OPINION

Trouble Makers

Andreas Marx, AV-Test.org
University of Magdeburg

It is a fact that no software – besides a single 'Hello World' program – is completely free of bugs or unintended side-effects. This applies not only to operating systems and *Office* applications, but also to anti-virus software.

Today's anti-virus scanners are very complex pieces of software. A small mistake in the program code or virus definitions can result in a small problem, such as a miss of a non-ItW virus, or a much bigger problem such as a crash – which is particularly serious if the program protects a server or Groupware system. Stability is one of the most important issues here: scan all incoming and outgoing traffic, but please don't crash!

History Lessons

In history we have had a few good examples of such crashes, especially on damaged or corrupted files. I can remember a problem in *Dr. Solomon's* anti-virus solution when scanning a 32-byte-long EXE file: only the normal MZ header was available, but the rest of the file was missing. This caused an exception fault in the command-line scanner.

OLE2 files also caused a lot of trouble in the past. The first macro virus scanners used the *Microsoft* OLE implementation, which worked quite well for standard files, but usually caused problems if the files were slightly damaged. Very fast, vendor-own implementations were developed, which included proper error checks to avoid crashes or infinite loops in the internal OLE file structure.

A few years later, Costin Raiu published an article in *Virus Bulletin* called 'The Little Fixed Variable Constant' (see *VB* October 1999, p.8), which discussed OLE documents having a 4 KB block size instead of the standard 512 bytes. A lot of programs simply ignored the files or did not find any virus, which could be fixed in later releases, because it was only an academic issue. However, at least two programs crashed – and that should not happen.

Usually, such malformed documents do not exist in a (relatively) trusted environment, such as within an organization. However, an attacker can send nearly anything to a company using email. These emails could contain viruses, may be malformed and so on, and their content can never be trusted. Usually, these will be scanned at the email gateway or – if this does not exist – in the Groupware environment, such as *Exchange* or *Notes*. And now the files are in the middle of an important, trusted environment, but still they can contain nearly any surprise.

In the Archives

On the Bugtraq mailing list a few months ago there was a posting about archive files like ZIP or ARJ, which contain files with names such as 'NUL.EXE' or '../NAME.EXE' (see http://www.securityfocus.com/archive/1/196965). The author looked at standard unpackers and found many problems: file names with reserved DOS names such as NUL, CLOCK$, AUX or PRN can cause *Windows 9x*-based systems to crash or simply to print out a file during extraction. *Windows NT*-based systems were not immune, but the trouble was limited.

We investigated how virus scanners would react if they found such a file: only one program crashed out of about 30 tested, but only two thirds were able to detect the viruses inside these files. In particular, this happens if they try to extract the file to disk under the name which is stored in the archive, which is not possible. A random name should be used instead, or the file should be scanned in memory – consider memory-mapped files in *Win32* environments or a RAM disk, for example.

However, this is only a small issue. A more interesting method is to embed files in an archive which contains files with names like '../NAME.EXE'. Such archives cannot be created using standard *Win32* tools, but can, for example, under Unix-based systems.

However, I was too lazy to start *VMware* so put a file in an archive with a name like 'XX_NAME.EXE' instead. Later, I changed the 'XX_' to '../' using a hex editor. It should be noted that this has to be done at two positions in ZIP files (simply use search and replace), but at only one location in ARJ archives. And, of course, more than just one '../' can be used for this – I used it up to six times for a test.

Put to the Test

At first, I tested archive programs and observed that nearly all of them were vulnerable and dropped the archived files in nearly every available subdirectory on disk. Using a virus scanner, the situation was much better: no command-line or GUI version seemed to be affected, all ran fine, found the virus and did not drop the files over the hard disk. Even if the program does not scan the files in memory, it has been extracted to a random file name in a temporary subdirectory, ignoring paths.

