# MALWARE ANALYSIS

## DEELAED LEARNING

*Peter Ferrie*
Microsoft, USA

Not long ago, a new virus writer appeared, using the name 'hh86'. Rumour had it that hh86 was female – a rarity in the virus-writing world. There was a flurry of activity from hh86 over a period of about three months, producing a handful of viruses using new techniques, and then... she was gone without a word. The model virus writer perhaps.

At first glance, I thought that her first virus (Deelae.A) was simply a copy of a virus created by the virus writer roy g biv. A slightly closer look revealed some novel size optimizations (as well as some opportunities that were missed, and some 'optimizations' that are the same size but slower to execute) as well as some differences in style. It's clear that hh86 was 'inspired' by roy g biv's work. In Hollywood, they'd call that 'reimagining'.

### TECHNOLOGY ANTHOLOGY

Let's start with the things that are the same. The most obvious is the construction of a Structured Exception Handler (SEH) which is used as a single point of exit when an error occurs. Even the exception condition is the same – an interrupt 3 instruction. One annoying technique the pieces of malware have in common is the loading of the stack with as many parameters as possible, prior to calling APIs in sequence (see *VB*, October 2004, p.4, for an illustrated example).

Another technique they share is the method used to retrieve a DLLBase value from the PEB_LDR_DATA structure in the Process Environment Block. This technique was actually first published by the virus writer Ratter, and he seems to be the more likely source here. The reason for this speculation is that roy g biv has used the technique only once – it wasn't in a virus at all (it was in the BASLR tool), and it loaded a different DLL from the one that hh86 is trying to load. That difference introduces a problem for hh86.

The PEB_LDR_DATA structure contains three structures ('InLoadOrderModuleList', 'InMemoryOrderModuleList', and 'InInitializationOrderModuleList', but *Microsoft* documentation lists the first structure as 'reserved', names the second one, and implies that the third does not exist). Any of the three structures can be parsed to find the list of DLLs that are loaded. The difference is in the order of the pointers that are referenced by the structures, and thus the amount of code used to complete the retrieval.

All three virus writers used the InInitializationOrderModuleList structure (although roy g biv called it the 'InLoadOrderModuleList'). This results in the smallest code, but it does not work on *Windows 7* when it is used to retrieve the DLLBase of kernel32.dll. The BASLR tool (created by roy g biv) loaded ntdll.dll, which does work on *Windows 7*, but hh86 (and Ratter) loaded kernel32.dll. Thus, Deelae.A and Deelae.B do not work on *Windows 7*. This might be considered the first bug, and hh86 seems to have thought so too, since it was fixed in Deelae.C.

### IMPORT-ANT DETAILS

The code used to resolve the imports is also similar, up to a point. Deelae uses the CRC32 method, to avoid the need to store the strings. However, the CRCs are not sorted according to the alphabetical order of the strings they represent, so multiple passes over the export table are required to resolve the imports. The code that resets the index contains one potential problem. The code that checks for the end of the list contains several potential problems.

We'll start with the problems in the list termination code. The first problem is that only one byte is checked (this is especially curious, given that four bytes were allocated in Deelae.A and Deelae.B). This prevents the use of any function whose CRC32 value would have the checked value in the lowest byte.

The second problem is that the checked value is not constant in Deelae.A and Deelae.B, thanks to the way in which it is constructed. Instead, it depends on the previous CRC32 value. If the top byte of the previous CRC32 value is less than 0x80, then the checked value will be zero. Otherwise, the checked value will be 0xff. As a result, the list cannot have the form 0x00-7f x1 x2 x3 0x00 y1 y2 y3, or 0x80-ff x1 x2 x3 0xff y1 y2 y3, because in each case, the list will appear to terminate too soon.

The third problem is that if the terminating value is 0xff, it leads to a real bug in Deelae.A and Deelae.B. A few instructions later, some space is allocated on the stack using a modification to the register that held the checked value. If the terminating value is 0xff, then the stack pointer is moved in the wrong direction, destroying the SEH registration, and leading quickly to a crash.

