

# VIRUS ANALYSIS 1

## High Anxiety

Péter Ször  
Data Fellows

The *Windows 95* platform is becoming an increasingly obvious and attractive target for virus writers – the number of different ways to implement working *Windows* viruses appears to be virtually endless. The latest variation of this trend is Win95.Anxiety, an unoriginal, slightly modified variant of Win95.Harry. Anxiety fixes a few of Harry's small bugs, which is why it is more successful, but some of the original release's fatal bugs still remain. So far, Anxiety is in the wild in Germany, Finland, Holland and the USA. Thus, it seems reasonable to assume that the source of the original infection is related to some FTP site.

Win95.Anxiety infects PE (Portable Executable) programs under *Windows 95*. Further, it can hook the IFS (Installable File System) without having a VxD dropper tailor-made for this purpose. Viruses such as Punch and Memorial are complex because they solve the problem of file system hooking with a specific VxD dropped by the PE part of the virus code and then have to arrange for the VxD to be loaded somehow. This makes them more complicated to write, although virus writers seem to have been optimizing the technique. Anxiety takes a quite different approach; that of patching its code into the Virtual Machine Manager (VMM) of *Windows 95*.

### Executing the Virus

When an infected PE program is executed, Anxiety takes control. Programs are executed at *Windows 95*'s application level, so they cannot perform system level functions in the way a VxD can. Anxiety bypasses this inconvenience by installing its code into the VMM, which runs in Ring 0.

Anxiety's installation routine searches for a large hole in the VMM's code area, above the address 0C0001000h. If a large enough area (consisting only of FFh bytes) is detected, the virus looks for the VMM header at 0C000157Fh, and checks the area by comparing it with 'VMM'. It saves the address of the Schedule\_VM\_Event system function from the VMM for later use, copies its code into the previously-located hole in the VMM, and changes the Schedule\_VM\_Event address to point to itself. Finally, Anxiety executes the original host program by jumping to its original entry point.

### Hooking the IFS

Before the host program can be executed, the VMM will call Schedule\_VM\_Event, causing the virus' initialization routine to run. As this code is executed in Ring 0, it is able

to call VxD functions. Anxiety hooks the IFS by calling IFSMgr\_InstallFileSystemApiHook from its initialization code. This installs the new hook API, and after that something peculiar happens.

The code here looks as if it opens and closes a file and calls some registry functions (with invalid parameters!). At first glance it looked like very tricky stuff, until the similarity to Win95.Harry was noticed. Anxiety actually contains some dead code from Harry. One of the areas in which the latter virus is not very successful is its activation routine. Harry creates a cursor image file called C:\SYRINGE.CUR and tries to activate it by modifying the registry. Usually Harry crashes at this point.



It could be that Anxiety was modified to prevent the execution of the risky instructions involved in the above procedure. Anxiety does not have a completely new activation routine, but the modified virus is able to replicate under most *Windows 95* environments without Harry's original problems.

After hooking the IFS, Anxiety resets the address of Schedule\_VM\_Event to point back to the original, and jumps to Call\_VM\_Event. The original host will be executed by *Windows 95* a second later.

### Infecting PE Programs

Once Anxiety has successfully hooked the file system, it waits for file-open calls. During these, the virus converts the file names with the UniToBCSPPath function and checks their extensions. After ensuring the file extension is EXE and that it is in PE format (by looking for the 'PE00' marker), Anxiety opens the file in write mode.

It then reads to the last section header and checks for an existing infection by comparing the seventh byte of the section name with FFh. Usually the last section name is .reloc (followed by a null, 00h). If the program is not considered to be infected, Anxiety patches the section header's other fields to fit into it. The same technique is used by Win32.Cabanas (see *VB*, November 1997, p.10) to make infection less risky and to reduce the chance of detection by heuristic scanners. Then, in a complicated procedure, the virus adds its code to the image.

