(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 October 2006 (19.10.2006)

PCT

(10) International Publication Number
WO 2006/110921 A2

(51) International Patent Classification:
G06F 12/14 (2006.01)

(21) International Application Number: PCT/US2006/014405

(22) International Filing Date: 14 April 2006 (14.04.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/105,977 14 April 2005 (14.04.2005) US

(71) Applicant *(for all designated States except US)*: **WEBROOT SOFTWARE, INC.** [US/US]; 2560 55th Street, Suite 300, Boulder, Colorado 80308 (US).

(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: **HORNE, Jefferson Delk** [US/US]; 6255 Habitat Drive, No. 1004, Boulder, Colorado 80301 (US).

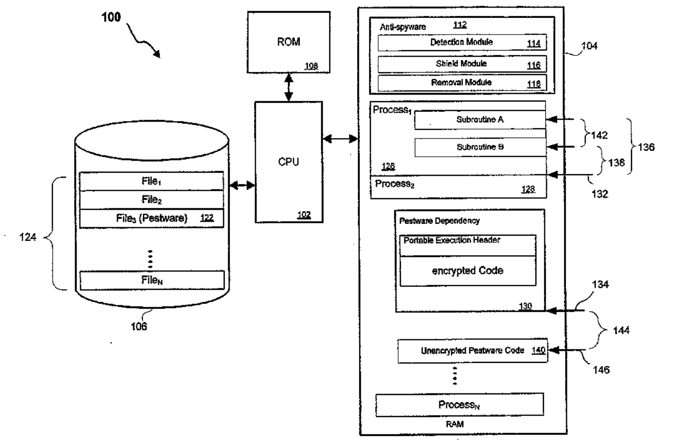
(74) Agent: DESANCTIS, Michael A.; Faegre & Benson LLP, Customer No. 35657, 3200 Wells Fargo Center, 1700 Lincoln Street, Denver, Colorado 80203-4532 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

(54) Title: SYSTEM AND METHOD FOR SCANNING MEMORY FOR PESTWARE OFFSET SIGNATURES



(57) Abstract: Systems and methods for managing pestware processes on a protected computer are described. In one implementation, a reference point in the executable memory that is associated with a process running in the executable memory is located. A first and second sets of information from corresponding first and second portions of the executable memory are then retrieved. The first and second portions of the executable memory are separated by a defined offset, and each of the first and second portions of the executable memory are offset from the reference point. The process is identifiable as a particular type of pestware when the first and second sets of information each include information previously found to be separated by the defined offset in other processes that are of the particular type of pestware. In some variations, the reference point is a starting address and/or an API implementation in the process.

WO 2006/110921 A2

SYSTEM AND METHOD FOR SCANNING MEMORY FOR PESTWARE OFFSET SIGNATURES

RELATED APPLICATIONS

[0001] The present application is related to the following commonly owned and assigned applications: U.S. Patent Application No. 11/105,978, and U.S. Patent Application No. 11/106,122, both of which are incorporated by reference in their entirety.

FIELD OF THE INVENTION

[0002] The present invention relates to computer system management. In particular, but not by way of limitation, the present invention relates to systems and methods for controlling pestware or malware.

BACKGROUND OF THE INVENTION

[0003] Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as "malware" or "pestware." These types of programs generally act to gather information about a person or organization—often without the person or organization's knowledge. Some pestware is highly malicious. Other pestware is non-malicious but may cause issues with privacy or system performance. And yet other pestware is actual beneficial or wanted by the user. Wanted pestware is sometimes not characterized as "pestware" or "spyware." But, unless specified otherwise, "pestware" as used herein refers to any program that collects and/or reports information about a person or an organization and any "watcher processes" related to the pestware.

[0004] Software is available to detect some pestware, but many variations of pestware are difficult to detect with typical techniques. For example, pestware running in memory of a computer is often difficult to detect because it is disguised in such a way that it appears to be a legitimate process that is dependent from a trusted application (e.g., a word processor application). In other cases, pestware is obfuscated with encryption techniques so that a pestware file stored on a system hard drive may not be readily recognizable as a file that has spawned a pestware process. In yet other instances, pestware is known to be polymorphic in nature so as to change its size in memory or to change its starting address in memory. Accordingly, current software is not always able to scan and remove pestware in a convenient manner and will most certainly not be satisfactory in the future.

