MALWARE ANALYSIS

FLIBI NIGHT

Peter Ferrie
Microsoft, USA

If we were to consider a computer virus to be a life form, then we could perhaps extend the analogy to include predators such as observant users and anti-malware solutions. We could also consider the need for mutation to produce new behaviours in order to evade predators and survive. The W32/Flibi virus aims to do just that.

EVOLUTION OF EVOLUTION

Previous efforts at evolving software have generally been in the form of fully programmed traits that are chosen using a weighting system, so that some traits are favoured more than others. An example of a piece of malware using this technique is W32/Simile (see *VB*, May 2002, p.4). This is a kind of evolution, but it cannot produce new behaviours. A variation on that theme was used by W32/Zellome (see *VB*, May 2005, p.7), whose traits corresponded to functions for the polymorphic decryptor, rather than the virus code itself. This cannot produce new behaviours either.

Flibi takes a new approach, and has some parallels with molecular biology. Each byte of the virus code (apart from the translator function) is equivalent to a codon[1]. We have a single 'reading frame', so each codon can be read in only one way, no matter where one starts reading. The translator function is equivalent to tRNA[2]. Its purpose is to convert the codons into amino acids[3]. Unlike in molecular biology, however, Flibi has no start and stop codons[4]. Instead, the translator function knows where the codon chain starts and how long it is (though both of these values are vulnerable to mutation which can affect future generations).

tRNA AT WORK

The virus begins by allocating 64KB of memory for the 'organism'. The preview version of the virus allocates the region using only read/write attributes. As a result, that version will not run if Data Execution Prevention (DEP) is enabled for all processes, but the bug was fixed in the release version of the virus. The translator function then converts the codons into amino acids and places them in the memory block. This is achieved by using the codons as an index into a table of instruction blocks. Each codon corresponds to one instruction block, and all but two of the blocks contain only a single instruction (the two exceptions contain two instructions each). There are many redundancies in the table (that is, many codons map to the same instruction). This is intentional, since it increases the robustness of the overall organism. Minor mutations can allow a modified codon to continue to be translated to the same amino acid (and this is yet another parallel to molecular biology). Each instruction block is eight bytes long, which leaves room for further minor mutations to occur without affecting the instruction itself.

IMPORT-ANT ASSIGNMENT

The virus builds the strings 'kernel32.dll' and 'advapi32.dll' using a sequence of add instructions. There are some other ways to achieve this, but the language contains only 45 instructions in the release version of the virus (43 instructions in the preview version), so there are not many other ways. (As a side note, the language can be simplified to as few as 18 instructions. The details will be provided in a future article.)

The virus loads kernel32.dll and then builds a series of pointers to memory, using another sequence of add instructions. The virus stores the image base value, and then locates the import table. It copies the import table pointers to a local buffer in memory. The virus parses the import table directly in order to resolve the APIs that it needs to replicate. Unlike other viruses that avoid using the GetProcAddress() API so that the import list is harder to determine, this virus avoids using the GetProcAddress() API in order to cause the list of APIs to become vulnerable to mutation. The virus still uses a list of hashes instead of function names, but if any of those hashes have been altered by mutation, then an entirely different API address might be retrieved, resulting in completely different behaviour. This is just one place where a new behaviour might arise from 'evolution'.

The virus parses the import table but in a rather peculiar way. Instead of examining the characters of the API name until the end of the name, the virus reads the first ten characters regardless of the length of the name. The characters are hashed using add, sub and xor, before and-ing the result to isolate the low 12 bits. There are three problems with this approach. One is that the API name

[1] A codon is a trinucleotide sequence of DNA or RNA (the nucleic acids that contain the genetic instructions used in the development and functioning of living organisms) that corresponds to a specific amino acid. See http://www.genome.gov/Glossary/index.cfm?id=36.

[2] Transfer RNA, or tRNA, is a small RNA molecule that is involved in protein synthesis. See http://www.wiley.com/college/boyer/0470003790/structure/tRNA/trna_intro.htm.

[3] Amino acids are the building blocks of proteins. See http://en.wikipedia.org/w/index.php?title=Amino_acid&oldid=412676887.

