oh, well).

(ii) Entry Point Obscuring Techniques

Various 32-bit *Windows* viruses apply a very effective anti-heuristic infection method known as the 'entry point obscuring' or 'inserting' technique. Many old DOS viruses, such as the Nexiv_Der virus[5], used this technique.

As expected, virus writers implemented inserting, polymorphic viruses in order to evade detection by heuristic scanners. To date, the entry point obscuring method is the most advanced technique of virus writers.

Fortunately, the more complicated virus code becomes, the less likely it will spread into the wild. Polymorphic, entry point obscuring viruses are especially complicated and therefore much less likely to cause trouble for users. Moreover, such viruses often limit themselves, because the virus code might not get control when the infected application is executed. Thus, even execution of a dozen infected applications might not infect the system.

SARC received more than 2,000 submissions of the Win32/Pretty worm during March 2000. By contrast, there was only one submission of Win95/SK[6] that became infamous for its complexity. Win95/SK is very difficult to detect just like many other inserting polymorphic viruses.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

Virus writers have already tried to implement mass-mailing features for this kind of complex virus. The Win32/CTX virus was introduced on top of the Win32/Cholera worm. There was a quick initial outbreak of these viruses. This shows that a complex virus can be introduced to the public in a very short time. This will only depend on the emailing features of the viruses. Since most Win32 worms use static code, their detection is very simple. Win32 programs can use extremely complex polymorphic or metamorphic code, their analysis takes a longer time and the scanning solutions are much more complex to create.

Therefore, we strongly suspect that future entry point obscuring, polymorphic viruses will use internal mass-mailing capabilities. In fact, these viruses could potentially convert themselves to polymorphic VBS scripts, which would introduce a very difficult-to-detect virus on local systems. These hidden infections could cause new outbreaks from time to time. Thus, this technique should be handled very carefully. Modern anti-virus software will not survive the coming decade without a scanning engine that supports the detection of such viruses.

Selecting a Random Entry Point in the Code Section

Win95/Padania was one of the first 32-bit *Windows* viruses that occasionally does not modify the entry point of the application (in PE header) to point to the start of virus code. Sometimes the virus searches the relocation section of the application to find a safe position to perform code replacement in the code section of PE files. The virus does change certain CALL or PUSH instructions to a JMP instruction that will point to the start of virus. It selects this position near to the original entry point of the application and therefore the virus code will likely get control when the original application gets executed, although there is no guarantee of that.

Whenever the virus is not encrypted, it is easily detected by those scanning engines which do not apply an 'entry point'-based scanning engine model, but also scan the top and bottom of each PE file for virus strings. Scanners that support neither that sort nor p-code (script language) scanning will be more difficult to update for this kind of attack.

Moreover, heuristic scanning needs to deal with the problem of PE emulation in a different manner. Emulators typically stop when encountering an 'unknown' API call. Thus, the actual position where the JMP is made to the virus entry point will not be reached! The actual issue here is that different APIs have a variable number of parameters and the actual APIs are responsible for their own stack cleaning. It is simply impossible to know all Win32 APIs and the number of parameters passed in to the function which is the minimal need to follow an accurate program code chain. Someone could implement the standard APIs, but what about the thousands of other DLLs?

Even dynamic heuristic file scanning is not effective against such creations. Some files might be detected heuristically whenever the actual jump to the virus start is placed near to the actual entry point of the PE application.

Recycling Compiler Alignment Areas

Several viruses such as Win95/SK or Win95/Orez recycle the compiler alignment areas that are often filled with zeros or 0xCCs by various compilers. Win95/Orez fragments its decryptor into these 'islands' of the PE files. Win95/SK uses a relatively short but efficiently permuted decryptor that replaces such an area in the PE files. Control is given from a randomly selected

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

place via a CALL. The problem is very obvious for scanners. Scanning of such viruses can cause certain speed issues for all products. Such infection can be detected by any of the following methods. Each method has its own benefits and disadvantages:

1. Various products use decryptor detection specific to a certain virus all the way in all code sections. Obviously, the speed of scanning will be dependent on the code section sizes of the actual applications. Decryptor detection was used before generic decryptors were introduced. This technique by itself can cause false positives as well as false negatives. It does not guarantee a solution for repair since the actual virus code will not be decrypted. However, this technique is relatively fast to perform when used after an efficient filter. (A filter can be anything that is virus-specific: the type of the executable, the virus' own identifier in the headers, suspicious code section characteristics or code section name, etc …) Unfortunately, some viruses give little place for filtering. Virus scanners grow slower because of inserting, polymorphic viruses, regardless of the actual virus scanning method used.

2. Another group of scanners uses cryptographic detection. The virus researcher can examine the polymorphic engine of the virus and identify the actual encryption methods used by the polymorphic engine. Methods like XOR, ADD, ROR, etc are often used with 8-bit, 16-bit and 32-bit keys. Sometimes the virus decryptor uses more than one method to encrypt a single byte.

Attacking the encryption of the virus code is called X-RAY scanning. The problem with this method appears when the start of virus body cannot be found at a fixed position and therefore the actual attack against the decryptor must be performed on a long area of the file instead. This causes slowdown. The benefit of the method is the complete decryption of the virus code, which makes repair possible. Moreover, an improperly created/or missing decryptor will not matter for this technique as long as the virus body was placed in the file with accurate encryption.

