MALWARE ANALYSIS 1

'AMFIBEE'-OUS VEHICLE

Peter Ferrie
Microsoft, USA

A cross-infector is typically implemented as two viruses stuck together, simply because it's the easiest (and in most cases the *only*) way to do it. However, the x64 technology is a special case – 64-bit instructions use a prefix which corresponds to a register decrement in 32-bit mode. With careful coding, that effect can be reversed as appropriate, and then code can be shared between the 32-bit and 64-bit platforms. There are already at least two known 32-bit and 64-bit cross-infectors for *Windows*, but now we have the first 32-bit and 64-bit cross-infector for *Windows* that is almost entirely a single block of code: W32/W64.Amfibee.

STACKING THE RANKS

The first generation of the virus begins by saving the relative virtual address of the original entry point on the stack. However, depending on the image base value that was used when building it, this value might be completely wrong. In order to account for Address Space Layout Randomization (ASLR), the virus applies the current image base value from the ImageBaseAddress field in the Process Environment Block. This is an interesting way to deal with ASLR – it is more common simply to calculate the difference between a branch instruction and the host entry point. The virus also saves the current stack pointer to a field in its body. Using this value the virus can undo any changes to the stack at any point during the execution of the code. This is particularly important during API resolution, since the virus cannot easily determine how many APIs have been saved before something goes wrong.

DETERMINISM

The virus begins by retrieving the base address of ntdll.dll. It does this by walking the InMemoryOrderModuleList from the PEB_LDR_DATA structure in the Process Environment Block. This is compatible with the changes that were made in *Windows 7*. The required offsets within the InMemoryOrderModuleList are platform-dependent. The virus determines the platform on which it is executing by using a RIP-relative instruction. On the 32-bit platform, the instruction returns a zero in the register. On the 64-bit platform, the same instruction returns an address in the register. The virus also saves the pointer to the current position in the InMemoryOrderModuleList so that it can resume the parsing later to find the base address of kernel32.dll. If the virus finds the PE header for ntdll.dll, it resolves the two required APIs: RtlAddVectoredExceptionHandler and RtlRemoveVectoredExceptionHandler.

The platform determination code is the first 'mistake' in the code. The wrong size of register is checked, resulting in a redundant prefix after every check, which leads to an unnecessary increase in the size of the code. There are a number of similar 'mistakes' in the code.

AN EXPORT EXHORT

The virus uses hashes instead of names, but unlike previous viruses by the same author [1–3], the hashes are not sorted alphabetically according to the strings they represent. As a result, the export table must be parsed repeatedly in order to resolve the APIs. This is not really a problem, it is just not an efficient way to do things. There is also no obvious reason for doing it, since the two registers that are used during the resolution are not altered by anything else. Thus, if the hashes were sorted, the API resolution could easily continue from the current position, with no increase in the size of the code.

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 in memory. The virus also checks that the exports really 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'. The virus retrieves the base address of kernel32.dll by fetching the next entry in the InMemoryOrderModuleList list, using the pointer that was saved earlier. The same routine is used to retrieve the addresses of the API functions that it requires, which is the minimum set of APIs that it needs for replication (but because of a bug, more than it actually uses): findfirst/next, open, map, unmap (see below), close.

As with previous viruses by the same author, this virus 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. The virus searches in the current directory (only), for objects whose names end in '.exe'. This is intended to be restricted to files, but can also include any directories that have such a name, and there is no filtering to distinguish between the two cases. For each such file that is found, the virus attempts to open it and map a view of the contents. There is no attempt to remove the read-only attribute, so files that have that attribute set cannot be infected. In the case of a directory, the file open will fail, and the map will be empty. The virus registers an exception handler at this point, and then checks whether the file can be infected.

RELOCATION ALLOWANCE

The virus is interested in Portable Executable files for either the x86 or x64 platforms. Renamed DLL files are not excluded, nor are files that are digitally signed. The subsystem value is checked, but incorrectly. The check is supposed to limit the types to GUI or CUI but only the low byte is checked. Thus, if a file uses a (currently non-existent) subsystem with a value in the high byte, then it could potentially be infected too.

The virus checks the Base Relocation Table data directory to see if the relocation table begins at the start of the last section. If it does, then the virus assumes that the entire section is devoted to relocation information. This could be considered to be a bug. The virus checks that the physical size of the section is large enough to hold the virus code. There are two bugs in this check.

The first is that the size of the relocation table could be much smaller than the size of the section, and other data might follow it. The data will be overwritten when the virus infects the file. Further, the value in the Size field of the Base Relocation Table data directory cannot be less than the size of the relocation information, and it cannot be larger than the size of the section. This is because the value in the Size field is used as the input to a loop that applies the relocation information. It must be at least as large as the sum of the sizes of the relocation data structures. However, if the value were larger than the size of the relocation information, then the loop would access data after the relocation table, and that data would be interpreted as relocation data. If the relocation type were not a valid value, then the file would not load. If the value in the Size field were less than the size of the relocation information, then it would eventually become negative and the loop would parse data until it hit the end of the image and caused an exception.