[4] See http://en.wikipedia.org/w/index.php?title=Genetic_code&oldid=412677908#Start.2Fstop_codons.

might be longer than ten characters, resulting in a possibly false match if those characters are shared by another API.

The second problem is that the API name might be shorter than ten characters, resulting in characters from the following name being read and incorporated into the hash. In that case, if other versions of *Windows* have different APIs after the one of interest, then the hash will not match on that platform. In fact, that is exactly what happened for the Sleep() API. The preview version of the virus carries a hash for the Sleep() API that is suitable for *Windows 2000* or *Windows XP*, where the Sleep() API is followed by the SleepEx() API. However, on *Windows Vista*, two additional APIs were inserted between the Sleep() and SleepEx() APIs: SleepConditionVariableCS() and SleepConditionVariableSRW(). The result is that the preview version of the virus crashes when run on *Windows Vista* or *Windows 7*. This bug was fixed in the release version by importing the Sleep() function explicitly.

The third problem is that the hashing algorithm is very weak and can easily result in false matches. It is unclear why the virus author did not use any kind of arithmetic rotation, given that the language supports both shift left and shift right operations. These operations can be combined to perform an arithmetic rotation in either direction.

After resolving the API addresses from kernel32.dll, the virus loads advapi32.dll and then branches to the code above while attempting to resolve some APIs from that DLL. After resolving the API addresses from advapi32.dll, the virus loads advapi32.dll again, because the API resolver code is not a subroutine, so the same code path is reached for the second time. However, the virus recognizes this case and it does not resolve the API addresses again.

WHIP IT INTO SHAPE

The virus constructs a new filename consisting of eight randomly chosen lower case letters, and then appends '.exe'. This is the name of the next-generation file. The virus retrieves the command line that was used to launch it. If the first character in the command line is a double quote, then the virus skips it and searches for the last double quote. Everything in between is extracted for use. If the command line does not begin with a double quote, then it is used as it is. The virus copies itself as 'x:\evorisss.exe', where 'x' is the drive letter taken from the command line. It also copies itself to the next-generation filename.

The virus opens the next-generation file and maps it into memory. For each byte in the file, there is a 0.02% chance that one of its bits will be flipped. There is also a 0.003% chance that any two adjacent dwords will be exchanged. Finally, there is a 20% chance per execution of the virus that between one and 32 bytes will be deleted by being overwritten with a corresponding number of the bytes immediately prior to the selected block, and then overwriting the prior bytes with no-operation instructions.

These mutations are often lethal, since the file headers are vulnerable to alteration, resulting in invalid executables. The random deletion of instructions will also eventually result in code that does the absolute minimum (such as copying itself only to the root directory of the current drive, and requiring user interaction in order for it to run again) in order to retain the 'worm' characteristic.

PAYLOAD

The virus carries a payload that displays with a 12.5% chance per execution of the virus. The payload is to display the message 'Kadyrov is a murderer!!!'. There are several 'infamous' people with the name Kadyrov; it is not known to whom this message refers. The string is not visible in the virus code, because it is constructed using yet another sequence of add instructions.

RUN WORM RUN

The virus waits for between 0 and 15ms, and then creates the registry key 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run'. The virus sets the default value to 'x:\evorisss.exe', where 'x' is the drive letter taken from the command line, as above. The virus also creates an autorun.inf file in the current directory, containing a reference to the next-generation file. The idea is that by browsing to the directory that contains autorun.inf, the worm will be launched without further user interaction.

The virus also enumerates all drives and checks the type of each of them. If the drive is removable, fixed, remote, or a ramdisk, then the virus attempts to copy autorun.inf to that drive, along with the next-generation file. There is a bug in this routine in the preview version of the virus, which is that if the drive is a floppy drive and there is no disk in it, then *Windows* will display an error message. This bug was fixed in the release version. The virus repeats the enumeration approximately once every seven minutes.

CONCLUSION

Evolution in software. It's an interesting idea, but this life form's biggest challenge is simply to survive. If an entire race of dinosaurs can be wiped out by a single meteor, it seems likely that good generic detection by anti-malware software could perform the same feat and kill this worm before it ever gets a chance to mutate beyond recognition.