

(step 456). Otherwise, if the procedure obtains the name of the new SCR, the procedure continues to step 455.

[0151] 5. In step 455, the procedure evaluates the newly found SCR. The evaluation involves checking the available information contained in the suspected module, for example, the list of legal SCRs that the current process expects to use, checksums, etc. If the procedure concludes that the new SCR is produced by a valid source, no alert is issued (step 457). If, however, the SCR is determined to be a suspected one, an alert is issued (step 456).

[0152] In the Embodiment of FIG. 5

[0153] In the embodiment of FIG. 5 the sensor performs more than one of the procedures as described in FIGS. 4A-4E. If a procedure of any of said tests detects with certainty an illegal action, an alert is issued. However, if non of said tests provides a result with certainty, a weight is given to each test result, and if the accumulated result of all the tests is found to be above a threshold value, an alert is issued.

[0154] 1. In step 291, the sensor accumulates information from a plurality of tests. The tests of block 291 are dealing with the comparison of structure by verification of characteristics such as: the number of SCRs within the stack; the chain order of the SCRs within the stack; the time-stamps of the SCRs within the stack; the names of the SCRs within the stack; a signature of each SCR within the stack; the number of bits of each SCR within the stack; a checksum of each SCR within the stack; the physical path and name of each SCR within the stack. Some of these tests and few orthogonal tests are elaborated by the block diagrams of FIGS. 4A-4E, and their corresponding descriptions.

[0155] 2. In step 292 a weight is given to each accumulated result, and a combined result is calculated. This is explained by FIG. 5.

[0156] 3. In step 294 the combined result of all tests is compared with a preset threshold value, as registered in knowledge base 150.

[0157] 4. If the combined result in step 294 is found to be above the threshold, an alert is issued.

[0158] 5. If, however, the combined result in step 294 is found to be below the threshold, no alert is issued.

[0159] General Considerations

[0160] 1. A private sensor may also cover almost all the cases that are covered by a public sensor. Public sensors better handle active offenders of the 'brain-transplanting' type (i.e., offenders that try to modify the behavior of a process). However, when implementing a private sensor, the special activities of the sensor's probe (which is an SCR) typically go into a separate thread, to minimize the extra load on the protected component.

[0161] 2. Hybrid sensors, such as the sensor of FIG. 2B (a high-level component coupled with a 'guaranteed' bottom-level handler), are best for detecting silent manipulators, including those of the 'direct-approach' type, which do not communicate with their originating process.

[0162] 3. Preferably, the sensor comprises an authorized 'learning' program which may be operated periodically to set and tune the threshold values by analyzing the sensor's performance. It may further tune the weights of inputs to the threshold function, change action parameters (e.g., to freeze an offender or not to freeze), and enhance the small heuristic knowledge base of the sensor (e.g., a digest of distinguished offenders).

[0163] 4. A risk-assessor program would weigh current threats against available system resources and ask a load-balancing program to load or unload sensors (or other agents) as needed.

EXAMPLES

[0164] The following are some examples for possible implementations of some of the concepts that are described herein. The implementations should run on 32 bit Windows™ operated machines. More particularly, the two offender mechanisms that are described here can run on both Win9x and NT, while the defender mechanism can run as is on Win9x, and a slight modification enables it to run on NT as well.

[0165] The concepts and mechanisms described here may of course be adapted to other Operating Systems. Furthermore, even on the OSs referenced herein, namely Windows™, there are many SCR chains, beyond the Windows Message Hook mechanism, that may be exploited using the principles and concepts that are described herein above.

[0166] The description herein is not meant to be fully detailed or comprehensive: it is given here just for providing an intuitive understanding of the mechanism. Many details are omitted for the sake of brevity while keeping the essence clear.

[0167] Following are the descriptions of two offender mechanisms, a description of a defender for the second scenario; this defender mechanism may be adapted to the other first offender mechanism with a slight modification. Thereafter, some notes are provided, concerning reference material and technical details.

[0168] This appendix should be read and interpreted only within the context of the main text.

[0169] Offender, Mechanism #1

[0170] 1. The offender launching program, LAUNCHER1.EXE, initializes its connection to the windows-hooks stack and does other common startup things.

[0171] 2. LAUNCHER1.EXE looks for its victim, APP.EXE, by calling ::GetWindow. If found, it finds its thread ID by calling ::GetWindowThreadProcessId and passes it to CatchInnocentApp, a function that is supplied by HELPER.DLL.

[0172] 3. Function CatchInnocentApp retrieves its current thread ID by calling ::GetCurrentThreadId and stores it for global use, then it calls ::SetWindowHookEx(WH_GETMESSAGE, . . .) on the victim's thread ID.

[0173] 4. It then calls ::PostThreadMessage on that thread, passing it a WM_NULL or other nonsense message, just to activate the hook on APP.EXE.