SUMMARY OF THE INVENTION

[0005] Exemplary embodiments of the present invention that are shown in the drawings are summarized below. These and other embodiments are more fully described in the Detailed Description section. It is to be understood, however, that there is no intention to limit the invention to the forms described in this Summary of the Invention or in the Detailed Description. One skilled in the art can recognize that there are numerous modifications, equivalents and alternative constructions that fall within the spirit and scope of the invention as expressed in the claims.

[0006] Embodiments of the present invention include methods for scanning files of a protected computer for pestware. One embodiment is configured to locate a reference point in the executable memory that is associated with a process running in the executable memory and retrieving a first set of information from a first portion of the executable memory and a second set of information from a second portion of the executable memory. The first and second portions of the executable memory are separated by a defined offset, and each of the first and second portions of the executable memory are offset from the reference point. The process is identified as a particular type of pestware when the first and second sets of information each include information previously found to be separated by the defined offset in other processes that are of the particular type of pestware.

[0007] In another embodiment, the invention may be characterized as a system for managing pestware. In this embodiment, a pestware detection module is configured to detect pestware on a protected computer, which includes a file storage device and a program memory. The system also includes a pestware removal module configured to remove pestware on the protected computer. In this embodiment, the pestware detection module configured to locate a reference point in the executable memory that is associated with a process running in the executable memory and to retrieve a first set of information from a first portion of the executable memory and a second set of information from a second portion of the executable memory. The first and second portions of the executable memory are separated by a defined offset, and each of the first and second portions of the executable memory are offset from the reference point. The pestware detection module is configured to identify the process as a particular type of pestware when the first and second sets of information each include information previously found to be separated by the defined offset in other processes that are of the particular type of pestware.

[0008] These and other embodiments are described in more detail herein.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] Various objects and advantages and a more complete understanding of the present invention are apparent and more readily appreciated by reference to the following Detailed Description and to the appended claims when taken in conjunction with the accompanying Drawings where like or similar elements are designated with identical reference numerals throughout the several views and wherein:

FIGURE 1 illustrates a block diagram of a protected computer in accordance with one implementation of the present invention;

FIGURE 2 is a flowchart of one method for locating portions of memory associated with processes and process dependencies running in memory of the protected computer; and

FIGURE 3 is a flowchart of a method for scanning the processes and process dependencies so as to identify pestware running on the protected computer.

DETAILED DESCRIPTION

[0010] Referring first to FIGURE 1, shown is a block diagram 100 of a protected computer/system in accordance with one implementation of the present invention. The term "protected computer" is used herein to refer to any type of computer system, including personal computers, handheld computers, servers, firewalls, etc. This implementation includes a CPU 102 coupled to memory 104 (e.g., random access memory (RAM)), a file storage device 106 and ROM 108.

[0011] As shown, the storage device 106 provides storage for a collection of *N* files 124, which includes a pestware file 122. The storage device 106 is described herein in several implementations as hard disk drive for convenience, but this is certainly not required, and one of ordinary skill in the art will recognize that other storage media may be utilized without departing from the scope of the present invention. In addition, one of ordinary skill in the art will recognize that the storage device 106, which is depicted for convenience as a single storage device, maybe realized by multiple (e.g., distributed) storage devices.

[0012] As shown, an anti-spyware application 112 includes a detection module 114, a shield module 116 and a removal module 118, which are implemented in software and are executed from the memory 104 by the CPU 102. The software 112 can be configured to operate on personal computers (e.g., handheld, notebook or desktop), servers or any device capable of processing instructions embodied in executable code. Moreover, one of ordinary skill in the art will recognize that alternative embodiments, which implement one or more components (e.g., the anti-spyware 112) in hardware, are well within the scope of the present invention.

[0013] Also shown in the executable memory 104 are TV processes, which in general, are executable programs that may be either known and trusted applications or pestware processes that are being executed by the CPU 102. Shown among the TV processes for example, are a first process 126 that is a pestware process and a second process 128 that is not a pestware process, but it is associated with a pestware dependency 130, which has spawned unencrypted pestware code 140. As discussed further herein with reference to FIGURES 2 and 3, several embodiments of the present invention effectively and quickly identify and remove the pestware 126, 130, 140.