3. Another technique uses a combination of emulation and decryptor detection. First, the possible decryptor location is identified (this method can assume false decryptor detection since the detection of the decryptor itself will not produce a virus warning). Then the decryptor is executed for an efficient number of iterations and the virus code is identified in the virtual machine of the scanner by checking for search string in the dirty pages of the virtual machine's memory. Such detection can often be created faster than the cryptographic method. However, its depends on the actual iterations of the decryptor loop. With short decryptors, the method will be fast enough to be useful. In the case of longer decryption loops (with a lot of garbage instructions) even partial decryption of the virus code might not be possible fast enough since the number of necessary iterations can be extremely high (in the few million iterations range).

4. A relatively new technique uses a combination of emulation and decryptor detection. This is very much like a heuristic detection of the virus code but leads to a much faster detection. For viruses with longer loops, such as the Win32/Dengue virus, emulation cannot perform very fast because of the long decryption loops. The possible entry point of the virus decryptor can be identified in a virus-specific manner. Then emulation checks which areas of the virtual machine's memory were

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

changed. In case of such a change, additional scanning code can check which instructions were executed during a limited number of iterations.

This way the decryptor detection is more fully proven. This technique can be used to repair the virus since the virus code will need to be emulated for a longer time (up to several minutes in bad cases) in order to decrypt the virus code completely with the use of the emulator.

It appears that inserting, polymorphic viruses are a big problem for all kinds of scanning techniques seeking to be time- and cost-effective. Heuristic methods are not completely useless against such viruses. Modern emulation-based heuristics have a chance to detect such viruses since the decryptor of the virus often represented at the entry point.

Furthermore, the pass of control to the start of virus decryptor is given near to the entry point and therefore it will be reached at least in some cases via emulation.

Note: complete control of the scanning engine and emulator usage is mandatory for the virus researchers to detect a particular polymorphic virus. If the actual scanning is not data-driven (p-code interpretation or some sort of executable object as part of the database) and therefore the standalone code needs to be updated for detection, the actual scanner will not meet expectations since it cannot be updated fast enough.

The virus researcher is in great trouble in that case and, as a result, the customers will be in trouble also[7]. It is very fortunate that none of the fast-spreading email worms have used very difficult polymorphic engines so far, and thus their detection is straightforward for most scanners. The situation can become very different if a fast spreading macro or script virus was to introduce a very difficult-to-detect binary virus on millions of systems, since some scanners could not be used against them in time (in a matter of a few hours).

Anti-Emulation Techniques

First generation heuristics were virus infection-specific. The heuristics checked for possible infection methods used on a particular application. Virus writers realized that some of the scanners use emulation for detecting 32-bit *Windows* viruses and started to create attacks specifically against the strongest component of the scanner – the emulator. In this section I explain some of the anti-emulation techniques that have been used by various 32-bit *Windows* viruses.

(i) Co-processor Instruction Usage

Some virus writers realized the power of emulators and they looked for weaknesses. They quickly realized that co-processor emulation was not implemented. In fact, most emulators skipped co-processor instructions until recently, while most currently used processors support co-processor instructions by default.

The virus does not limit itself any longer when it uses co-processor instructions. In early years of virus development (back in the '80s), virus writers could not utilize the co-processor techniques that much since early processors did not include a co-processor unit. With the introduction of 486 processors *Intel* placed the unit into the actual processor instead. Nowadays, more than 90% of the systems in use have the capability to execute co-processor code.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

Some of the non-polymorphic but encrypted viruses already used co-processor instructions to decrypt themselves. The first polymorphic engine that used co-processor instructions was the Prizzy Polymorphic Engine (PPE). The PPE is capable of generating 43 different co-processor instructions for the use of its polymorphic decryptor.

Obviously, co-processor instructions need to be supported by modern anti-virus software.

(ii) MMX Instruction Usage

Some other virus writers went as far as implementing a virus that used the MMX (multimedia extension) instructions of the *Pentium* processors. The first of this kind was the intended Win95/Prizzy. The virus was simply too buggy to work and no anti-virus researchers managed to replicate the distributed sample of Win95/Prizzy, regardless of the extra care taken to replicate it.

For instance, anti-virus researchers tried to replicate it on the Czech version of *Windows 95* since the virus writer could use such a system, being Czech himself. Fortunately, this virus failed to spread. The Prizzy Polymorphic Engine was capable of generating as many as 46 different MMX-instructions.

In any case, Win95/Prizzy introduced the idea of using the co-processor as garbage instructions in the decryptor of the polymorphic virus. Subsequently, there were other, more successful viruses created, for example Win32/Legacy[8] and Win32/Thorin. These were both considerably better written than their predecessor.

Most MMX-capable polymorphic engines do not assume that MMX is available in the processor. Some viruses try to create two polymorphic decryption loops and try to check the existence of MMX support by using the CPUID instruction. The virus might attempt to replicate on those *Pentium* systems that do not have MMX support, but will fail unless it somehow checks the processor type itself. The CPUID instruction is not available in older processors, while a 386 processor is enough to execute a PE image. Not surprisingly, some of the MMX viruses fail to determine the MMX support properly and execute MMX instructions on non-MMX systems. As a result, they generate a processor exception.

Virus writers may remove the CPUID check from polymorphic engines of the future. This is because the number of *Pentium* systems that support MMX is growing very quickly. Starting from year 2000 MMX emulation is a must for all anti-virus software.

