VIRUS ANALYSIS 1

Sleep-Inducing

Peter Ferrie
Symantec Security Response, Australia

W32/Serot@mm is another creation from Benny the virus-writer. Its name is derived from the word 'serotonin', which is a chemical found in the brain that has been linked to the onset of sleep, among other things. If anyone is wondering whether, this time, Benny has released a bug-free virus … the answer is no. Serot is plagued by programming errors that almost disable it, however some of its capabilities are worth describing, in case another virus appears with these bugs fixed.

Serot uses a 'plug-in' architecture – which was very successful in W95/Hybris. However, the plug-ins in Serot are almost completely self-contained, even carrying their own buffers if the buffers are 'small' enough, resulting in an enormous collection, and much redundancy in the code.

Buggy Benny

Serot is designed for *Windows NT* and later operating systems. The virus checks explicitly for *Windows 9x*/*Me*, and will exit if it is run on any of them. In addition, the virus checks whether NTDLL is accessible using the GetModuleHandleA() API (i.e. that it is resident in memory already), which could affect emulators.

The virus assumes that PSAPI.DLL is present on the system – which is not always the case – and the code will crash if the file does not exist. Serot contains some anti-debugging code which is supposed to cause the virus to exit gracefully if a debugger is detected – due to a bug, however, Serot will usually crash instead.

Attack of the Clones

Next the virus checks whether Serot is running on the system already. It attempts to open a mutex called '$serotonin@', but there is no branch to exit if the open is successful. This results in multiple copies of the virus running at the same time.

Serot contains some variables whose contents are increased or decreased by small random values. However, since there are no bounds checks on the alterations, the contents can become negative values – with disastrous consequences. Serot also relies on the result of several APIs being a certain fixed value, even though they are defined as returning either zero or non-zero. If *Microsoft* should ever change this value, Serot will not work at all.

Serot enumerates processes, looking for the Explorer.exe process, which it uses to remain memory-resident. If the process is found, Serot injects some code into the process, and runs that code.

The injection technique has been used several times by viruses since the first time it was demonstrated, in the W32/Dengue virus, in the year 2000. Previous viruses hooked an API in order to gain control and run the code, for compatibility with *Windows 9x*/*Me*. Since Serot is specific to *Windows NT* and later operating systems, it has no need to hook an API, and can use the CreateRemoteThread() API to run the code.

The Cat that ate the Rat that …

The injected code waits for 10 minutes before executing its main routines. This gives the launch process time to terminate, as well as providing an anti-heuristic device. After the time has elapsed, Serot will attempt to delete the file that was used to launch the code, and the registry key that might have been used to launch the file.

At this point, Serot calls a member of the first group of its plug-ins. The plug-ins are in groups based on type. For example, the first group terminates anti-virus and firewall software, based on the window title; the second group gathers email addresses from the 'mailto:' string in files, from the MAPI Address Lists if *Outlook* is running, or from the Windows Address Book; the fourth group sends the virus by email; the seventh group converts *.NET* files into droppers of the virus.

For groups that contain more than one member, the routine that calls the plug-ins will call a single random member from within that group.

After the first plug-in has returned, Serot generates a keypair for use in encrypted communication with other infected machines, then calls the plug-in that gathers email addresses. If that plug-in returns a failure, then a bug in the code results in the loss of the keypair, and a leakage of the cryptographic context.

Let's Swap

Serot carries a list of IP addresses of known infected machines, and looks for other infected machines by attempting to connect on port 194 to IP addresses found in the registry key 'HKCU\Software\Microsoft\Ftp\Accounts'. There is another routine that attempts to connect to random IP addresses – however, as the result of a bug, this routine is never called.

Port 194 is a well-known port, reserved for the Internet Relay Chat (IRC) protocol, so traffic on this port is not unusual. If a machine is found to be listening on this port, Serot will verify that the machine is infected, by attempting to establish a communication with it using a private protocol. This protocol contains data encrypted using public key cryptography, in order to avoid packet sniffing.