[0014] It should be recognized that an operating system (not shown) of the protected computer 100 is not limited to any particular type of operating system and may be operating systems provided by Microsoft Corp. under the trade name WINDOWS (e.g., WINDOWS 2000, WINDOWS XP, and WINDOWS NT). Additionally, the operating system may be an open source operating system such operating systems distributed under the LINUX trade name. For convenience, however, embodiments of the present invention are generally described herein with relation to WINDOWS-based systems. Those of skill in the art can easily adapt these implementations for other types of operating systems or computer systems.

[0015] Referring next to FIGURE 2, shown is a flowchart depicting steps traversed in accordance with a method for scanning processes (e.g., the Nprocesses) in executable memory for pestware. hi the exemplary embodiment, the steps enumerated in FIGURE 2 are carried out by the detection module 114 of the anti-spyware application 112. As shown in FIGURE 2, blocks of memory associated with the TV processes and any dependencies associated with each process are initially enumerated (Blocks 202, 204). hi addition, the base address of each of the ./V processes and any dependencies associated with each of the TV processes are enumerated (Block 206). With respect to the exemplary processes depicted in FIGURE 1, blocks of memory and the base addresses associated with the //processes and the pestware dependency 130 are enumerated.

[0016] As shown in FIGURE 2, once the base addresses for the processes and dependencies are identified, portable execution (PE) headers for the processes and dependencies are obtained (Block 208), and the original entry point (OEP) for each process and each dependency are obtained from the processes' and dependencies' respective portable execution headers (Block 210). Next, the start address of each process and each dependency is calculated by adding the OEP and base address of each process and each dependency together (Block 212).

[0017] Referring to FIGURE 1, for example, a start address 132 of the first pestware process 126 is calculated and another start address 134 of the pestware dependency 130 is calculated. Advantageously, the steps outlined with reference to blocks 202-212 allow a reference point in executable memory (e.g., memory 104) to be established relative to processes and dependencies that have neither a corresponding file in the file storage device 106 nor have a process identifier (PID). For example, the pestware dependency 130 may be a dynamic link library (DLL), that does not have a process PID, but several embodiments of the present invention enable the block of memory associated with the DLL to be enumerated and allow a base reference point (e.g., the start address 134) for the dependency 130 to be identified.

[0018] After calculating the start addresses of the processes (e.g., the //processes) and any dependencies (e.g, the dependency 130) each of the processes and any dependencies are scanned utilizing their respective start addresses (Block 214). It should be recognized that, due to various techniques (e.g., morphing techniques) the calculated starting address discussed above with reference to block 212 may not yield a viable base location from which to scan the associated process/dependency. To address this situation, some embodiments of the present invention also locate one or more API implementations, which are then utilized as a reference point to scan the associated process.

[0019] Referring next to FIGURE 3, shown is a flowchart depicting steps carried out in accordance with a dynamic offset scanning technique in accordance with several embodiments of the present invention. In general, the offset scanning technique in these embodiments scans, utilizing offsets from a base location (e.g., a start address and/or an API implementation) only portions of the executable memory that are associated with each of the processes (e.g., the W processes) so as to scan the processes quickly. But the portions of the executable memory that are scanned are selected so as to provide for very effective scanning. Moreover, in many variations of the present invention, the information obtained from the selected portions of the executable memory is op code that provides very specific information about the processes so as to avoid false positives.

[0020] As shown in FIGURE 3 with simultaneous reference to FIGURE 1, in several embodiments, portions of the files in storage device 106 are scanned along with portions of the ./V processes and the dependency 130 in memory 104 so as to provide an initial assessment the types of processes that are residing in the executable memory 104 (Blocks 302, 304, 306).

[0021] Based upon the information gathered with the initial scans (Blocks 304, 306), specific offsets are selected for scanning each of the *Nprocesses* and the dependency 130 (Block 308). For example, a partial scan of the pestware dependency 130 reveals that it is a loader for encrypted code, and as a consequence, a specific offset 144 from the start address 134 of the pestware dependency 130 to a memory location 146 that is outside of the memory block for the pestware dependency 130 is selected in order to scan for unencrypted pestware code known to be located at the offset 144 from its associated loader when running in memory.

