MALWARE ANALYSIS 1

'LAHF'ING ALL THE WAY

Peter Ferrie
Microsoft, USA

We have seen a recent example of a virus that uses the obscure side effects of a certain instruction [1] to decode itself. Now we have a virus which decodes itself by using a much more subtle side effect of multiple instructions – the state of the CPU flags. We call this virus W32/Frilly.

ADMINISTRATA

The virus begins by setting the error mode to prevent serious errors from causing alerts to be displayed. This is not the same as exception handling. The virus has no exception handler, so in the highly unlikely event that the encoder causes a crash, *Windows* will simply terminate the program. The virus generates a new filename for itself, using eight randomly chosen lower-case letters, and then attempts to copy itself to 'c:\<filename>.exe'. Thus, every execution might result in a new file being created. This copy operation will fail on *Windows Vista* and later platforms, for non-administrator users. The virus attempts to open the copied file and map a view of it. If this is successful, then the virus will encode the contents using a very interesting encoding method.

ENCODER

With an almost 99% chance per iteration, the virus will generate a trash sequence. This will repeat until the 1% chance is hit. When the 1% chance is hit, the virus will generate a non-trash sequence. The entire logic is repeated until all bytes are encoded. The encoding method breaks a byte (eight bits) into two nybbles (four bits each). Each bit in a nybble is mapped to the location of a particular flag, according to the flag layout when the 'lahf' ('Load Status Flags into AH Register') instruction is used. The virus generates certain instruction sequences to set the CPU flags in a controlled way, to reproduce the bit values.

TRASH

The virus has six methods for generating instruction sequences, but in trash mode only two of them can be chosen. Thus, there is a 50% chance each that the first or second method will be chosen. The virus intended to select among the six methods with an approximately 16% chance for each, but due to a bug, only the first two can ever be chosen. This bug would have been very difficult to detect by looking only at the decryptor, because in non-trash mode five of the six methods can be selected (method four appears to have been overlooked). Unless someone was looking specifically for each of the instruction types, it would seem that all of the expected instructions are present somewhere in any given decryptor. It is also only by luck that the second method can be chosen in trash mode. If the virus author had implemented the method selection process here in the same way as in another place in the code, then the second method would not have been available, either.

The two methods generate a nybble decoder but using a random number instead of an encrypted byte value. There is no restriction on the requirements that need to be met (see below).

NON-TRASH

In non-trash mode, there is a 20% chance each for selecting among five styles of encoding. The styles are: using method six alone; using method one and then method five; using method six and then methods two and five (which is equivalent to just using methods two and five, because that combination replaces completely the effects of method six); using method three followed by method five; or using method five followed by method one. After applying an encoding style, there is a 50% chance each that the virus will use the 'pushfd' or 'lahf' instruction. If the 'pushfd' instruction is selected, then there is a 50% chance each that the virus will use the 'pop eax/stosb' sequence or the 'pop reg32/mov [edi],reg8l/inc edi' sequence, where reg is eax/ecx/edx/ebx. If the 'lahf' instruction is selected, then there is an approximately 33% chance each that the virus will use the 'mov [edi],ah/inc edi' sequence, the 'mov al,ah/stosb' sequence, or the 'xchg/stosb' sequence (and then another 50% chance each that the virus will use the 'ah,al' order or the 'al,ah' order in the 'xchg' instruction).

For each method, the virus will make up to 42 attempts to find values that satisfy the requirements, if any. If no values are found, then the virus will return to the top of the algorithm (the generation of trash instruction sequences until the 1% chance is hit) and resume from there.

METHOD ONE

The first method attempts to find a random number that causes the appropriate flags to be set when that number is rotated. The rotation is either to the left or to the right, using a count from 1 to 31. The virus can choose from three forms of rotation, with an approximately 33% chance of choosing any one of them. The left rotation can be in one of two forms – immediate by 1 or by cl, with a 50% chance of either one being chosen. The right rotation can only be by cl. It is unknown why the virus restricts the right rotation in

this way. It might be a bug, given that later code checks if the immediate form is in use.

The virus will choose a random register to hold the value. The register can be eax, edx or ebx. The ecx register is excluded because it might be used by the rotation. If the 'rotate by cl' form is in use, then the virus writes a 'mov ecx' instruction to assign the cl value. Then the virus writes the rotate instruction.

This method supplies the carry flag (that is, bit 0) for a given nybble, and it is used in conjunction with method five to supply the remaining flags in order to construct the complete nybble.

METHOD TWO

The second method attempts to find a random number that causes the appropriate flags to be set after performing an arithmetic adjustment on the number. The adjustment is in the form of 'aaa' ('ASCII Adjust After Addition') or 'aas' ('ASCII Adjust After Subtraction'), with a 50% chance of either one being chosen. The virus must use the eax register to hold the value, followed by the chosen instruction.