(iii) Structured Exception Handling Usage

The usage of structured exception handling in 32-bit *Windows* viruses is as old as the first real Win32 virus. Win32/Cabanas[9] used a structure exception handler (SEH) for anti-debug purposes as well as to protect itself from potential crash situations. Many 32-bit *Windows* viruses set up an exception handler at their start and return to the host program's entry point (in case of a bug condition) automatically.

Other viruses set up an exception handler in order to create a trap for the emulators used in anti-virus products. Such trick was introduced in Win95/Champ.5447.B. Some of the polymorphic engines generate random code as part of their decryptor. After setting up an exception handler the virus executes the garbage block, forcing an exception to occur. During actual code execution, the virus indirectly executes its own handler, which gives control to another part of the

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

polymorphic decryptor. When the AV product's emulator is not able to handle the exception, the actual virus code will not be reached during code emulation.

Unfortunately, it is not a matter of implementing an exception handler recognizer module in AV products to solve this problem. While some of the exception conditions can be determined easily, the emulated environment (what we can build into an AV product) cannot handle all exceptions perfectly. In a real-world situation it is simple impossible to be 100% sure to know that a particular instruction will cause an exception to be generated[10].

Thus, a virus that uses random garbage blocks to cause a fault might not be realized in the emulation, so the heuristic scanner might not trigger perfectly or at all.

(iv) Random Virus Code Execution – Is this a Virus Today?

Some viruses use a random code execution trap at their entry point. This problem is already recognised in those DOS viruses that only execute on a randomly generated date or time condition. The Win95/SK virus used this logic in its archive droppers, but it relied on interrupt usage just like some other DOS viruses.

One of the first viruses that used random execution logic was Win95/Invir[11], which uses two polymorphic code sections. The first one is inserted in the slack area of the code section. The first polymorphic code calculates the entry point of the virus decryptor that is placed in the last section. However, this depends on a random condition. The virus either transfers control to the host program (original entry point) or gives control to the virus decryptor. In other words, executing an infected program will not guarantee that the virus gets loaded.

Win95/Invir uses the FS:[0Ch] value as the seed of randomness. On Win32 systems under *Intel* machines the data block at FS:0 is known as the Thread Information Block (TIB). For instance, the DWORD value FS:[0] is a pointer to the exception handler chain. The WORD value FS:[0Ch] is called the 'W16TDB' and only has meaning under *Windows 9x*. *Windows NT* specifies this value as 0.

When the value is 0, the virus will execute the host program. How elegant, the virus will not try to load itself under *Windows NT*. Win95/Invir uses VxD functions to hook the file system and is therefore incompatible with *Windows NT/2000*. Executing the virus-infected executable will not cause an error message to be displayed under *Windows NT* and the host executes properly.

W16TDB (FS:[0Ch]) is basically random under *Windows 95*. The TIB is directly accessible without calling any particular API. That is one of the simplest ways to get a random number. No additional code is necessary – that would be difficult to mutate. (Using port commands would be an option, but again that would be incompatible with *Windows NT/2000*.)

The basic scheme of the first polymorphic block is the following:

```
MOV  reg, FS:[0C]
AND  reg, 8
ADD  reg, jumptable
JMP  [reg]
```

Actual garbage instructions are inserted into this while some of the essential instructions are mutated to various forms. Any register can be used to hold the 'reg' value and make the calculation. Control redirects through the calculated pointer.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

The problem is obvious for emulators. Without having the proper value at FS:[0Ch] the virus decryptor will not be reached at all. Basically it is a matter of complexity and the detection of such viruses could be extremely difficult even with virus-specific detection methods. Heuristics can easily fail to detect such a tricky virus.

(v) Use of Undocumented CPU Instructions

Although there are not many undocumented *Intel* processor instructions, there are some. Win95/Vulcano uses the undocumented SALC instruction in its polymorphic decryptor as garbage to stop some of the processor emulators of certain AV engines that are unable to handle it.

Some emulators' implementation of these instructions may differ subtly from the processor's. Thus, a virus could differentiate between executing under emulation and executing 'normally'. Emulators should not stop when encountering unknown instructions. However, if the size of the actual opcode is incorrectly calculated, the dynamic heuristic scanner misses the virus.

(vi) Use of Brute Force Decryption of Virus Code

Some viruses such as Win32/Crypto use a brute-force algorithm (also known as RDA) to decrypt themselves. This method was known in old DOS viruses such as the Spanska family and some other old, Russian viruses such as the RDA family. All these old tricks have been recycled over the last two years.

The brute-force decryption does not use fixed keys but rather tries to determine the actual method and the proper keys by trial and error. This logic is relatively fast in case of real-time execution. However, it generates very long loops causing zillions of emulation iterations and that way the actual virus body will not be reached easily. The decryption itself can be suspicious activity and modern dynamic heuristics can make good use of these tricks. RDA is a typical attack against emulators that, unfortunately, is very effective.

(vii) Use of Multi-Threaded Virus Functionality

This idea of virus writers does not work very well yet. Many viruses tried to use threads in order to give a hard time to emulators. This idea could be very effective against heuristic scanners in combination with entry point obscuring (inserting) techniques.

Emulators were first used against DOS applications. DOS only supported 'single-threaded' execution and that was a much simpler model for emulators than the multi-threaded model. Emulation of multi-threaded *Windows* applications could be very challenging since the synchronization of various threads would be crucial. Virus writers will certainly try to use this anti-emulation trick as emulators become stronger.