[0022] As another example, a first offset 136 and a second offset 138 are selected relative to the start address 132 of the first pestware process 126. As shown, these offsets 136, 138 from the start address 132 point to portions of the executable memory 104 where two specific subroutines for the first pestware process reside. These offsets 136 and 138 are selected based upon known pestware of the type matching the first pestware process 126.

[0023] It should be recognized that other base reference locations may be utilized for the processes and dependencies in addition to (or instead of) a start address. For example, it is often the case that a relative offset between portions of code within a pestware process is static even though the start address of the process may vary. In some embodiments for example, API implementations are located and utilized as base reference points when the start address is not a viable reference point, hi such a case, one or more offsets from the API implementation may be scanned for code that is known to be associated with particular pestware.

[0024] As shown in FIGURE 3, for each process and each dependency, portions of code are retrieved from locations in memory that are located at the selected offsets from the base reference location associated with each process and each dependency (Block 310). The code that is retrieved from memory at the locations that are offset from a reference base of a particular process/dependency is then compared with code associated with known pestware so as to determine whether the particular process/dependency is pestware (Block 312).

[0025] In several embodiments, the code retrieved at the offsets is op code (e.g., X86 assembly instructions) as opposed to strings or flat Unicode text. In this way, the identification of pestware is much more accurate because the op code associated with known pestware is very specific to the pestware. As a consequence, the frequency of false positive identifications of the process as pestware is substantially reduced.

[0026] It should be recognized that the process depicted in FIGURE 3 is exemplary only and that one of ordinary skill in the art will appreciate that one or more steps may be varied and or omitted without departing from the scope of the present invention. For example, the steps enumerated by blocks 304 and 306 may be varied or omitted, and each process/dependency may be scanned by scanning memory associated with each process/dependency with various offsets from the base reference point of the process/dependency without the benefit of narrowing the number of offsets utilized.

[0027] In conclusion, the present invention provides, among other things, a system and method for managing pestware. Those skilled in the art can readily recognize that numerous variations and substitutions may be made in the invention, its use and its configuration to achieve substantially the same results as achieved by the embodiments described herein. Accordingly, there is no intention to limit the invention to the disclosed exemplary forms. Many variations, modifications and alternative constructions fall within the scope and spirit of the disclosed invention as expressed in the claims.

WHAT IS CLAIMED IS:

1. A method for scanning executable memory of a protected computer for pestware comprising:

identifying at least one reference point in the executable memory of the protected computer, wherein the at least one reference point is associated with a process running in the memory of the protected computer, wherein the process is potentially a particular type of pestware;

selecting, as a function of the particular type of pestware, a first offset and a second offset;

accessing the memory at the first offset from the at least one reference point so as to identify a first set of information in the executable memory that begins at the first offset from the at least one reference point;

accessing the memory at the second offset from the at least one reference point so as to identify a second set of information in the executable memory that begins at the second offset from the at least one reference point; and

wherein the first and second sets of information are separated in the executable memory by information not included in the first and second sets of information, and wherein the process is identifiable as the particular type of pestware when the first and second sets of information each include information associated with the particular type of pestware.

2. The method of claim 1, wherein the identifying includes identifying a start address as the at least one reference point.

3. The method of claim 1, wherein the identifying includes identifying a particular API implementation as the at least one reference point.

4. The method of claim 1, wherein accessing the memory at the first and second offsets includes accessing executable op code at the first and second offsets so as to identify executable op code in the first and second sets of information.

5. A method for scanning executable memory of a protected system for pestware comprising:

locating a reference point in the executable memory that is associated with a process running in the executable memory;

retrieving a first set of information from a first portion of the executable memory and a second set of information from a second portion of the executable memory, wherein the first and second portions of the executable memory are separated by a defined offset, and wherein each of the first and second portions of the executable memory are offset from the reference point; and

identifying the process as a particular type of pestware when the first and second sets of information each include information previously found to be separated by the defined offset in other processes that are of the particular type of pestware.

6. The method of claim 5, wherein the locating the reference point includes locating a starting address of the process.

7. The method of claim 5, wherein the locating the reference point includes locating an API implementation in the process.

