MALWARE ANALYSIS 2

AS ABOVE, SOBELOW

Peter Ferrie
Microsoft USA

In June 2009, an interesting article describing 'Heaven's Gate' appeared on a popular VX website. This is an undocumented feature used by the 32-bit *Windows* environment when running on 64-bit versions of *Windows*, which allows for the transition between 32-bit and 64-bit code. In August 2011, we saw the first virus to make use of it: W32/W64.Sobelow.

32-BIT PLATFORM

In infected 32-bit files, the virus begins by identifying the platform. It does this by checking the value of the GS selector. On 32-bit versions of *Windows*, the value of the GS selector is always zero and in this case the virus immediately passes control to the host. On 64-bit versions, the GS selector is always non-zero. If a 64-bit version of *Windows* is detected, the virus aligns the stack to a multiple of eight bytes, if necessary (because the 64-bit environment requires a 64-bit aligned stack, just as the 32-bit environment requires a 32-bit aligned stack). Then the virus uses a tricky method to jump to 64-bit mode.

A jump to 64-bit mode requires the 'magic' 64-bit gate selector ('Heaven's Gate') and the offset that serves as the location from which to resume execution. Since the size of the selector is only 16 bits, that leaves 16 bits unused on a 32-bit platform. The virus takes advantage of this with a single instruction that combines two required elements. The instruction pushes a value onto the stack that corresponds to the selector, and also carries the opcode for a far return instruction. The virus then performs a near call to that far return instruction (which implicitly saves the offset on the stack). The far return passes through the gate, causing the switch to 64-bit mode, and resumes execution at the first instruction after the call instruction – all in a completely position-independent manner. This is a very efficient way to do it.

CURRENT DIRECTORY

Now the virus is in 64-bit mode, but there is still one small change that must be made to the environment to make it usable: the current directory must be set. On 64-bit versions of *Windows*, the 32-bit environment exists as a sub-environment of the 64-bit environment, and a transition to the kernel often requires a traversal through the 64-bit environment to get there. As a result, many of the common data structures exist in two places: one used by the 32-bit mode, and the other used by the 64-bit mode. The current directory is one of these – but since the 32-bit environment can query and set the current directory without invoking the kernel, the 64-bit version is unused in normal circumstances.

Though it is unused in 32-bit mode, the 64-bit version of the current directory would be used in 64-bit mode if related 64-bit APIs were called, and it is not undefined – it is always set to '%windir%'. However, using this directory would pose a problem for the virus because '%windir%' is no longer a great location for finding files to infect. Furthermore, there is no 64-bit version of kernel32.dll in this mode (and it cannot be loaded), so there is no typical user-mode API to change it. In any case, the 32-bit version of the API alters the 32-bit data structure. The 64-bit version of the API is available only to native 64-bit applications. Even if the API were available, the virus would need to know its current location from the 32-bit mode in order to make it the same in the 64-bit mode, which would normally require calling an API from the 32-bit mode.

Instead of all of that, the virus digs straight into the RTL_USER_PROCESS_PARAMETERS structure for the 32-bit and 64-bit modes, thus avoiding the need for any APIs at all. This structure holds the pointer to the current directory. However, instead of simply copying from one to the other, as we might expect, the virus heads straight to an undocumented location nearby and copies the pointer there instead. It turns out that this undocumented location is used by the ntdll RtlDosPathNameToRelativeNtPathName_U() function while resolving the current directory, despite the existence of the official location. This can lead to interesting results if they somehow become unsynchronized – a query of the current directory might return a location other than the actual current directory, along with all of the mischief that implies.

At this point, we reach the start of the 'really' native 64-bit code, and the entrypoint for an infected 64-bit file.

64-BIT PLATFORM

In an infected 64-bit file, the first instruction constructs a pointer to the host entrypoint for execution later. In an infected 32-bit file, the first instruction at this location constructs a pointer to the same far return instruction that was used to enter 64-bit mode. When called from 64-bit mode with the proper parameters (the regular 32-bit code selector and the offset that serves as the location from which to resume execution, as before), it can be used to leave 64-bit mode.

The next instruction saves a value from an undocumented location in memory. In this case, the value corresponds to

the stack pointer in the 64-bit context structure that is active when an exception occurs. Whenever an exception occurs, the context is filled in to allow it to resume afterwards, and a mode switch occurs if necessary. However, in order to prevent recursive calls into the code that performs a mode transition, the value in the undocumented stack location is set to zero.