This third problem was fixed in Deelae.C in two ways. The first was by changing the comparison register to one that always holds a zero – but the byte-check remains, and therefore so does the first problem. This is despite the fact that the comparison register is entirely zero, and thus all four bytes could have been checked. However, in Deelae.C, the CRC table was changed to end with a single byte, and this change is also present in Deelae.D and Deelae.E. The second way was to allocate the stack space using an instruction specifically designed for the purpose, so no arithmetic overflow can occur.

The problem with the index reset code is that the index might be set to 0xffffffff instead of zero, because of the second problem above. This index is used to look inside the Name Pointer table. In the (obviously impossible) case that the Name Pointer table had a Relative Virtual Address (RVA) of zero (that is, pointing to the 'MZ' part of the file header), the attempt to retrieve the first name RVA would access memory outside of the image, and cause an exception. However, the exception would be intercepted and the virus code would simply exit without issue. There is also the remote possibility that a wanted string appears immediately before the import table, and that would lead to the wrong function pointer being retrieved. That could result in unexpected behaviour, but a crash seems likely. The exception should be intercepted here too, but depending on what is called, a real crash might occur. The file to infect might also be corrupted.

## DIFFERENCE OF OPINION

One of the major ways in which hh86's viruses differ from those of roy g biv is in the file handling. roy g biv's viruses would not infect files that are protected by the System File Checker (this feature was added in Deelae.F), they would remove the read-only attribute if required, and they would not infect files that were for a different CPU, or were DLLs if the virus did not support them.

In constrast, hh86's viruses do not care about the read-only attribute, they do not check the CPU, and they will infect DLLs (in the unlikely event that they are misnamed). They do, however, clear the 'NX' bit if it is set. This allows the viruses to execute on systems with Data Execution Prevention enabled, even if the section is not marked as 'executable'. roy g biv's viruses, on the other hand, set the 'executable' bit in the section header. Both techniques achieve the same goal. roy g biv's viruses search recursively through all directories, and infect files regardless of their filename; hh86's viruses search only within the current directory, and only for '*.exe'. hh86's viruses are also aware of SafeSEH (see below), which did not exist when the majority of roy g biv's viruses were created, but even his most recent viruses do not support it. Interestingly, neither Deelae.A nor Deelae.B is aware of the NO_SEH bit (see below), but Deelae.C, Deelae.D and Deelae.E are.

Now it's time to describe the specifics of each of hh86's viruses.

## CERTIFICATE HOLDER

Deelae.A checks if the Certificate Table data directory indicates the presence of a digital certificate of at least 4KB in size after the end of the last section. If one is found, the virus zeroes the Certificate Table data directory (because it will be overwritten) and the Load Config Table to disable SafeSEH. Disabling SafeSEH allows the virus to raise exceptions without causing the application to be terminated by the operating system *if the NO_SEH bit is also clear*. The reason for this is because the NO_SEH bit states that no code within the image uses SEH, and thus no code is allowed to call SEH within the image. This bit was introduced in *Windows XP* SP2, but was not documented until recently.

The virus increases the virtual and physical sizes of the last section, and the SizeOfImage value, by 4KB. It copies itself over the certificate data, and then saves and alters the host entrypoint to point to the start of the certificate data. Finally, the virus copies itself over the certificate data and then forces an exception to occur, thus ending the infection process for that file. In the past, few files carried digital certificates, which would have limited the number of available candidates for this virus. However, an increasing number of files are now released with digital certificates, so overwriting the certificate has become a viable technique to avoid an increase in file size.

## EXPECT DEELAES

Deelae.B is identical to Deelae.A with the exception of the entrypoint hook. This time, the virus is interested in the Delay Import Descriptor table.

The Delay Import Descriptor table is used, as the name implies, to delay the importing of DLLs until they are needed. That can improve the start-up time for some applications, and also reduce the memory requirements. If a particular code-path is the only one that requires a certain DLL, and if that code-path is not taken, then the DLL will not be loaded unnecessarily. The table has been documented for a long time, but incorrectly, despite several revisions to the documentation. Specifically, the 'Attributes' field is documented as 'Must be zero', and 'As yet, no attribute flags are defined. The linker sets this field to zero in the image.' However, this is not true. The linker sets this field to 1, indicating that the table is valid. If bit zero is not set in the table, then an exception is raised by the application.