8. The method of claim 5, wherein the retrieving includes retrieving op code from the first and second portions of the executable memory.

9. The method of claim 5 including:

scanning a file of a hard drive that is associated with the process so as to obtain information about the file, wherein the defined offset varies based upon the information about the file.

10. A system for managing pesrware comprising:

a pestware removal module configured to remove pestware on a protected computer, the protected computer including at least one file storage device and an executable memory; and

a pesrware detection module configured to:

locate a reference point in the executable memory that is associated with

a process running in the executable memory;

retrieve a first set of information from a first portion of the executable memory and a second set of information from a second portion of the executable memory, wherein the first and second portions of the executable memory are separated by a defined offset, and wherein each of the first and second portions of the executable memory are offset from the reference point; and identify the process as a particular type of pestware when the first and second sets of information each include information previously found to be separated by the defined offset in other processes that are of the particular type of pestware.

11. The system of claim 10, wherein the pestware detection module is configured to locate a starting address as the reference point.

12. The system of claim 10, wherein the pestware detection module is configured to locate an API implementation as the reference point.

13. The system of claim 10, wherein the pestware detection module is configured to retrieve op code from the first and second portions of the executable memory.

14. The system of claim 10, wherein the pestware detection module is configured to:

scan a file of a hard drive that is associated with the process so as to obtain information about the file, and wherein the the defined offset varies based upon the information about the file.

15. A computer readable medium encoded with instructions for scanning executable memory on a protected computer for pestware, the instructions including instructions for:

locating a reference point in the executable memory that is associated with a process running in the executable memory;

retrieving a first set of information from a first portion of the executable memory and a second set of information from a second portion of the executable memory, wherein the first and second portions of the executable memory are separated by a defined offset, and wherein each of the first and second portions of the executable memory are offset from the reference point; and

identifying the process as a particular type of pestware when the first and second sets of information each include information previously found to be separated by the defined offset in other processes that are of the particular type of pestware.

16. The computer readable medium of claim 15, wherein the instructions for locating the reference point include instructions for locating a starting address of the process.

17. The computer readable medium of claim 15, wherein the instructions for locating the reference point include instructions for locating an API implementation in the process.

18. The computer readable medium of claim 15, wherein the instructions for retrieving include instructions for retrieving op code from the first and second portions of the executable memory.

19. The computer readable medium of claim 15 including instructions for scanning a file of a hard drive that is associated with the process so as to obtain information about the file, wherein the defined offset varies based upon the information about the file.