Since the virus uses exceptions intentionally and intercepts them when they occur, on return to the host, the value in that location would always be zero. If an exception occurred later in the host code (intentional or not), then the exception handler dispatcher in *Windows* would see that the saved stack pointer was invalid and terminate the application. The virus makes sure this value is restored before transferring control to the host.

IMPORT-ANT DETAILS

The virus resolves the address of ntdll.dll by walking some data structures in a way that is compatible with *Windows 7*. As noted above, the 64-bit version of kernel32.dll is not available in this mode, so the virus restricts itself to functions that are available from ntdll.dll, to the extent that no other DLLs are loaded. This includes the System File Checker DLL, so the virus is not able to avoid protected files that meet the infection criteria.

The virus uses the CRC32 method to avoid the need to store the strings, and stores the results onto the stack. The checksums are sorted alphabetically according to the strings they represent, allowing the virus to perform a single pass of the export table in order to resolve all of the APIs required. After resolving the APIs, the virus begins a search for files. The virus searches for files in the current directory and all subdirectories, using a linked list instead of a recursive function. It examines every file that is found, regardless of its extension.

The virus author likes to combine many push instructions for multiple API calls, perhaps as some kind of optimization, but perhaps just to make the code more difficult to analyse. Fortunately, due to some details related to the calling convention that *Microsoft* chose to implement on the 64-bit platform, this is mostly no longer possible. Despite that, we still have the following monstrosity:

```
push    rdi
push    rsi
push    rax
push    rcx
push    rsp
pop     rax
push    rbp
push    rbp
push    rsp
pop     rdi
push    rbp
push    rbp
push    rbp
push    rax
push    ...
push    OBJECT_ATTRIBUTES_size
push    rsp
pop     rsi
push    ...
push    rbp
sub     esp, 20h
mov     r9, rsp
mov     r8, rsi
mov     edx, FILE_WRITE_ATTRIBUTES | SYNCHRONIZE
push    rdi
pop     rcx
```

TOUCH AND GO

When a file is found that meets the infection criteria, it will be infected. Files are considered candidates for infection if they are *Windows* Portable Executable (PE) format, character mode or GUI applications for the *Intel 386+* CPU or the *AMD64*-compatible CPU, with a non-zero entrypoint if they are DLLs. The files must have no digital certificates, and they must have no bytes outside of the image. The latter condition is the infection marker. Interestingly, despite its reliance on exceptions during the infection process, the virus does not check that exceptions are allowed by the host – the NO_SEH (No Structured Exception Handling) flag is not cleared in the header. If the flag is not cleared, then *Windows* will terminate the application at the moment that an exception occurs.

INFECTION

When infecting a file, the virus removes the read-only attribute, if it is present. The virus resizes the file by a random amount in the range of 4–6KB in addition to the size of the virus. This additional data will exist outside of the image, and serve as the infection marker. The virus registers a Vectored Exception Handler to protect against problems, which also intercepts the end of infection exception.

If relocation data is present at the end of the file, the virus will move the data to a larger offset in the file, and place its code in the gap that has been created. If no relocation

data is present at the end of the file, the virus code will be placed here. The virus checks for the presence of relocation data by checking a flag in the PE header. However, this method is unreliable because *Windows* ignores the flag, and relies instead on the base relocation table data directory entry.

The virus increases the physical size of the last section of the file by the size of the virus code, then aligns the result. If the virtual size of the last section is smaller than its new physical size, then the virus sets the virtual size to be equal to the physical size, and increases and aligns the size of the image to compensate for the change.

It also changes the attributes of the last section to include the executable and writable bits. The executable bit is set in order to allow the program to run if DEP is enabled, and the writable bit is set because the virus needs to save the current stack pointer in its body in order to intercept exceptions.

The virus alters the host entrypoint to point to the last section, to the code that is appropriate to the file format. The virus saves the difference between the current entrypoint and the original entrypoint. This allows a transfer of control in a position-independent manner, and allows it to work on files that have Address Space Layout Randomization enabled.

When the infection routine has finished, the virus recalculates the checksum, if necessary, and then forces an exception to occur. This 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 Vectored 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. The common code restores the file date and time, and the read-only attribute if it was set previously. Then the virus searches for another file to infect.

When there are no more files to be found, the virus restores the stack pointer and undocumented value, and then returns to the saved entrypoint. For infected 32-bit files, the stack is further restored to account for any alignment that was performed originally.

THE GATE TO ...

Code that passes through 'Heaven's Gate' is currently immune to most, if not all, anti-malware emulators, but the act of using the gate in this way is suspicious in itself. Perhaps that will be enough to stop the technique before it becomes widespread.