(viii) Use of Interrupts in Polymorphic Decryptor

Like some of the old DOS viruses a few *Windows 95* viruses use the trick of random INT instruction insertion in their polymorphic decryptor. Some old generic decryptors might stop emulating the program when the first interrupt call is reached. This is because most encrypted viruses do not use any interrupts before they are completely decrypted. Win95/Darkmil adopted the very same attack – it uses INT calls such as INT 2B, INT 08, INT 72, etc.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

(ix) Use of an API to Transfer Control to the Virus Code

The Win95/Kala.7620 virus was one of the first to use an API to transfer control to its decryptor. The virus writes a short code segment into the code section of the host application. This short code makes a call to the CreateThread() API via the import address table of the host program. The actual thread start address is specified in the last section of the host program that is created by the virus. Obviously, the virus code will not be reached with emulation if the CreateThread() API is not emulated by the scanner.

Modern anti-virus software needs to address this problem by adding support for certain APIs whenever necessary. It is crucial to have API emulation as part of the arsenal of the AV software. Over the last couple of years we expected to see viruses using random API calls in their polymorphic decryptor, similar to the technique described in the previous section. It is very likely that such techniques will be introduced in the near future since Win95/Kala.7620 and similar viruses already use the basics of such a technique.

Viruses That Do Not Change Any Section to Writeable

It is a very useful heuristic to check if a code section is also marked to be writeable – particularly the entry point section. Some of the viruses do not need to mark any section to be writeable or do not necessarily change any section to writeable. For instance, Win95/SK decrypts itself on the stack and not at the location of the actual virus body in the last section. Therefore, SK does not need to set the writeable flag on itself. Most viruses that decrypt on the stack will not need to rely on the writeable flag and therefore the virus-infected file will be more difficult to identify.

Accurate PE File Infection

First generation heuristics can check the PE file structure for accuracy. Several first generation viruses modify the PE file structure in a way that corrupts the file. When one tries to execute an infected file on a *Windows NT/2000* system, the system loader will refuse to execute such an application. Unfortunately, 50% of all 32-bit *Windows* viruses are already classified into the Win32 class, meaning that the infection strategy is properly done on more than one major Win32 system. Therefore, this heuristic detection will decline in usefulness against new viruses.

Checksum Recalculation

A possible heuristic is the recalculation of the system DLL's checksum (e.g. KERNEL32.DLL). When the KERNEL32.DLL's checksum is incorrect, *Windows 95* still loads the DLL for execution (unless major corruption is identified, such as too short an image). Win32 viruses that access KERNEL32.DLL will try to recalculate the checksum of the DLL by using Win32 APIs. Others implement the checksum calculation. The actual checksum API does the following[12]:

1. Sum up entire file word by word. Add the carry as another word each time.
(For example: 0x8a24 + 0xb400 = 0x13e24 -> 0x3e24 + 0x1 = 0x3e25)
2. Add the final carry if there is one.
3. Add the file size (as DWORD).

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

Win32/Kriz was the first virus to implement the DLL checksum recalculation algorithm without calling any system APIs. It is more difficult to see an application inconsistency when the checksum is properly calculated. Some other viruses recalculate the checksum for PE files whenever the checksum is a non-zero value in the PE header. Therefore, this first generation heuristic will fail to identify modern Win32 viruses.

Renaming Existing Sections

Some of the first generation heuristic scanners try to check if a known non-code section gets control during emulation. Sections such as '.reloc', '.data', etc. do not contain any code in normal circumstances (unless an on-the-fly packer patches itself to an existing section similarly to a virus). Some viruses change the section name to a random string. For instance the Win95/SK virus randomly renames (1 from 8) the '.reloc' section name to a five character random name.

Thus, a heuristic that would check for the existing '.reloc' section name could not pinpoint the virus easily just by seeing that the base relocations were zeroed out in the PE header. However, in case of SK a new heuristic could be used to see if there is any '.reloc' section name while the base relocation value in the header is zero.

Avoiding Header Infection

First generation heuristics can check if the entry point is right after the section headers. Win95/Murkry, Win95/CIH and a large number of Win95/SillyWR variants use header-infection strategy and can easily be detected heuristically or generically. Some of the modern 32-bit *Windows* viruses avoid infecting the PE header area because of that.

Avoiding Imports by Ordinal

A few early *Windows 95* and Win32 viruses patched the host program's Import Table to include export by ordinal imports. Most of the new Win32 viruses do not use such logic and this heuristic becomes useless against them.

No 'CALL to POP' Trick

Most 32-bit *Windows* appending viruses use the 'CALL to POP' trick to locate their start address for their data relocations. Since first generation heuristics could easily look for that, the virus writers tried to implement new ways to get the base address of the code. Normally the trick looks like as follows:

```
0040601A E800000000     call 0040601F
0040601F 5E             pop si
```

Since, in normal circumstances, code similar to the above is not generated by compilers, the use of E800000000 opcode is a suspicious activity. For instance, Win32/Kriz avoids using E800000000 opcode that could be very useful for a static heuristic scanner that looks for small suspicious string. The trick implemented in the Win32/Kriz virus:

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

```
0040601A  E807000000    call 00406026h
0040601F  34F4          xor al,F4
00406021  F0A4          lock movsb
00406023  288C085EB934ACsub [eax+ecx-53CB46A2],cl
0040602A  0200.         add al,[eax]
```

The call instruction at 0040601A offset will reach a POP ESI instruction. Dynamic heuristics are necessary to see if the CALL instruction points to an actual POP.

