MALWARE ANALYSIS

SO, ENTER STAGE RIGHT

Peter Ferrie
Microsoft, USA

Some virus writers try to find obscure side effects of instructions in an attempt to confuse virus analysts. Sometimes they succeed (indeed, sometimes they do so accidently). Sometimes we already know about the side effects, but we just don't talk about them. The latter is the case with the technique used in the W32/Flizzy virus.

MAKING A HASH OF THINGS

The first generation of the virus begins by fetching the value in the ImageBaseAddress field of the Process Environment Block, and applying it to the original entry point value. This allows the virus to work correctly in processes that have Address Space Layout Randomization enabled. The virus continues by setting up a Structured Exception Handler (SEH) in order to intercept any errors that occur during infection. The virus retrieves the base address of kernel32.dll by walking the InMemoryOrderModuleList from the PEB_LDR_DATA structure in the Process Environment Block. The address of kernel32.dll is always the second entry on the list. The virus assumes that the entry is valid and that a PE header is present – a safe assumption because the SEH that the virus has registered will intercept any invalid memory access.

The virus resolves the addresses of the bare minimum set of API functions that it needs for replication: find first/next, open, map, unmap, close. The virus uses hashes instead of names, but they are sorted alphabetically according to the strings they represent. The virus uses a reverse polynomial to calculate the hash – the return of the magic '0xEDB88320' value, that no-one seems to understand. Since the hashes are sorted alphabetically, the export table only needs to be parsed once for all of the APIs. Each API address is placed on the stack for easy access, but because stacks move downwards in memory, the addresses end up in reverse order. The virus also checks that the exports exist by limiting the parsing to the number of exports in the table. The hash table is terminated with a single byte whose value is 0x2a (the '*' character). This is a convenience that allows the file mask to follow immediately in the form of '*.exe', however it also prevents the use of any API whose hash ends with that value. As with previous viruses by the same author, Flizzy only uses ANSI APIs. The result is that some files cannot be opened because of the characters in their names, and thus cannot be infected.

GETTING A HANDLE ON IT

The virus searches in the current directory (only), for objects whose names end in '.exe'. There is a bug in the code in that it does not close the handle that is used to search the directory. As a result, a handle is leaked for as long as the process runs. The search is intended to be restricted to files, but can also include directories, and there is no filtering to distinguish between the two. For each such file that is found, the virus attempts to open it and map an enlarged view of the contents. There is no attempt to remove the read-only attribute, so files that have this attribute set cannot be infected. In the case of a directory, the open will fail, and the map will be empty. The map size is equal to the file size plus a little more than 4KB, to allow the file to be infected immediately if it is acceptable. The value of the size increase is hard-coded in the virus, which is strange, given that the size of the encoded form of the virus is only slightly more than half of that value. Using the post-infection size during the validation stage allows the virus to avoid having to close the file and re-open it with a larger map later. The virus assumes that the handle can be used, and then checks whether the file can be infected.

The virus is interested in Portable Executable files for the *Intel* x86 platform with no appended data. Renamed DLL files are not excluded, nor are files that are digitally signed (at least, not explicitly – most of them will be filtered implicitly, because it is common for the signature to be placed after the end of the last section, but this is not a requirement). The subsystem value is restricted to console mode applications, despite a comment in the source code which suggests that GUI applications were the intended target. If the file passes all of these checks, then the virus increases the file size by 4KB+1 bytes. The extra byte serves as the infection marker, because it will appear to be appended data, and the virus will not attempt to infect the file.

The virus increases the virtual and physical sizes of the last section, and the SizeOfImage, by 4KB. The section attributes are marked as executable and writable. The virus constructs a new decoder, and then zeroes the region that will hold the encoded bytes, even though *Windows* zeroed the region automatically when the file was mapped.

The virus zeroes the RVA of the Load Configuration Table in the data directory. This has the effect of disabling SafeSEH, but it affects the per-process GlobalFlags settings, among other things. The virus saves the original entry point in its body, and then sets the host entry point to point directly to the virus code. The virus code ends with an instruction to force an exception to occur, which is used as a common exit condition. However, it does not recalculate the file checksum, and does not restore the file's date and timestamps either, making it very easy to see which files have been infected.

ENTER HERE

When an infected file is executed, the virus decodes itself using an obscure stack operation. The 'enter' instruction is

used most often to allocate space on the stack for variables. However, it can also be asked to copy previous stack frames into the new one. Specifically, the ebp register value will be adjusted according to the nesting level. The resulting pointer will be used to read from a memory location, and the value at that location will be pushed onto the stack. This is both an indirect memory push, and one with no obvious reference to the location. In a flat memory space, such as on the *Windows* platform, the SS and DS registers have the same value. As a result, the 'enter' instruction can be used to copy data to the stack from anywhere in memory (and can even be used to perform a memcpy() of up to 31 dwords). The virus uses this feature to order the bytes of its body randomly, and creates a table of pointers that correspond to their original position. The ebp register indexes each of the table entries in order to restore the body to its original form.

The decoder has some other unusual characteristics, which increase the size for no good reason. For example, the stack register value is saved in another register, in order to restore it later. The reason the virus must save the stack register is because of a misuse of the 'enter' instruction. The virus requests that the previous stack frame be copied into the new stack frame, but it also requests that a dword be reserved on the stack. This dword is not used, and the reservation could have been avoided. The virus corrects the stack pointer to discard the reservation and the previous stack pointer, but does so by using an 'add' instruction that is larger than the two equivalent 'pop' instructions (and if no variable space were reserved, then only one 'pop' instruction would have been needed). The indirect memory value that was pushed by the 'enter' instruction is then popped, leaving the previous stack frame pointer on the stack. This value could have been popped, too, and the combination (assuming that the variable reservation did not exist) would still be shorter than the 'add' instruction. Also, by using the 'pop' method, the stack pointer would be balanced after the loop finishes, and there would be no need to save the original stack pointer value at all.

The virus caches the ImageBaseAddress field value in the decoder, even though its value is not altered and it is not used until after the decoding has completed. The way in which the ImageBaseAddress value is used is also strange, given that the virus writer focuses on size optimizations. The virus fetches the ImageBaseAddress value and then adds the original entry point value to it. Instead, the virus could have moved the original entry point into a register, and then added the ImageBaseAddress value to it. This would have required fewer bytes, and avoided the use of another register.

CONCLUSION

This use of the 'enter' instruction is an interesting idea, but the effect is documented (complete with pseudo-code), so there shouldn't be any surprises for emulators.