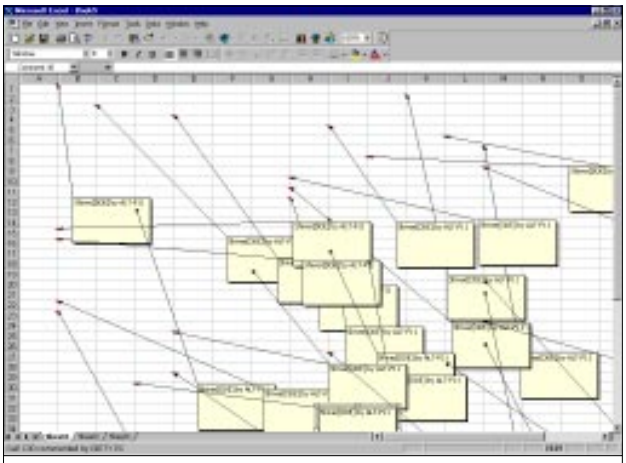

Fig 4: The visual mess produced by the *Excel* payload is easily corrected and, fortunately, no data is altered.

All three variants have the same *Excel* payload. On opening an infected spreadsheet, with one in 800 probability, Shiver attaches a comment to thirty randomly-selected cells in the top left 30 x 30 block of the active sheet (see Fig 4). No data is lost, but the sheet looks seriously messed up. This is rectified by applying Edit/Clear/Comments to the top left 30 x 30 corner of the spreadsheet.

Conclusion

It seems likely that Shiver is the precursor to a bunch of multi-application viruses using a similar approach. The use of DDE (and other VBA goodies, as yet undiscovered by the virus writers) is a much more elegant and portable cross-infection method than the machinations employed by both Cross and Teocatl. This method seems likely to be extended to more applications. It can now only be a small step to polymorphic cross-application macro viruses.

Shiver

Aliases:	None known.
Type:	*Word 97/Excel 97* cross-application macro infector.
Self-recognition:	Potential hosts that contain a VBA module named 'Module1' are assumed infected. If the value of Shiver[DDE] in the Registry key HKCU\Software\VB and VBA Program Settings\Office\8.0 is not 'ALT-F11', cross-application infection is initiated.
Payload:	Several, triggered randomly (see text).
Removal:	Delete NORMAL.DOT, WORD8.DOT and PERSONAL.XLS. In a clean *Word* environment, delete Module1 from infected documents via the Organizer. There is no simple, manual procedure for disinfecting spreadsheets.

VIRUS ANALYSIS 2

Breaking the Lorez

Péter Ször
Data Fellows

These days, writing *Windows 9x* viruses is not as difficult as it was a year ago. More and more *Windows 9x* viruses are documented, complete with source code, in virus writer magazines. Those sources can be used as 'study guides' in the creation of other viruses, which certainly simplifies the writing process. Nevertheless, it is still early days for *Windows 9x* and *NT* viruses, and new techniques are often to be found in the latest creations.

Once Upon a Time

Early *Windows 95* viruses were direct action infectors, hence not going resident. As they often caused a noticeable slowdown during infection, virus writers soon started looking for more effective ways of implementing fast infection. The most challenging problem of *Windows 95* viruses became the question of TSR mechanisms.

Creating an active virus is not an easy task and despite their complexity, early *Windows 95* viruses evolved swiftly. Punch (*VB*, April 1997, p.8) and Memorial (*VB*, September 1997, p.6) used specific VxD droppers, the result of which were full VxD-based virus bodies. This made the virus code complicated, because it had to be converted into different file formats. Just a couple of months later we saw viruses capable of direct VxD and *Word* document infection, such as Navrhar (*VB*, November 1997, p.15).

Soon after that, Anxiety (*VB*, January 1998, p.7) simplified things by patching into the VMM directly. Then Cabanas (*VB*, November 1997, p.10) introduced a per-process resident strategy by hooking the imports of host programs (so far this is the only method which also works under *NT*). Last but not least, HPS (*VB*, June 1998, p.13) demonstrated VxDCall hooking by patching KERNEL32.DLL's local heap, and installing itself in shared memory.

Most *Windows 95* viruses are Portable Executable (PE) infectors, although some infect DOS COM and EXE programs, VxDs, *Word* documents and 16-bit *Windows* New Executables (NE) as well. Others may accidentally infect dynamic-link libraries (DLLs) which are linked in PE (or NE) formats. In these cases, the infection is unable to spread further because the 'standard' entry point of a DLL, which such accidental infections usually intercept, is not called by the system loader. Normally, a DLL's execution starts at its specified DLLEntry-point.