The second bug is that by checking only the physical size and not the virtual size, whatever the virus places in the file might be truncated in memory if the virtual size of the section is smaller than the physical size of the section. Both of these bugs are also present in some of the other viruses created by the same author.

If the section appears to be large enough, then its attributes are marked as executable and writable, and the virus copies itself to the relocation table. After copying itself, the virus zeroes the value in the Offset field of the Base Relocation Table data directory, saves the original entry point in the virus body, and then sets the host entry point to point directly to the virus code.

OFF THE MAP

The virus code ends with an instruction to force an exception to occur. This is used as a common exit condition. However, the virus does not recalculate the file checksum, even though it might have changed as a result of infection. It also does not restore the file's date and timestamps, making it very easy to see which files have been infected, even though the file size does not change.

There is a bug here, the severity of which depends on the configuration of the environment.

The bug is that the virus calls the wrong API when attempting to unmap the view of the file (and therefore the 'unmap' API is never used). The correct index is used to access the API table, but the virus uses the wrong calling convention, so it ends up calling the CloseHandle() API instead. The act of calling the CloseHandle() API with an invalid handle has a particular effect if a debugger is present, and the same effect if a debugger is not present but if a certain registry value contains a certain value. Normally (that is, no debugger and no registry value), the result is simply that an error code is set, and there is no visible effect. This probably explains why the bug was not noticed.

If a debugger is present, then *Windows* will raise an exception. If a debugger is not present, but the FLG_ENABLE_CLOSE_EXCEPTIONS (0x400000) flag is set in the 'HKLM\System\CurrentControlSet\Control\Session Manager\GlobalFlag' registry value prior to the system being rebooted, then an exception will also be raised. Further, if the flag were set while the virus was running, then the virus would be terminated by *Windows* because the virus unregisters the exception handler prior to closing the file. In that case, the bug would have been very noticeable.

However, the bug will be particularly noticeable in a directory that contains many executable files, regardless of their suitability for infection. The problem is that since the map view is never unmapped, it remains in memory – one map per file that is examined. The size of the map is equal to the size of the file, and each of the maps is aligned to a 64KB block. This is in addition to the host image which is also present in memory, along with its associated DLLs. Thus, it might require only a few hundred large files to be mapped before the memory becomes so scarce that the host simply cannot run after the virus completes its work.

SIZE DOES MATTER

The code is mostly optimized for size, but some obvious size optimizations are missing, such as during the transfer of control that appears after the API resolution. The existing

code determines the platform and then jumps through the stack using a platform-specific value. However, the check could have been avoided completely by using a particular single-byte instruction earlier in the code. The result would be a single jump instruction using a value that is common to both platforms.

In other cases, such as in the vectored exception handler, at least part of the two code paths could be merged by using a nice trick whereby the platform check is used to skip just the REX prefix. There are other examples of multiple code paths where a single one could have been used, such as finding the image base of kernel32, or indexing the API table on the stack.

There are also cases where the REX prefix is redundant because of an unexpected register behaviour on the 64-bit platform. Specifically, assigning a value to a 32-bit register results in the upper 32 bits of the corresponding 64-bit register being zeroed. This also results in a bug which, fortunately for the virus writer, does not have an effect because of a compatibility decision by *Microsoft*. The bug is that the virus retrieves the 32-bit RVA of the export table from ntdll.dll or kernel32.dll, but forgets to add the full 64-bit image base prior to using it. As a result, only the low 32 bits are valid, but it just so happens that the image base of both of those DLLs is always in the low 2GB range, and thus the size of the image base never exceeds 32 bits. In the case of kernelbase.dll and a number of other introduced DLLs, on the other hand, the size of the image base does exceed 32 bits. If the virus had attempted to access any APIs from such a DLL, then the bug would have caused a crash. However, since only ntdll.dll and kernel32.dll are used, the REX prefix is not needed to access their memory.

CONCLUSION

A virus that can run its code natively on both 32-bit and 64-bit platforms is a bit like a lungfish that can live in water or on land (but perhaps less ugly). Fortunately, this virus is in the early stages of evolution – however, we can probably expect to see future advances in this technique.

REFERENCES

[1] http://www.virusbtn.com/virusbulletin/archive/2011/09/vb201109-Holey.

[2] http://www.virusbtn.com/virusbulletin/archive/2012/01/vb201201-sig.

[3] http://www.virusbtn.com/virusbulletin/archive/2012/02/vb201202-Svar.