

MALWARE ANALYSIS 2

NOT DROWNING, WAV-ING

Peter Ferrie

Microsoft, USA

There is a big problem with Pseudo Random Number Generation (PRNG) algorithms: they're not random. They require a seed as a starting point, and then generate values in a cyclic manner (of course, the cycle can be very large). Hence, given the seed and the number of iterations, the next value and all subsequent values can be determined. The obvious solution is to make the random number generator really random, by turning the generator into a provider instead, and finding a suitable source of random numbers to collect. The W32/Mammer virus attempts to do just that.

IMPORT BUSINESS

The virus begins by registering a Structured Exception Handler in order to intercept any errors that occur during infection. The virus retrieves the base address of kernel32.dll. It does this 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 in the list. The virus assumes that the entry is valid and that a PE header is present there. This assumption is fine, because of the Structured Exception Handler that the dropper has registered.

The virus resolves the addresses of the API functions that it requires: find, set attributes, open, map, unmap, close, malloc, free and LoadLibrary. The virus uses hashes instead of names and uses a reverse polynomial to calculate the hash. Since the hashes are sorted alphabetically according to the strings that they represent, the export table needs to be parsed only 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 in memory. The hash table is terminated with a single byte whose value is zero. While this saves three bytes of data, it also prevents the use of any API whose hash ends with that value. This is not a problem for the virus in its current form, since none of the needed APIs have such a value, but it could result in some surprises for any virus writer who subsequently tries to extend the code.

The virus loads 'winmm.dll' and resolves the addresses of the API functions that it requires, using the hash method again. It then allocates a 689KB buffer.

TIDAL WAVE

The virus registers a second Structured Exception Handler, and then opens the audio input device for recording. The

recording format is a *Microsoft*-proprietary audio format, using two channels of 44,100Hz, and 16-bit samples at 172KB/s. The input is not filtered in any way. The virus registers a third Structured Exception Handler, prepares the audio buffer for receiving data, and initiates the recording. Next, the virus repeatedly calls an API to finalize the buffer, and loops while the function returns a status indicating that the buffer is not yet full.

The reason the virus has to call the API repeatedly is because it did not register a callback function when it prepared the audio buffer. If the virus had registered a callback function, then that function would receive the notification that the buffer was full, and the virus could have used a delay loop to wait for that event to occur.

The virus checks only the low byte of the status value, which might appear to be a bug, but this is actually safe because of the way in which the error codes are constructed for the API. The error codes are built on ‘bases’, which are specific to the type of function in use. The base that corresponds to the wave format fits entirely within an eight-bit value, so only those eight bits need to be checked.

Once the virus receives the status that the buffer is full, it uses a breakpoint instruction to force an exception to occur and to transfer control to the third Structured Exception Handler. This technique was first seen in the Chiton [1] family, and it appears a number of times in the virus code. It is an elegant way to reduce the code size, in addition to functioning as an effective anti-debugging method. Since the virus has protected itself against errors by installing a Structured Exception Handler, the simulation of an error condition results in the execution of a common block of code to exit a routine. This avoids the need for separate handlers for successful and unsuccessful code completion.

When the third Structured Exception Handler receives control, it closes the audio device, and then checks if the buffer received at least 16 bytes before the exception occurred. If the buffer is not full enough, then the virus uses another breakpoint instruction to force an exception to occur and to transfer control to the second Structured Exception Handler. When the second Structured Exception Handler receives control, it frees the allocated buffer, and then uses yet another breakpoint instruction to force an exception to occur and to transfer control to the first Structured Exception Handler.

MISTYPE, MISS TYPE

If the buffer is full enough, then the virus intends to copy the first 16 bytes from the buffer to use as decryption keys. The one major bug in the virus code is right here. A simple typographical error – the letter ‘a’ instead of the letter ‘c’

– results in the wrong buffer being used as the source for the keys. So, instead of copying the audio buffer *data*, the audio buffer *header* is copied. The result is that three of the four keys are entirely constant, and half of the remaining key is constant, too. This makes decryption trivial, even in the absence of an emulator. The fact that the bug was not discovered suggests that the operating system that was used for testing is one that implements Address Space Layout Randomization, since otherwise even the fourth key would very likely be constant. Of course, this observation is simply a minor point of interest and serves no other purpose.