Fixing Code Size in Header

A possible heuristic logic is the recalculation of the actual used code section sizes. The PE header holds the size of all code sections. Most linkers will calculate a proper code size for the header according to the raw data size of the actual code sections in the file. Some viruses set their own section to be executable, but ignore the PE header recalculation. Static heuristics can take advantage of that. Unfortunately, some of the Win32 viruses such as the Win32/IKX recalculate the PE header's code section to a proper value in order to avoid heuristic detection.

No API String Usage

A very effective anti-heuristic/anti-disassembly trick appears in various modern viruses. For example, the Win32/Dengue virus uses no API strings to access particular APIs from the Win32 set. Normally, an API import happens by using the name of the API such as 'FindFirstFileA', 'OpenFile', 'ReadFile', 'WriteFile', etc. used by many first generation viruses. A set of suspicious API strings will appear in non-encrypted Win32 viruses. For instance, if we use the string command to see the strings in the virus body of the Win32/IKX virus we will get something like the following:

> Murkry\IKX
> EL32
> CreateFileA
> *.EXE
> CreateFileMappingA
> MapViewOfFile
> CloseHandle
> FindFirstFileA
> FindNextFileA
> FindClose
> UnmapViewOfFile
> SetEndOfFile

The name of infamous virus writer 'Murkry' appears on the list. (Actually, the name of certain virus writers or vulgar words are useful for heuristic detection.) Moreover, we see the *.EXE string as well as almost a dozen APIs that search for files and make file modifications. This can make the disassembly of the virus much easier and potentially useful for heuristic scanning.

Modern viruses use a checksum list of the actual strings instead. The checksums are recalculated via the export address table of certain DLLs such as KERNEL32.DLL and the address of the API

Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

is found. It is more difficult to understand a virus that uses such logic since the virus researcher needs to identify the APIs used. By examining the checksum algorithm of the actual virus the virus researcher can write a program that will list the API strings. Eugene Kaspersky uses this technique to identify the used API names relatively quickly. Moreover, he gives a reference for the use of IDA that can be used directly by the dissassembler[13]. Although this technique is useful, it is virus-specific and cannot be applied in heuristics due to speed issues.

Windows 95 Full Stealth File Viruses

Some of the *Windows* virus writers realized that various scanning techniques had become very strong and thus the chance of a PE file virus becoming in-the-wild was reduced. The Win95/Smash virus was the first virus found in the wild in Russia using full stealth file techniques. When opening an infected file and comparing it to a clean one, there is virtually no difference between the two as long as such a virus is active in the computer's memory. This technique is the strongest possible attack that can be made against scanners and the strongest reason to implement memory scanning for Win32 platforms.

2 THE PROBLEMS OF PE FILE REPAIR

Disinfection is a mandatory feature of modern anti-virus software. For years, some anti-virus developers tried to follow the 'detection without repair' strategy. However, macro viruses changed this concept forever. Repair of infected documents is crucial for everybody and, fortunately, it can be done very efficiently. Some of the more advanced anti-virus software can remove unknown macro viruses from documents without dumping all the macros from the file (overwriting the user-created macros if there are any). Thus, we can say that almost perfect repair can be achieved in cases of most macro viruses that infect documents. So far, there have only been a few attempts by macro virus creators to develop viruses that infect existing user macros. This is the reason why macro repair can be done very safely. Unfortunately, this is not the case with PE file viruses. In this section, I introduce the basic problems of PE file repair.

PE file viruses modify a lot of different parts of the executable image. In many cases a particular virus saves the necessary information to perform a 'perfect repair'. A perfect repair would make the repaired object identical byte-for-byte to the original clean copy.

This perfection cannot be reached with the big majority of PE viruses, because several viruses neither infect the files perfectly nor save enough information for the repair. Here are a few examples:

1. The virus modifies the section characteristics of a section (usually the last) to include new flags such as executable and writeable. The big majority of the viruses do not save the original section header anywhere and therefore the modification is not reversible. It is impossible to know which were the original section flags.

2. The virus appends itself to the end of the file, but does not pay attention to the actual raw data size of last section and overwrites part of the last section (that is usually the zero-filled area) with its own code. Such an executable will have a

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England. Tel +44 1235 555139.** No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

different size after the repair. Furthermore, the header information of the file will reflect the change, because the total size of all executable sections is listed there.

3. The virus overwrites some unused area of the application. For instance, some viruses overwrite 0xCC, 0x90 (NOP) or zero-filled areas of the PE executables. It is impossible to know what byte sequence was overwritten in such cases. Thus, the repair will very likely use zeros instead of the original pattern.

4. The calculated checksum will be different from the original if a single byte was different after the repair.

5. The virus writes its self-recognition ID into the PE header and does not save the complete PE header into its body (that is not mandatory for the virus at all.)

6. Some viruses manipulate the relocation entries of the file without saving these entries anywhere.

7. The virus overwrites the relocation section completely with its own code and turns the relocations off in the header. The section will be virtually empty when the repair is done.

Unfortunately, in the vast majority of cases some of the above problems appear. PE repair needs to assume that the file will work after the repair even though the file could be 10 KB shorter. There is simply no guarantee that the actual repaired file will run identically to the original application. Regardless of this, users want to have repair since they often do not have backups.