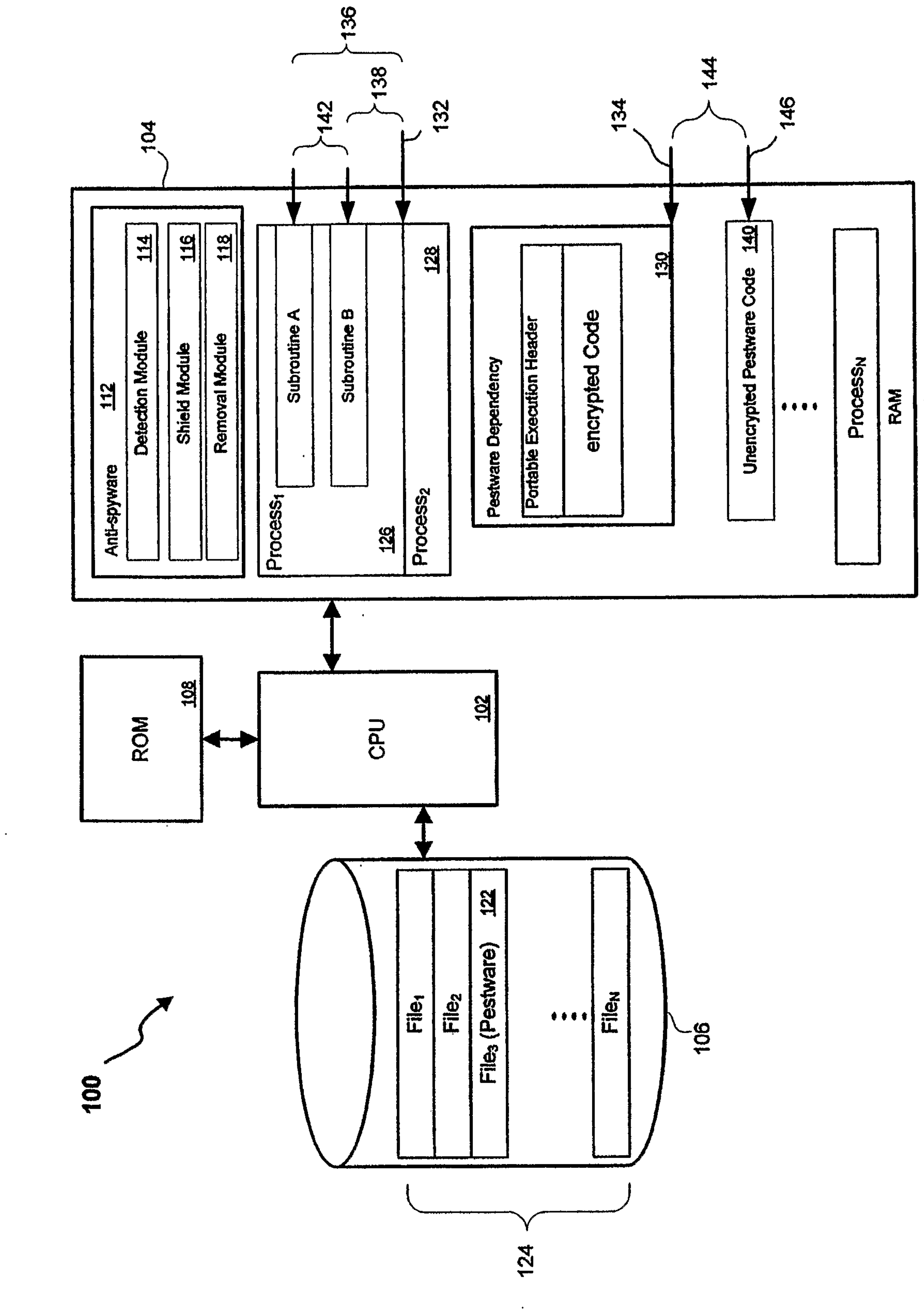

100
104
Anti-spyware 112
Detection Module 114
Shield Module 116
Removal Module 118
Process₁ 126
Subroutine A
Subroutine B
Process₂ 128
Pestware Dependency
Portable Execution Header
encrypted Code
130
Unencrypted Pestware Code 140
Process_N
RAM
136
138
142
132
134
144
146
ROM 108
CPU 102
File₁
File₂
File₃ (Pestware) 122
File_N
106
124