The protocol is established by Serot on the local machine by sending its public key to the server on the remote machine, and examining what is returned. If a valid public key is returned from the remote machine, then the copy of Serot on the local machine will send its current configuration to the remote machine, encrypted with the public key that was returned by the copy of the virus on the remote machine. In return, the copy of the virus on the remote machine will send its current configuration to the local machine, encrypted with the public key that was returned by the copy of Serot on the local machine. Thus, the two copies of the virus are able to exchange behavioural characteristics and their plug-ins.

Several bugs exist in this code, resulting in a number of allocated memory buffers that are never freed.

A New Way to Express It

If no other infected machines are found, Serot will attempt to send itself by email. Yet another bug exists in this code, resulting in the occasional failure to send messages.

The email message uses a 'feature' of *Outlook Express* which involves the UTF-7 encoding of plain-text messages. Using UTF-7 encoding, it is possible to encode HTML message bodies that contain scripts, which are run without user interaction. If the *Internet Explorer* security settings allow Active scripting, and the scripting of ActiveX controls that are not marked as safe, then the script will run without prompts, regardless of the zone in which it is executing.

Additionally, the email message contains no attachments, because Serot creates a message body that contains the virus in an encoded text form.

The function of the script is to decode the virus body and drop it as a file called 'c:\setup.exe'. After the file has been dropped, the registry value 'HKCU\Software\Microsoft\Windows\CurrentVersion\Run\Serotonin' is created so that *Windows* will execute the file whenever the current user logs on. This could also be considered a bug, since if there are multiple users, only the current user will spread the virus.

Dot Dot Dot Net

Once all of the other actions have been performed, Serot begins listening on port 194, and runs the final plug-in. The current version of this plug-in will convert *.NET* executable files into droppers of Serot.

The plug-in begins by searching in every subdirectory on drive C: for files whose name ends in 'exe'. Even the name matching is buggy – not checking for the '.' to indicate a suffix results in the matching of names such as 'SerotRexe'. For every file that is found, the virus checks whether it is a *.NET* file that does not contain resources or relocations, and is not protected by a Strong Name hash. If this check passes, then Serot will attempt to infect the file.

The Compiler at Your Service

Serot infects files by using the Compiler Services methods that are exposed by the *.NET* framework. Using these methods, it is possible to decompile an existing file, add new methods and variables, and recompile the file, without requiring any understanding of the underlying structures.

The recompilation is achieved using just a few method calls in *.NET*, requiring far less effort than the manual reconstruction of standard Portable Executable files that was implemented in W95/ZMist (see *VB*, March 2001, p.6).

A virus using the manual reconstruction technique seems unlikely, since the underlying structures in *.NET* are extremely complex and contain many interdependencies. For example, an entry in the method table contains references to the #Strings stream and the #Blob stream. The entry in the #Blob stream contains references into the TypeRefs table. The TypeRefs table contains references into the #Strings stream, and also coded index references into the AssemblyRefs table. The AssemblyRefs table also contains references into the #Strings stream and the #Blob stream. It would take a long time for someone to find out what all of the links are, and work out how to update them manually.

After creating the new code that will drop and run a copy of the virus, Serot attempts to access the file that contains the data that will be dropped. However, errors in the control flow mean that the file the virus attempts to access might not have been created at all.

If the file does exist, Serot will reserve 64Mb of memory for itself, but never free it, resulting in the eventual exhaustion of system resources, since this allocation occurs for every *.NET* file on a machine.

More memory is not freed if a particular *.NET* DLL cannot be loaded, and because of another bug, that dll usually cannot be loaded. Serot also assumes that the infection will always succeed, and uses the MoveFile() API to replace the original file, resulting in deletion of the original file if an error occurred.

Conclusion

Despite its potential, this very buggy creation of Benny's snatches defeat from the jaws of victory. However, it demonstrates the Compiler Services, which seem likely to form the basis for future *.NET* file infectors. A recompiling virus like W95/Anxiety, but without needing the source code, combined with an inserting virus like W95/ZMist, but without rebuilding the file manually … The beast is unleashed, and its full power is unknown.