Are there any major pitfalls with inexact PE file repair? Yes. With the growing number of PE file-based worms and Trojans, the chance that a worm or a Trojan program becomes infected with another virus is major. A system that runs a virus will much more likely get another infection also. This happens when the machine does not run an anti-virus product in the first place.

Imagine the very possible scenario of Win32/Ska.A and another PE file virus – Win32/Kriz – on the same machine. Sooner or later Kriz will infect the PE file (the actual standalone EXE) that is distributed by Ska. Shortly after that the user will notice that something is going wrong with the system and run a virus scan. Since the image of Ska is infected with Kriz two things can happen:

1. The scanner does not know about Kriz and misses it. It will very likely miss Win32/Ska also. This is because the code of the worm was changed. (Do not forget the fact that some viruses modify the code section of PE executables too, potentially changing all the areas where the identification of the virus was made.)

2. The scanner knows Win32/Kriz. The user tries to repair all the infected images at that point and disinfects all the images. Due to the inexact repair, the Ska executable will be changed. In this situation the scanner can easily fail to detect Win32/Ska. In principle, the resulting executable will spread as a new virus. Its binary was changed slightly but some products that use CRC-based identification (that includes the header information) will miss it.

It is recommended to use PE header information-independent detection for such viruses. The PE header itself will very likely change during virus replications. This is a rare case (when exact identification itself is not practical) and should not be used without additional generic detection.

Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

3 THE IMPACT OF WINDOWS 2000 ON VIRUSES

At the 1999 *Virus Bulletin* conference Darren Kessner explained some of the new features of *Windows 2000* that could potentially be used for malicious purposes by virus writers[14]. Some of the new features (e.g. *IntelliMirror* or *Microsoft Installer*) still wait for virus writers to discover them in the near future. Although the new features make it easy to create new virus types and spreading mechanisms, some of the old 32-bit *Windows* viruses failed to work on *Windows 2000* release version. In this section, I review some of the basic incompatibility problems.

Windows 95 Viruses

Most *Windows 95* viruses will fail to work on *Windows 2000* completely. As of May 2000, 50% of all 32-bit *Windows* viruses were classified as 'Win95', meaning that they only work properly on a *Windows 95/98* system. Most of the *Windows 95* viruses that have the potential to spread were developed with the use of VxD functions.

The VxD driver model is not supported under *Windows 2000* (just like under *Windows NT*). Thus, approximately 50% of all 32-bit *Windows* viruses will not effect someone's system if the machine is *Windows NT*- or *Windows 2000*-based. An upgrade from *Windows 95/98* to *Windows 2000* will give this benefit as well as several *Windows NT/2000* security features.

Win32 Viruses

Half of all 32-bit *Windows* viruses are classified as 'Win32'. These viruses are able to replicate under at least two major Win32 systems. *Windows 2000* is an *NT*-based system with a number of major enhancements. Thus, someone might believe that all viruses that worked under *Windows NT* would still work under *Windows 2000* also. Another common misbelief is that *Windows 2000* is so special that virus writers need to write completely new viruses to support it.

There were announcements from several anti-virus vendors related to the W2K/Installer virus. The virus was quickly labelled as 'the only virus able to work on *Windows 2000*'. In fact, the virus used the SFC feature (explained later on) of *Windows 2000* but it was not mandatory for its replication routine. The virus limited its spreading to *Windows 2000* by checking the OS version. Otherwise, it was little more than a Win32 virus that happened easily to work on other Win32 operating systems.

This is why it was interesting to see if there were any Win32 viruses that did not work on the new *Windows 2000* release version. It turns out that 25% of all Win32 viruses (half of all 32-bit *Windows* viruses) can not work on the release version of *Windows 2000*. This happens because of small incompatibility problems that appear in those viruses that were created in assembly.

(i) KERNEL32.DLL Base Address

Several Win32 viruses search for the loaded KERNEL32.DLL in the process address space at various locations. Many Win32 viruses do not search the complete process address space but check the 'MZ', 'PE' sequence at the known KERNEL32.DLL base addresses. The common location of the KERNEL32.DLL is at 0x77F0000 under *Windows NT*. *Windows 95/98* uses a

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

higher address since the DLL needs to be loaded to the shared memory address space. That is 0xBFF70000 for both versions. Some of the viruses checked for the loaded KERNEL32.DLL at that address in order to identify the addresses of all APIs they need to call.

Early *Windows 2000* betas (*Windows NT 5.0* at the time) used very different KERNEL32.DLL base addresses for almost every release. For instance, the beta 1 release used the address 0x77EF0000; that was changed to 0x77ED0000 in beta 3. The RC2 version used the 0x77E80000 address. Not surprisingly, many viruses work on the RC2 version of *Windows 2000* only but fail to work on the release version. This is because the final version specified the address as 0x77E00000. Those viruses that check for the loaded DLL at one wrong location will fail to work. Viruses such as Win32/Cabanas family do not pay attention to the moving DLL base address and fail using that method. However, Cabanas uses more than one method to get the addresses of APIs. If the actual host application has an import to GetModuleHandle() and GetProcAddress() APIs, the virus is able to replicate to other files easily. Viruses that use only one method will fail. They make up almost 25% of all Win32 viruses.

