MALWARE ANALYSIS 1

IS OUR VIRUSES LEARNING?

Peter Ferrie
Microsoft, USA

Rarely is that question asked – not least because the grammar is improper. It's rare to see a virus advertised as demonstrating machine learning in any form, but W32/Grimgribber does just that.

INCORRECT VERSION

The virus begins by retrieving the operating system version. If the version does not match the expected value, then the virus skips the anti-emulation trick that follows, and continues execution anyway. Initially, the anti-emulation trick works with only one particular version: *Windows XP*, build 2600. Note that the service pack and patch level (among other things that can affect file structure) are not determined, which restricts the portability of the trick.

The main execution begins by retrieving the operating system version again. This time, though, the value is saved for later use in replicants, so that the anti-emulation trick can be applied on the current platform (no matter which platform it is).

The virus generates a new filename for itself using eight randomly chosen lower-case letters, and then attempts to copy itself to 'c:\<filename>.exe'. Thus, every execution might result in a new file being created. This copy operation will fail on *Windows Vista* and later platforms for non-administrator users.

ALL ORDS INDEX

The virus loads kernel32.dll, and then begins a search for a potential function to use in its anti-emulation trick. For each iteration of the search, there is a one-in-64 chance of skipping the anti-emulation trick entirely. Otherwise, the virus chooses a random ordinal from among the first 1,024 ordinals, and checks whether it maps to an address inside the kernel32.dll.

Once a valid ordinal has been found, the virus saves the current stack pointer, then pushes 16 DWORD zeroes onto the stack as dummy parameters. The number 16 appears to have no special significance. It is probably an arbitrarily chosen, but seemingly safe number, since most, if not all, of the kernel32 APIs require fewer parameters. (FormatMessage is a special case that can potentially exceed this limit, but such a situation occurs only when the string to format has many tokens, and only when those tokens are also passed onto the stack. The API itself has only seven defined parameters.) It's also probably safe to say that if an API were to require that many parameters, then it might not be implemented well and would not be used very much. Passing a single structure parameter would reduce the complexity significantly, and it would also allow for future expansion of the field count in the structure (and thus the indirect parameter count), without requiring a new API.

EXCEPTIONAL HANDLING

The virus registers a Vectored Exception Handler to intercept any problems that might be caused by the null parameters. The most likely problem to occur here would be for a parameter to be assumed to be a valid pointer. Any attempt to use the parameter as a pointer will cause an exception. If the called API has no exception handling mechanism of its own, then the virus Vectored Exception Handler will receive control, since no current kernel32 API registers a Vectored Exception Handler of its own. If the API uses a Structured Exception Handler, then the virus Vectored Exception Handler will still receive control. This is because Vectored Exception Handlers are favoured over Structured Exception Handlers. If the Vectored Exception Handler handles the exception, then the Structured Exception Handler will never receive control.

The situation described above is a bug that exists in the virus, but the effect is unlikely to be visible there. The bug is that since the virus Vectored Exception Handler handles the exception, the API does not complete its execution, and thus does not unregister its Structured Exception Handler. This can lead to unexpected behaviour if an otherwise unhandled exception occurs later, since the API Structured Exception Handler will suddenly receive control for an exception that was not generated during its execution. Depending on the behaviour of the API Structured Exception Handler, stack corruption (or worse) could occur.

The virus attempts to run the API, and checks whether an exception occurs as a result. If an exception occurs, then the virus restores the original stack pointer and searches again for an ordinal. If no exception occurs, then the virus determines the number of parameters that the API used (by comparing the new stack pointer with the original one) and saves this value for use later. The virus then restores the original stack pointer.

REGISTER ARTIFACTS

If no exception has occurred, then the virus generates 16 random numbers to be used as parameter values, and pushes

these values onto the stack. The virus registers a Vectored Exception Handler to intercept any problems that might be caused by the random parameters. The virus attempts to run the API, and checks whether an exception occurs as a result. If an exception occurs, then the virus restores the original stack pointer, and searches again for an ordinal. If no exception occurs, then the virus saves the values of the ECX and EDX registers for use later. The virus then restores the original stack pointer. There is a vanishingly small, but real risk that one of the randomly generated parameters is accepted in a way that could have serious consequences when passed to the 'right' API. For example, the first parameter might point to a DLL filename and be passed to DeleteFile().

As a verification, the virus pushes the same random parameter values onto the stack again, and then randomizes the general register values. This is intended to detect dependencies on the register values. The virus attempts to run the API again with the random parameter values and random register values. If an exception occurs, then the virus restores the original stack pointer, and searches again for an ordinal. If no exception occurs, then the virus checks if the values of the ECX and EDX registers have changed. If either register value has changed, the virus forces an exception to occur, to simulate a failure of the API call. Otherwise, the virus restores the original stack pointer, and will use the random parameter values, the selected API and the register values in the anti-emulation trick.

CODE COMPLETE

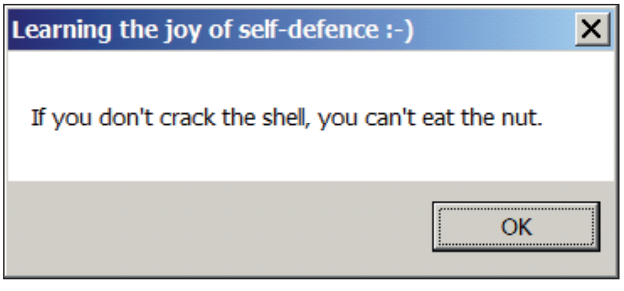
The virus attempts to open the copied file and map a view of it. If this is successful, then it inserts code to call the API. The virus uses the random parameter values chosen earlier, along with the selected API, and adds code to check that the ECX and/or EDX register value(s) match the expected value(s). If either the ECX or EDX register value appears to point into the kernel32.dll image, which is determined by matching the top byte of the register against the top byte of the kernel32 image base, then the corresponding check is skipped.

The anti-emulation trick begins by loading kernel32 and adding the relative virtual address of the selected API. This is an attempt to avoid problems with Address Space Layout Randomization. However, it does nothing to avoid problems with localized and/or patched versions of kernel32.dll. While the operating system version will remain constant even after service packs are applied, and is constant across different language versions, the location of the API can differ enormously as a result. This makes the anti-emulation trick extremely fragile.

After the anti-emulation trick is generated, the virus attempts to create the 'HKLM\Software\Microsoft\Windows\CurrentVersion\Run' key and set the default value to the name of the generated file. This action fails on *Windows Vista* and later for non-administrator users. If it succeeds, then a new file will be created on each reboot, potentially quickly filling the root directory of the boot drive.

The virus creates a hidden file named 'autorun.inf' in the current directory. This contains a reference to the generated filename. The virus enumerates the drive letters from A: to Z:, and queries the drive type. For removable, remote, and RAM disks, the virus copies the autorun file to the root of the drive. For those drive types and also fixed drives, the virus copies the generated file to the root of the drive. After all drives have been examined, the virus waits for a random period of up to a maximum of about 32 seconds, and then checks if the payload should run.

The payload runs only about 1% of the time, and simply displays the following message:



This is apparently a translation of an old Persian quotation. It means that if you don't make the effort, you can't enjoy the reward. The virus then performs the drive enumeration again. This cycle repeats endlessly.

CONCLUSION

So, what have we learned? The virus performs an anti-emulation trick that is so platform-specific, it even defeats itself when it is run on another platform. That's a new meaning for 'self-defence'.

SUMMARY: W32/GRIMGRABBER

Type: Worm.

Size: 4096 bytes.

Targets: local and remote drives, removable media.

Payload: Displays a message.

Removal: Delete detected files.