VIRUS ANALYSIS 2

Tricky Relocations

Péter Ször
SARC, USA

Last December I came across a bizarre Dynamic Linked Library (DLL) that our product had a minor issue with. It was part of a *Borland Quattro* product and linked with a *Borland Linker*. The Base Address value of the DLL was set to 0x2CC – weird. That value is clearly incorrect. Applications could never load to such an exact address since the file is always mapped from the start of a given page. Furthermore, the address is far too low to be correct. I realized such a file would be incorrect and not accepted by the system loader and never run. I was partially right. The LoadLibrary() function did not load the DLL under *Windows NT*/*2000*. However, *Windows 9x* systems loaded the file nicely. This was well worth testing.

Basically, the *Windows 9x* loader does not check for such a specific case. Therefore, if the application or DLL has relocations (in the .reloc section) the loader will try to relocate the image to an available correct address in the process address space. One more mystery solved.

It crossed my mind that a 32-bit virus could probably use a trick which would force relocation to be used, and then add an item to the ‘to do’ list: ‘Applying relocation items for PE files.’ This trick can be applied at least two different ways. It can either use an incorrect Base Address as mentioned above (this only works under *Windows 9x*) or try to load the image to an address that is not available, since a system DLL is already loaded to the address (which works with limitations under *Windows NT*/*2000*).

What could be the use of such trick in a virus? Relocations were used in the DOS days as an anti-emulation/heuristics method. For instance, the Tentacrille virus used a trick that was based on EXE relocations. It is not difficult to see how a 32-bit *Windows* virus could implement such an approach for a similar reason. ‘It is always good to know something before the virus writers catch on, since we might have the support for it in our engine before they realize the possibility’ I thought, and informed my fellow anti-virus researchers about the problem. Who would think of such a trick? Less than a week later I saw the first virus to use forced relocations. It is obviously a virus created by a 29A member. The actual virus is not really new since it is largely based on the Resur virus – we called this new one W95/Resurrel, indicating the relocation trick.

W95/Resurrel is the first encrypted binary virus that does not implement a decryptor. The virus code is encrypted and runs just fine when the application is executed. It uses the forced relocation trick and lets the system loader decrypt the virus via relocations.

How does Resurrel work?

W95/Resurrel is written entirely in C and 80% of the code is based on the W95/Resur virus. When an infected file is executed, the virus will execute the original host application as a thread. Resurrel can work silently in the background and infect local as well as network drives with their base address value set to 0x00400000. It does not infect DLLs.

The virus has four different sections. It will modify the entry point of the host to point into its own code section. The four different sections of the virus code are patched into the section table if there is enough space in the header. Resurrel is careful not to corrupt the host by overwriting the code right after the section table area. If there is a .reloc section, it will overwrite it. Resurrel uses a mutex set to ‘29A’ in order to run only one active copy at a time. Other executed copies will only run their host program. The virus traverses the directories of each drive and infects files everywhere but the SYSTEM directory.

The virus sets the Base Address of infected files to a DWORD value of 0xBFxxxxxx where xxxxxx is a random value set via the GetTickCount() API. It adds a relocation entry for each DWORD value of its own code section. Each DWORD is encrypted in the following way – the virus adds the new Base Address random value to the actual DWORD of its code section, then subtracts 0x400000 from it. Before being placed into the virus’ relocation section, each entry is set to IMAGE_REL_BASED_HIGHLOW type. Finally, the virus adjusts the necessary field of the PE header and the infection is complete.

When the actual infected application is executed the system loader will try to load the image. The value is either wrong (does not start at a beginning of a page) or simply indicates a load over the KERNEL32.DLL which is placed to the same area of the process address space under *Windows 9x*. Therefore, the system loader will check for the available relocations then it relocates the code completely, thus decrypting the virus code section that is encrypted differently based on the random Base Address values.

Conclusion

A simple string picked up from the code section of the virus will not be sufficient to detect this virus. The virus does not encrypt its other sections in its first release. However, it is better to apply the decryption logic that is simple enough for algorithmic detection. The data section of the virus carries the Win95/SVK by Tcp/29A string and visible in each infected file.

It is time to implement support for relocations for PE infections – other viruses could challenge 32-bit emulators with a similar trick in the very near future.