BITS AND PIECES

In any case, the virus begins the replication phase as though everything were fine. It registers a fourth Structured Exception Handler, and then searches in the current directory (only) for PE files, regardless of their extension. The virus uses *Unicode*-only APIs, which allows it to examine files that would otherwise be inaccessible to ANSI APIs. It uses a nice trick to find the files, which was first seen in the Chiton [1] family: the file mask is ‘*’ which, when pushed onto the stack, can be interpreted as a zero-terminated *Unicode* string because it is followed by three zeroes. The rest of the code is derived from the Mikasa [2] virus.

The virus attempts to remove the read-only attribute from whatever is found. It attempts to open the found object and map a view of it. If the object is a directory, then this action will fail and the map pointer will be null. Any attempt to inspect such an object will cause an exception to occur, which the virus will intercept. If the map can be created, then the virus will inspect the file for its ability to be infected.

SEEK AND DESTROY

The virus is interested in Portable Executable files for the *Intel* x86 platform that are not DLLs or system files. The check for system files could serve as a light inoculation method, since *Windows* ignores this flag. The virus checks the COFF magic number, which is unusual, but correct. The reason for checking the value of the COFF magic number is to be sure that the file is a 32-bit image. This is the safest way to determine that fact because, apart from the executable (‘IMAGE_FILE_EXECUTABLE_IMAGE’) and DLL (‘IMAGE_FILE_DLL’) flags in the Characteristics field, all of the other flags are essentially ignored by *Windows* (from the point of view of the virus, that is true, but technically it’s not quite accurate – setting the ‘IMAGE_FILE_RELOCS_STRIPPED’ flag has the effect of disabling Address Space Layout Randomization for the process). This

includes the flag ('IMAGE_FILE_32BIT_MACHINE') that specifies that the file is for 32-bit systems.

As an added precaution, the virus checks that the size of the optional header is large enough to hold the BaseRelocationTable directory. If the optional header is also large enough to hold the LoadConfigurationTable data directory, then the virus requires that the LoadConfigurationTable RVA is zero. The reason for this last check is because the table includes the SafeSEH structures, which will prevent the virus from using arbitrary exceptions to transfer control to other locations within its body. The virus checks that the file targets the GUI subsystem.

RELOCATION ALLOWANCE

The virus checks the Base Relocation Table data directory to see if the relocation table begins at the exact 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 too strict. 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 bug 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 might 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 as well, 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.

TOUCH AND GO

If the section appears to be large enough, then the virus overwrites the relocation table with the decryptor and

the encrypted virus body. Overwriting the relocation table means that infected files do not show an increase in file size. The encryption method is to use 32 rounds of XTEA, using the 'keys' from above. The virus changes the section characteristics to writable and executable, and sets the host entry point to point directly to the virus code. The virus clears only two flags in the DLL Characteristics field: IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY and IMAGE_DLLCHARACTERISTICS_NO_SEH. This allows signed files to be altered without triggering an error, and enables Structured Exception Handling. The virus also zeroes the Base Relocation Table data directory entry, to prevent the virus code from being interpreted as relocation data, in the event that the file opted in to Address Space Layout Randomization.

The host's original entry point RVA is saved in the decryptor code. When the decrypted code is run, the virus converts the RVA to a virtual address by adding the ImageBase value from the Process Environment Block to it. This allows the virus to behave correctly if the file is relocated in memory.

Once the infection process has completed, the virus uses a breakpoint instruction to force an exception to occur and to transfer control to the fourth Structured Exception Handler. When the fourth Structured Exception Handler receives control, it unmaps and closes the file, and restores its file attributes, but not the file date and times. After all files have been examined, the virus uses a breakpoint instruction to force an exception to occur and to transfer control to the first Structured Exception Handler. When the first Structured Exception Handler receives control, it transfers control to the host entry point.

CONCLUSION

True random number generation certainly has its uses, and the recording of ambient sound as a source of random numbers is a valid technique, even though the implementation is flawed in this example. In any case, this virus gains no advantage by using a truly random number generator, because it must still carry the decryption keys.

REFERENCES

- [1] Ferrie, P. Unexpected Results [sic]. Virus Bulletin, June 2006, p.4. <http://www.virusbtn.com/pdf/magazine/2002/200206.pdf>.
- [2] Ferrie, P. It's mental static! Virus Bulletin, March 2013, p.8. <http://www.virusbtn.com/pdf/magazine/2013/201303.pdf>.