Win95/Lorez is a PE infector that targets KERNEL32.DLL with a new attack. Instead of modifying the entry point, it patches the export RVA (Relative Virtual Address) of a

critical API to point to the virus code at the end of the DLL. But how is it able to replace the original KERNEL32.DLL with an infected one?

Executing Lorez

The virus code gains control when an infected program is executed. The entry point of the infected program points to the first byte of the virus body. At first Lorez tries to determine the OS version, checking the stack for the KERNEL address and calculating the base value of the return address. This new trick seems likely to be used in the future, as it simplifies the infection process.

Lorez tries to infect in cases where the base vlaue is BFF70000h (*Windows 9x*) and 77F00000h (*NT*), or it terminates by executing the host program. It tries to install itself under *NT* too, but perhaps the virus writer did not have the time or equipment to test the virus under any system other than *Windows 95*? Some of the code's assumptions are wrong and for this reason the virus is unable to work under *NT*. However, it works under *Windows 95* without any major side effects.

Next, the virus obtains the addresses of the KERNEL32 APIs it needs to call. It also acquires the address of the ExitProcess API, which is used not by the virus but by the original host program. These addresses are stored in a DWORD table at the end of Lorez' code for use by its own 'GetProcAddress' function. Its attention is now turned to infecting KERNEL32.DLL.

KERNEL32.DLL/EXE Infection

First, Lorez obtains the *Windows* and system directories using standard APIs. Normally, KERNEL32.DLL is in the system directory and cannot be written to when it is running. It can, however, be read so the virus copies it to the *Windows* directory. Then it sets a kernel infection flag for itself, before calling its standard PE infection routine.

The infection routine is passed the full path and name of the file as a parameter. It calls the GetFileAttributesA function, setting ECX to 12345678h before the call. This is a check for its presence in memory – Lorez' GetFileAttributesA routine simply terminates if ECX is 12345678h. If that happens, KERNEL32.DLL infection is not necessary. However, if the function returns the correct attributes and no error, Lorez assumes it has not already hooked the API, so it saves the original file attributes and changes them to archive, thus allowing the file to be written to. The copy of KERNEL32.DLL is opened for infection and the DLL's original time/date are saved (including the creation time/last access/last written to fields) for later use.

Lorez then checks if the file is an EXE by looking for the 'MZ' marker. After that, it moves to the PE header to check the PointerToSymbolTable field. If this is non-zero it aborts the infection. This field is zero in most PE files (including the original KERNEL32.DLL) and is usually only used in OBJ files and PE files with COFF debug information. At the completion of infection, the virus sets this field to a random value to avoid re-infection.

If the host seems clean, Lorez checks the kernel infection flag. Since this is set during initialization, the virus skips the Base Address check. The Base Address of the executable has to be 40000h in normal EXEs – Lorez does not infect the rest. This is a simple, effective way to avoid infecting nonstandard applications.

Next, Lorez gets a 'random' number from GetTickCount to use as an infection marker in the PointerToSymbolTable field of the PE header. The size and attributes fields of the final section are changed to reflect the increased size and executable status. If infecting KERNEL32.DLL it determines the location of GetFileAttributesA, otherwise it modifies the PE entry-point. To do the former it finds the image offset of GetFileAttributesA from the export table.

At this point the code does some nonessential checks for the name of the export section in order to eliminate different versions of KERNEL32.DLL for *Windows 95, 98* and *NT*. Despite this, Lorez only works on *Windows 95*. This function eventually patches the GetFileAttributesA export RVA to point to the hook function in the virus code, saving the original so it can jump back to it.

Finally, it writes the new PE header to the host and appends its body to the end of the victim. The size of the virus is 1766 bytes, but the effective length varies because of the section alignment. It resets the attributes and date/time stamp and executes the original host.

Going Resident

There is now an infected KERNEL32.DLL in the *Windows* directory but the original is untouched in the system directory. When the PC is next booted, *Windows 95* will load the infected version. It seems that KRNL386.EXE uses the 16-bit LoadLibrary search logic and thus will load KERNEL32.DLL from the current directory first.It appears that the current directory here is the *Windows* directory while the directory in which the application is executed is the system directory. (The Win32 version of LoadLibrary uses the directory from which the application loaded then the current directory and so on.) For this reason the infected KERNEL32.DLL will be loaded at boot time.