This method supplies the carry flag (that is, bit 0) for a given nybble. The virus wants to use this method also to supply the auxiliary carry flag (that is, bit 4), but the result is destroyed because this method is used in conjunction with method five to supply the remaining flags in order to construct the complete nybble.

METHOD THREE

The third method attempts to find a random number that causes the appropriate flags to be set when the number is shifted. The shift is either to the left or to the right, using a count from 1 to 31. The virus can choose from five forms of shift, with a 20% chance of choosing any one of them. The left shift can be in one of two forms – immediate by 1 or by cl, with a 50% chance of either one being chosen. The right shift can only be by cl. It is unknown why the virus restricts the right shift in this way. It might be a bug, given that later code checks if the immediate form is in use.

The virus will choose a random register to hold the value. The register can be eax, edx or ebx. The ecx register is excluded because it might be used by the shift. If the 'shift by cl' form is in use, then the virus writes a 'mov ecx' instruction to assign the cl value. Then the virus writes the shift instruction.

This method supplies the carry flag (that is, bit 0) for a given nybble. The virus wants to use this method also to supply the parity flag (that is, bit 2), but the result is destroyed because this method is used in conjunction with method five to supply the remaining flags in order to construct the complete nybble.

METHOD FOUR

As noted above, the fourth method is not usable in any form, but it will be described anyway. This method attempts to find two random numbers that cause the appropriate flags to be set when one of the following logical operations is performed on them: and, xor, test. The first two operations have a 25% chance each of being selected. There is a 50% chance that the 'test' operation will be selected.

The virus will choose a random register to hold the first value. The register can be eax, ecx, edx or ebx. There is a 50% chance that the virus will choose a random register to hold the second value, otherwise an immediate value will be used instead. The second register can also be eax, ecx, edx or ebx, but not the same as the register which holds the first value. Then the virus writes the logical instruction.

The virus could potentially have used this method to supply the parity and sign flag (that is, bits 2 and 7). However, since the state of the auxiliary carry flag is officially undefined (of course in reality, it is well defined and understood, it is just not documented), there is no method that can supply that flag while preserving the others.

METHOD FIVE

The fifth method attempts to find a random number that causes the appropriate flags to be set when the number is either incremented or decremented. There is a 50% chance each for selecting either direction of the adjustment. The virus will choose a random register to hold the value. The register can be eax, ecx, edx or ebx. Then the virus writes the adjustment instruction.

This method supplies the parity, auxiliary carry and sign flags (that is, bits 2, 4 and 7) for a given nybble.

METHOD SIX

The sixth method attempts to find one (or two, as appropriate) random number(s) that cause(s) the appropriate flags to be set when one of the following operations is performed on it (or them): neg, add, sub, cmp. Each operation has a 25% chance of being selected.

The virus will choose a random register to hold the first value. The register can be eax, ecx, edx or ebx. For the 'neg' instruction, the virus writes the instruction and continues execution. For the other instructions, there is a 50% chance that the virus will choose a random register to hold the second value, otherwise an immediate value will be used

instead. The second register can also be eax, ecx, edx or ebx, but not the same as the register which holds the first value. Then the virus writes the instruction.

This method supplies the carry, parity, auxiliary carry and sign flags (that is, bits 0, 2, 4 and 7) for a given nybble. This forms a complete nybble, which is why the method can be used on its own.

DECODER

For the decoder, as noted above, the virus generates certain instruction sequences to set the CPU flags in a controlled way, to reproduce the bit values. The values of carry, parity, auxiliary carry and sign flags (that is, bits 0, 2, 4 and 7) are grouped in a particular order to form one nybble at a time of each byte of the original code. Two nybbles are combined to recover one byte of the original code. This cycle is repeated until all of the original bytes are recovered.

After decoding itself, the virus attempts to create the 'HKLM\Software\Microsoft\Windows\CurrentVersion\Run' key and set the default value to the name of the generated file. This action fails on *Windows Vista* and later for non-administrator users. If it succeeds, then a new file will be created on each reboot, potentially quickly filling the root directory of the boot drive.

The virus creates a hidden file named 'autorun.inf' in the current directory. This contains a reference to the generated filename. The virus enumerates the drive letters from A: to Z:, and queries the drive type. For removable, remote, and ram disks, the virus copies the autorun file to the root of the drive. For those drive types and also fixed drives, the virus copies the generated file to the root of the drive. After all drives have been examined, the virus waits for a random period of up to a maximum of about 32 seconds, and then performs the drive enumeration again. This cycle repeats endlessly.

CONCLUSION

Viruses that integrate the encoded virus body are a nuisance for static analysis, because there is no easy way to decrypt the non-existent single block of data. Fortunately, examples like this are trivial to emulate, and no effort is required to dump the decoded data automatically. At that point, no amount of obfuscation makes any difference.

REFERENCES

[1] Ferrie, P. So, enter stage right. Virus Bulletin, June 2012, p4. http://www.virusbtn.com/pdf/magazine/2012/201206.pdf.