When VxD code is executed, calls are patched by the VMM. This turns CD20h, <function id> (Int 20h, <function id>) into FAR CALLS. Some of the VxD functions consist only of a single instruction. In those cases, the VMM patches a further six bytes to make the single instruction fit. The VMM does this dynamically with all executed VxDs to speed up their execution. This on-the-fly VxD function

patching means the virus is unable to copy its image immediately to files, since those applications would not work in a different *Windows* environment. Anxiety has a function which patches all its VxD functions back to their normal format, and only then will it save its code into the host program. Various PE header fields are modified to reflect the host's infected state. Finally, the characteristics of the last section are set to MEM\_EXECUTE and MEM\_WRITE, then Anxiety closes the host program.

### The Bug

Unfortunately (from the disinfection point of view), Anxiety has the same problem as Harry. When the virus infects a file, it overwrites part of the original program, usually containing zeroes, because of the section alignment. This makes disinfection difficult and often impossible. Programs ending in code will not work after infection and cannot be repaired by a disinfectant. Most of the time, however, the application ends in a long zero-filled area, and the virus works without any noticeable problems, making disinfection possible. The text 'Anxiety.Poppy.95 by VicodinES.' is viewable in the code, but never displayed.

### Conclusion

The number of different techniques used by virus writers is growing as quickly in the *Windows* environment as it did in the early days of DOS viruses. The most successful infection methods (introduced by the 'pioneers') will eventually become the 'standard' infection techniques of the next century. Although this 'standard' is not ready yet, it will be finalized during the next few years. As *Windows* virus writers share source code with each other, buggy viruses can currently be fixed by any of those who introduced them into the wild.

## Win95.Anxiety

Aliases:	Win95.Harry.B.
Type:	<i>Windows 95</i> PE infector.
Self-recognition in Files:	FFh at offset 7 in the last section header's name area.
Self-recognition in Memory:	Not needed.
Hex Pattern in PE files:	2BFF BF00 1000 C0B8 FF00 0000 B9FF FFFF FFF2 AE8B D90B C90F 8480 0000 0081 FF00 C000 C073
Intercepts:	Hooks IFS API OpenFile.
Payload:	None.
Removal:	Recover infected files from backup or replace with originals.

## VIRUS ANALYSIS 2

### DarkParanoid – Who Me?

*Eugene Kaspersky*

*KAMI Associates*

Computer viruses are not going to disappear. Once, we anticipated the emergence of new, protected, virus-free operating systems – an era when DOS and all its viruses, would perish. It seems we were right – DOS is becoming obsolete as an independent OS, and maybe in the near future (a century or so) it will be replaced. We were wrong too. Modern operating systems have virus-protection levels similar to those of good old DOS, i.e. they have *no* protection. Viruses have now been written for all popular OSes, including *Windows 95*, *NT*, and *OS/2*.

DOS viruses are still showing signs of development. They contain honed versions of old ideas like polymorphism and stealth, and some new tricks. Their writers may be readying these techniques for inclusion in non-DOS viruses.

There are several known tricks that viruses use to hide their code in both files and memory. The most popular is encryption, where encrypted code is decrypted when necessary. Some viruses employ several methods of encryption, including 'on-the-fly' in memory encryption, where subroutines are decrypted before execution and encrypted after it. Despite their different tricks and encryption algorithms, it is true of such viruses that at some point either their complete code, or major subroutines, are decrypted. This is not the case for the new, polymorphic virus, DarkParanoid.

### Encryption

This virus uses an ultra-complex method of 'on-the-fly' encryption. At any time, only one instruction is in unencrypted form (apart from the en/decryptor code). The virus manages this by using tricks with Int 01h tracing mode. When the virus first receives control, it hooks Int 01h. Subsequently, the execution of almost all instructions causes the Int 01h code to be invoked. After executing the current instruction, DarkParanoid takes control with the Int 01h hook, encrypts the current instruction and decrypts the next one. Thus, at any given moment, either all of the virus code is encrypted, or just one instruction is clean. Moreover, the virus encrypts/decrypts the code of previous/next instructions imprecisely, as bytes/words at several offsets from the current instruction's address.

Three blocks of code cannot be encrypted at the same time – the Int 01h handler (en/decryptor), the start code in infected files, and the Int 21h handler. Both the start code and the Int 21h handler hook Int 01h on receiving control, then switch to tracing mode, passing control to the installation routine and back to the Int 21h handler, respectively.