When a DLL that a newly-executed application requires exports from is already loaded, *Windows 95* does not reload the DLL. Instead it attaches the application to the DLL by mapping. When any PE program is executed (for instance, a Win32 anti-virus program) in a Lorez-infected environment, it will be attached to a non-reliable, infected representation of the KERNEL32.DLL.

The new hook function will redirect the call to the infection routine when the application calls the GetFileAttributesA API. The general infection routine is the same during

KERNEL32.DLL infection, but the entry points of the victims are modified to point to the virus code instead of looking for GetFileAttributesA exports there.

Conclusion

As we know, *Windows 9x* is far from secure. *NT* is able to stop Lorez' KERNEL32.DLL manipulation, because the system loader makes several additional checks before it loads the image. Most importantly, *NT* will only load KERNEL32.DLL from the SYSTEM32 directory.

Further, system DLLs contain a checksum in their PE header. Unlike that of *Windows 9x*, *NT's* loader calculates the file's checksum before loading a DLL. If this does not match the recorded checksum, the loader halts during the blue screen boot-up with an error message.

These additional checks do not mean that such a virus cannot be implemented for *NT*, but they do make it more complicated. While the checksum algorithm is not documented by *Microsoft*, there are APIs available for these purposes. Even this is not enough for the *NT* loader – there are several other checks to pass, but it seems prudent to assume that virus writers will solve these problems.

Lorez is based on the Yurn virus and it only works under *Windows 95*. Win32 viruses are already on the road to polymorphism – Lorez's infection technique may be combined with a polymorphic engine in the future. This would lead to scanning problems similar to those caused by polymorphic inserting DOS viruses like Zhengxi and Nexiv_Der (see *VB*, April 1996, p8 and p11 respectively).

Lorez

Aliases: None known.

Type: Windows 95 PE infector, attacks KERNEL32.DLL.

Self-recognition in Files:
If the PointerToSymbolTable field in PE header is non-zero, the virus does not infect. This field is set to a random value during infection.

Self-recognition in Memory:
With ECX=12345678h call function GetFileAttributesA. The virus' handler simply terminates without returning an attribute and no indication of error.

Hex Pattern in PE files:

```
58FF E08B 8557 1740 0050 B978
5634 12FF 95E6 1640 0089 8553
1740 0083 F8FF 7501 C36A 208B
```

Payload: None.

Removal: Replace infected files from backup or from clean originals.

FEATURE

1998 – The Year of the Net?

The last twelve months or so have seen many interesting new directions followed by the virus writers. Perhaps most notable among these, and an area in which we fully expect we will see a lot more development, is the addition of network-awareness to viruses.

In the Beginning…

PCs started out as the small-fry of the business computing world. Much as their growth and success has seen them become the home computer of choice, this largely follows their unprecedented success in the corporate realm. Today we have near-ubiquitous networking. However, in the heady days of the first ten years of the PC's development, networking was not only rare but the most impenetrable of the black arts of computer configuration and support. Those times seem much more distant than the seven or so years that actually separate that era from this.

Prior to this current age of ubiquitous networking – in fact, until quite recently – virus authors have largely depended on fate to distribute the seeds of their labours. Reports from the past show that some boot infectors, and the very occasional file infector, had a 'lucky break', usually being widely distributed on magazine cover diskettes or driver diskettes. Very rarely, infected application software has also been distributed.

In the good old days, 'sneakernet' was the commonest form of PC-to-PC software transmission. Thus, it was quite understandable that boot viruses and multi-partites accounted for most virus infection incidents. Despite some initial trepidation, the initial blossoming of Internet usage saw little or no change to this. Although there were occasional instances of viruses and Trojan Horses posted widely on Usenet, for example, the balance of field infection reports was little changed.

Net Results

Of course, the cost of networking has fallen dramatically, along with the cost of all other PC components and software. Further, the difficulty of setting up and configuring networks has fallen, though not as quickly as prices. One advantage of the PC's popularity has been increased production and lowered cost. Another benefit has been that as network software has become more standardized, it has become easier for software developers to write more universally useful code.

With increasing interest in 'the Net' amongst both end-users and service and product suppliers, the last few years has seen a huge rush of interest in establishing and partici-