(ii) System File Checker

Windows 2000 introduced the System File Checker (SFC) feature. The SFC uses two directories under the WINNT folder, 'Driver Cache\I386' and 'SYSTEM32\DLLCACHE'. The 'Driver Cache\I386' directory contains a CAB file called DRIVER.CAB. This file is an archive of all the *Microsoft* drivers for *Windows 2000* as well as other crucial system components.

The 'DLLCACHE' directory contains other DLLs and executables such as NOTEPAD.EXE or CALC.EXE. The directory might not mirror all the applications. A limit is specified according to the drive's free disk space during basic installation of *Windows 2000*. For a large disk, the DLLCACHE mirrors almost every standard application and DLL.

The WINLOGON process is always loaded on *Windows NT/2000*. WINLOGON was selected to contain the SFC extension. When WINLOGON starts, it registers a directory change notification request callback function of its own to the system. Thus, whenever certain directories' contents change, WINLOGON receives a notification. Then WINLOGON looks for the change. It seems SFC uses a catalogue of cryptographic signatures of all system files. In case of a hash difference SFC will use the DLLCACHE or the DRIVER.CAB file to replace the modified file silently.

If an original copy of the file is not available under the SFC directories, WINLOGON generates a message box requesting a CD that contains the installed *Windows 2000* version. The changed files are copied from the CD and the files to be replaced are overwritten automatically. SFC compares the file contents in other ways.

The version information seems to be a key element too. Closely related system component versions might be replaceable by presenting a new copy of the driver or executable in the SFC directory first. A newer version can overwrite the existing copy. Clearly SFC's primary purpose is not virus protection. However, SFC can be used even from the command line and it generates 'information logs' that can be viewed with Event Viewer – see Figure 1.

Some viruses will fail to work completely because of the SFC feature of *Windows 2000*. For instance, the Win32/Kriz virus tries to create an infected copy of KERNEL32.DLL first in the SYSTEM32 directory. During boot time that newly created file would replace the old one. However the SFC will not let the modification remain. When the machine boots, the

Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

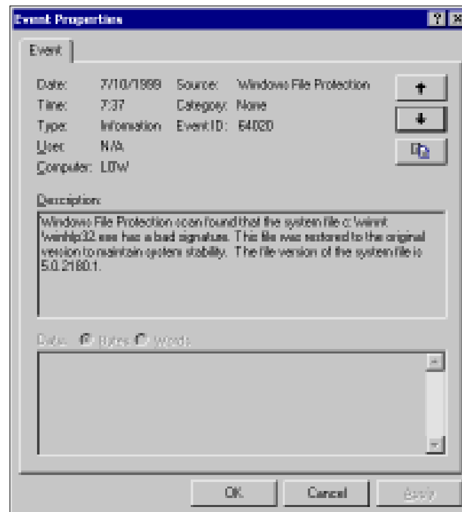

Figure 1: SFC Event Log Entry

Win32/Kriz-infected DLL file is gone and KERNEL32.DLL is not replaced. This is because SFC pays special attention to kernel components and checks their integrity prior to anything else.

Actually, SFC does let the modification happen. It does not try to prevent the modification in any way. It tries to replace the modified file with the old copy. This is a special way of protection from installation software that replaces certain DLLs or EXE or SYS files with a different incompatible version. This problem is known as ‘DLL hell’. SFC was not developed against viruses and virus writers might find ways to fool it via different methods.

However, SFC has a certain benefit against the majority of Win32 viruses that modify files in the WINNT folder whenever they have the necessary rights to do so. If the SFC-related components are protected with standard system security such as file protection against writes, only Kernel mode *Windows 2000* viruses could challenge SFC.

It is difficult to notice this automatic backup system. When a Win32 virus such as Win32/MIX is executed under *Windows 2000* a very noticeable disk activity will follow the execution of the virus. This is because the virus infects new executables and SFC starts to get the modification notices and tries to replace the files with original copies. As long as all the copies of original EXEs are available, *Windows 2000* will be able to work silently without displaying a single warning about the fact that SFC was used (zero user administration).

When one checks the task list during this time, WINLOGON shows very high processor usage. This is because WINLOGON copies the files from the mirror directories one after the other. This entails the copying of many megabytes of data on the disk, which is obviously noticeable. It would be better to have an optional warning message, say after the first ten modified executables were changed!

It is strongly recommended for System Administrators to look into the Event logs very frequently. Personally, I believe that it could be a better choice to put this feature under the ‘Security Log’ instead. The standard ‘Information’ message might not appear to be important but it could be a sign of virus infection. Since non-standard applications are not involved in SFC in

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

any way, those executables that do not have cryptographic signatures are not protected.

Some of the newer viruses such as W2K/Installer, Win32/Dengue and Win32/CTX do not infect the executables if the executable was found to be SFC-protected. This is because virus writers can use the SfcIsFileProtected() API exported by SFC.DLL to check if a particular file was protected and avoid infecting it.

SFC does not stop the spreading of certain viruses at all and it is not a virus security feature. However, it makes the spreading of regular PE file viruses less obvious. Virus writers will certainly try to switch off this feature of the system in many ways. In any case, it is strongly recommended to use SFC regularly.

(iii) Which Win32 Viruses work under Windows 2000?

Most of the Win32 viruses work, since a large number of them were created in a HLL such as Delphi or C. Those viruses which use the Import Address Table of the host program will keep working without special modification in the virus code. Several viruses fail just because of the base address of KERNEL32.DLL and the virus writers can fix them easily. Win32 worms are fully functional. It is interesting to note that the *Windows 2000* standard KERNEL32.DLL API set contains a few new APIs that were available only in *Windows 95* or *Windows 98* so far. This means that some new *Windows 95* viruses that could not work on *Windows NT* might work on *Windows 2000* since the API is available in the new Win32 subsystem implementation.