The virus retrieves the RVA of the Delay Import Descriptor, and checks that it points within the first section. The virus then retrieves the RVA of the Delay Import Address Table and checks that it points within the second section. If both checks pass, then the virus hooks the first address in the Delay Import Address Table to point to the start of the certificate data.

This entrypoint-obfuscating technique is new. roy g biv showed that the Bound Import Table can be hooked, though he went to extremes to hide the code (see *VB*, December 2006, p4). hh86 showed that the Delay Import Table can be hooked, and in multiple ways (see below), but she did not attempt to hide it (however, the popular tool known as the *Interactive DisAssembler* does it for her – it hides the entrypoint in an attempt to avoid bad links in corrupted files).

## FURTHER DEELAES

Deelae.C and .D are based on Deelae.B, but with some minor changes, and one major change. One minor change relates to the file mapping. Both Deelae.A and Deelae.B overwrote the certificate data, and thus mapped the file according to its original size. Deelae.C and Deelae.D increase the file size by 4,213 bytes, even before checking if the file is a candidate for infection. Of course, if an error occurs, then the viruses restore the file to its original size.

The major change is that the viruses will not infect files that have appended data. This is the infection marker – a technique that roy g biv has used for most of his viruses.

The viruses increase the virtual and physical sizes of the last section, and the SizeOfImage value, by 4KB, and then append themselves to the image (perhaps hh86 feels that there are not enough candidates for infection). The file size is increased permanently at this point by 4KB+1, which introduces a potential problem later. The viruses retrieve the RVA of the Delay Import Descriptor and check that it points within the first section. The viruses retrieve the RVA of the Unload Delay Import Table and check that it also points within the first section. If both checks pass, then the viruses hook the first address in the Unload Delay Import Table to point to the virus body. The problem here is that if either pointer is not within the expected range, then the file remains marked as 'infected', and contains the virus body, but there is no pointer to the code. This will be quite a common occurrence, since the Unload Delay Import Table is not used particularly often, and so the pointer will be null.

If the file is infected successfully, then the viruses attempt to zero only the Load Config Table (the Certificate Table data directory is no longer zeroed). However, there is a bug in Deelae.C which is that the destination of the write is a critical field within the section table instead of inside the data directories. As a result, files infected successfully by Deelae.C will often be corrupted. This bug was fixed in Deelae.D.

## YET MORE DEELAES

Deelae.E is based on Deelae.D, but with two minor changes. One change is the field that is used for hooking. Instead of the Unload Delay Import Table, the virus uses the Bound Delay Import Table if the Time Stamp field is non-zero. The virus hooks the first address in the Bound Delay Import Table to point to the virus body.

The other change is the introduction of a bug. Previously, the host entrypoint was stored as an RVA, and adjusted dynamically. This allowed the virus to work with files that had Address Space Layout Randomization (ASLR) enabled. However, since the addresses in the Bound Delay Import Table are Virtual Addresses (VAs), hh86 seems to have assumed that it is acceptable to replace a VA for an external file with a VA for the infected image. This does not work if ASLR is enabled, unless a corresponding relocation item is included, because the host image can move, even if the referenced DLL does not. Note that roy g biv's Bounds viruses have the same problem.

## MAXIMUM DEELAES

Deelae.F is based on Deelae.E, but with the combination of hooking methods from Deelae.E, Deelae.C/.D, and Deelae.B, in that order. That is, if the Time Stamp field is non-zero, then the virus uses the Bound Delay Import Table. If the Time Stamp field is zero, and if the Unload Delay Import Table is non-zero, then the virus uses that table. Otherwise, the virus uses the Delay Import Address Table. As before, the first address in the selected table is hooked, and the original address is stored in the virus body. A jump instruction is modified in the virus body to convert the address to a VA if that is required.

Deelae.F also makes use of the FPU to copy some pointers. This is intended to reduce the code size, but the implementation is flawed, and so the resulting code is larger than if the FPU had not been used at all.

## CONCLUSION

Five new viruses and four kinds of entrypoint in three months. hh86 was the 'Energizer bunny' of the virus-writing community. It's a good thing that her batteries ran out.