Next, I looked at *Exchange 2000* and mail gateway solutions – not only anti-virus, but also content filtering programs. Some of the products I encountered used the standard unzip utilities which are not secure. Sure enough, I was able to send an email with an archive and the files within it were dropped to a special location on the hard disk. Also, I was able to overwrite programs like

explorer.exe on *Windows* systems, because the extract process acts with administrator rights. I shan't continue here, but just point out that it was very trivial to hijack such an insecure system within minutes. Of course, I notified the vendors of all programs about this issue (this was in August 2001).

As a quick fix, I suggested putting the temporary (unzip) directory into another partition or onto a RAM drive. Following this, the systems were no longer affected by the problem.

Following notification of the issue, a few of the vendors responded that this problem is rather academic, but after I sent them our test files inside unencrypted ZIP archives (without viruses, but causing small problems on their own mail Gateway), they realized that it is a real problem. :-)

Heavy Nesting

A few large companies suggested that we investigate what happens if an anti-virus program has to scan a heavily-nested ZIP archive.

I obtained a sample file called '42.ZIP'. This was a ZIP in a ZIP in a ZIP etc. – it had six recursion layers, with 16 new files inside every layer (size was about 2 GB in every case), until I was able to see a file called '0.DLL', that contained only a few random (mostly zero) data. These data can be compressed very easily, which meant that the ZIP file was only 42 KB long.

The companies that had suggested this investigation had encountered great problems at their mail gateways – script kiddies had sent such files by email over and over again. Most virus scanners require about two to three days(!) on an Athlon 1,33 GHz system to scan such a file with 100% CPU load.

Therefore, I suggest strongly that an option should be added to all content security programs to set a time-out value (e.g. 180 seconds) for a file such that, if this time-out is reached, the file will be skipped, treated as a virus and quarantined. Also this would help if the scan process crashes unexpectedly.

An alternative would be to limit the maximum size of an attachment to be delivered, but for our '42.ZIP' example this would be nearly useless, since it is already very small. Finally, the number of recursion layers could be limited to a 'harmless' three or four. The user could select the maximum number of layers in the main program according to their requirements.

Testing Issues

We tested these issues in our last *Exchange 2000 SP1* test (the results of which can be seen on the Web site http://www.av-test.org/). Many programs had a problem with 100% CPU load and in at least one there was an option to limit the scan time, but this feature did not work.

Also, one email gateway scanner was unable to detect Win32/Sircam due to the malformed (non-RFC) headers it uses. The attachment was not found and the email was delivered to the clients.

Following the Win32/Nimda outbreak, many vendors added better detection for EML files in their scan engines, but there was also a problem with 'trusted data'.

One solution I tested internally reserved for memory operations only the number of bytes the mail header showed, which caused a buffer overflow if the attachment was longer than expected. Another program crashed if the attachment was truncated or if the MIME structure was corrupted – this can easily happen automatically due to transportation problems.

I can continue with *Win32* runtime compressors – simply attempt to change a few values in the main compressed files and the decompression routine of a scanner shows unexpected behaviour. With a little information about the compressed file structure and about which special tokens have been used, it should not be a problem to create such a file.

Of course, no scanner can find compressed worms or viruses in such files any more and I do not expect this. The only thing that should not happen is a crash of the scanner, neither as a result of a GPF nor a 100% CPU load problem.

Engine Developers Take Note

My suggestion for engine developers is that they should check all input from files carefully, especially all variables that the program uses internally for pointers, to reserve memory etc. I suggest that program developers include features to limit the damage of problematic files, for example setting a maximum scan layer for archives or a simple time-out.

The internal QA should check the behaviour of the scanner specifically on malformed files. This should include white box testing, where the tester knows the internal structure of the program and tries to find problematic routines, as well as black box testing, where the tester modifies a file with the intention of causing problems to the scan engine. This can be done automatically where random parts of a file which is likely to be problematic will be changed or overwritten in a loop, until a problem occurs.

Testers Take Note

For magazine testers, my suggestion is to include malformed files in a test set. It does not help a user if a scanner finds nearly all infected files, has no false positives, and is fast – but it crashes directly or requires 100% CPU load if it receives a malformed file to scan from a non-trusted source, such as the Internet. Currently, I suggest including only easy malformed files, such as the archive files described above. Later, we can add more problematic issues.