FIGURE 1

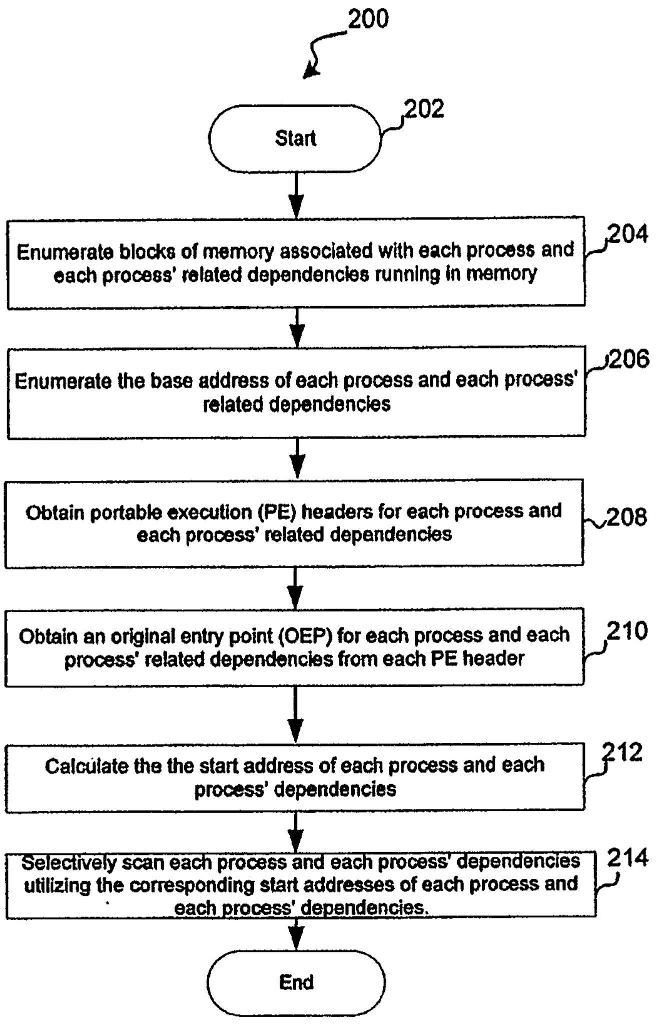

200
202
Start
Enumerate blocks of memory associated with each process and each process' related dependencies running in memory
204
Enumerate the base address of each process and each process' related dependencies
206
Obtain portable execution (PE) headers for each process and each process' related dependencies
208
Obtain an original entry point (OEP) for each process and each process' related dependencies from each PE header
210
Calculate the the start address of each process and each process' dependencies
212
Selectively scan each process and each process' dependencies utilizing the corresponding start addresses of each process and each process' dependencies.
214
End

FIGURE 2

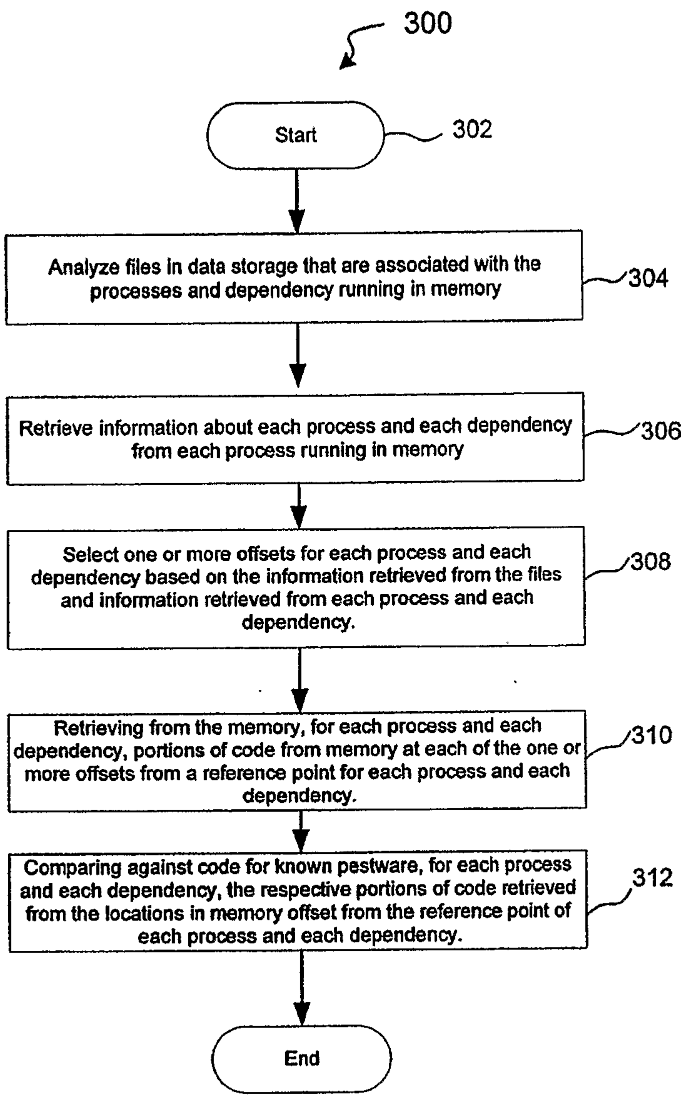

300
Start
302
Analyze files in data storage that are associated with the processes and dependency running in memory
304
Retrieve information about each process and each dependency from each process running in memory
306
Select one or more offsets for each process and each dependency based on the information retrieved from the files and information retrieved from each process and each dependency.
308
Retrieving from the memory, for each process and each dependency, portions of code from memory at each of the one or more offsets from a reference point for each process and each dependency.
310
Comparing against code for known pestware, for each process and each dependency, the respective portions of code retrieved from the locations in memory offset from the reference point of each process and each dependency.
312
End

FIGURE 3