4 BINARY WORMS – IS A MORE GENERIC DETECTION POSSIBLE?

A very obvious problem of Win32 heuristic is the detection of worms. Standalone code (that is not attached to executables in any way) is much more difficult to detect. Infected files often have an extremely suspicious file structure. Unfortunately, most Win32 worms are written in a HLL such as Delphi, C or Visual Basic 5/6 (VB5/VB6).

For instance, Win32/ExploreZip was created in Delphi. The code of the worm is a few kilobytes in source format, but the resulted compiled executable is 210 KB! Even for a human it takes a few hours work to analyse the code in detail. Delphi applications are particularly difficult to understand, but utilities such as IDA can provide excellent help.

Worm development is still in a very early phase. Even though the logic of some of the worms are very similar to each other, their actual binary code is extremely different. For heuristic detection, one would need to understand what the program does. Often the executables have suspicious imports for various mail related DLLs, but this information is not too meaningful by itself.

Currently only one type of Win32 worm has more developments. Visual Basic-written and compiled worms have many different variants that mostly use MAPI[15] to replicate. Unfortunately, there are many different versions of Visual Basic. Versions 3, 4, 5 and 6 have been used by worm and Trojan creators many times. This is unsurprising – VB applications are as easy to create as *Office* macros and so it is likely that many binary worms will be created in them.

Fortunately, decompilation of VB applications is not impossible. It is, however, a great deal of work. This is because the file formats of VB executables vary from version to version. More generic file heuristic detection could be developed once the product could decompile the de-

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England. Tel +44 1235 555139.** No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

tected VB applications and analyse them. Such a heuristic could be developed, but we need to understand the VB file structures in detail, and there is no documentation of any kind available. Generic worm detection needs behaviour blocking solutions. Virus detection needs to be done more proactively.

5 CONCLUSION

Heuristic scanning should not be hard-coded in anti-virus products. Heuristics need to be updated from time to time to challenge the virus creators and make their job more difficult. I am positive that most of today's viruses' anti-heuristics features will become the best heuristics against future viruses.

Heuristic scanning will take on more importance as viruses grow more complex. It is very unlikely that we need to add exact identification of all the upcoming thousands of viruses. Heuristic and generic detection can relieve the load that virus researchers are facing in the coming years. So, can we conclude that heuristics form the ultimate solution to all virus problems? No[16].

Heuristics need to be developed further, because the virus situation is moving from one platform to another in a matter of a few years. AV researchers need to update heuristic and generic detection continuously. Heuristics do not form an ultimate solution, but definitely work better in the long run compared to other scanning techniques.

Modern virus writing moves from the decade of computer viruses to the decade of the network worms. Win32 viruses and worms will challenge anti-virus researchers, system administrators, and users. It is interesting to note that HLL worms become more and more popular and successful. Heuristics did not need to deal with such viruses since traditional HLL viruses were extremely easy to notice. Unfortunately, networked worms revise many of the unwritten rules of traditional virus research.

Be prepared for the upcoming thousands of viruses and educate your users as much as you can about them. Education may work better once people have learned about bad things the hard way.

ENDNOTES

1 Péter Ször, 'Attacks on Win32', *Virus Bulletin*, April 1998, pp.57–84.

2 Péter Ször, 'Happy Gets Lucky', *Virus Bulletin*, April 1999, pp.6–7.

3 Dr. Steve R. White, 'Open Problems in Computer Virus Research', Proceedings of the Eighth Annual *Virus Bulletin* Conference, 1998, pp.277–287.

4 Andy Nikishin, 'Inside Infis', *Virus Bulletin*, November 1999, p.8.

5 Péter Ször, 'Nexiv_Der: Tracing the Vixen', *Virus Bulletin*, April 1996, pp.11–12.

6 Eugene Kaspersky, 'Don't press F1!', *Virus Bulletin*, January 2000.

7 Dr. Igor Muttik, personal communications (1997).

8 Snorre Fagerland, 'Merry MMXmas!', *Virus Bulletin*, December 1999, pp.10–11.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England. Tel +44 1235 555139.** No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.

9 Péter Ször, 'Coping with Cabanas', *Virus Bulletin*, November 1997, pp.10–12.

10 Kurt Natvig, personal communication (1999).

11 Péter Ször, 'The Invirsible Man', *Virus Bulletin*, May 2000, pp.8–9.

12 Wason Han, personal communications (1999).

13 Eugene Kaspersky, personal communications (1998–2000).

14 Darren Kessner, '*Windows 2000* and Malware', Proceedings of the Ninth *Virus Bulletin* Conference, 1999, pp.125–132.

15 Carey Nachenberg, 'Computer Parasitology', Proceedings of the Ninth *Virus Bulletin* Conference, 1999, pp.1–25.

16 Righard Zwienenberg, 'Heuristic Mania', Proceedings of the Sixth *Virus Bulletin* Conference, 1996, pp.129–132.

VIRUS BULLETIN CONFERENCE **©2000 Virus Bulletin Ltd, The Pentagon, Abingdon, Oxfordshire, OX14 3YP, England.**
Tel +44 1235 555139. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form, without the prior written permission of the publishers.