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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
3 February 2011 (03.02.2011)

PCT

(10) International Publication Number
WO 2011/014623 A1

(51) **International Patent Classification**:
G06F 11/07 (2006.01) *G06F 21/00* (2006.01)

(21) **International Application Number**:
PCT/US2010/043660

(22) **International Filing Date**:
29 July 2010 (29.07.2010)

(25) **Filing Language**: English

(26) **Publication Language**: English

(30) **Priority Data**:
61/229,497 29 July 2009 (29.07.2009) US

(71) **Applicant** *(for all designated States except US)*: **REVERSINGLABS CORPORATION** [US/US]; 169 Msgr. O'Brian Highway, Apt. 802, Cambridge, MA 02141 (US).

(72) **Inventor; and**

(75) **Inventor/Applicant** *(for US only)*: **PERICIN, Tomislav** [RS/RS]; Matijie Hudji 76/2, Sremska Mitrovica, 22000 (RS).

(74) **Agent**: **PLACKER, Jeffrey, T.**; Holland & Knight LLP, 10 St. James Avenue, Boston, MA 02116 (US).

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

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) **Title**: PORTABLE EXECUTABLE FILE ANALYSIS

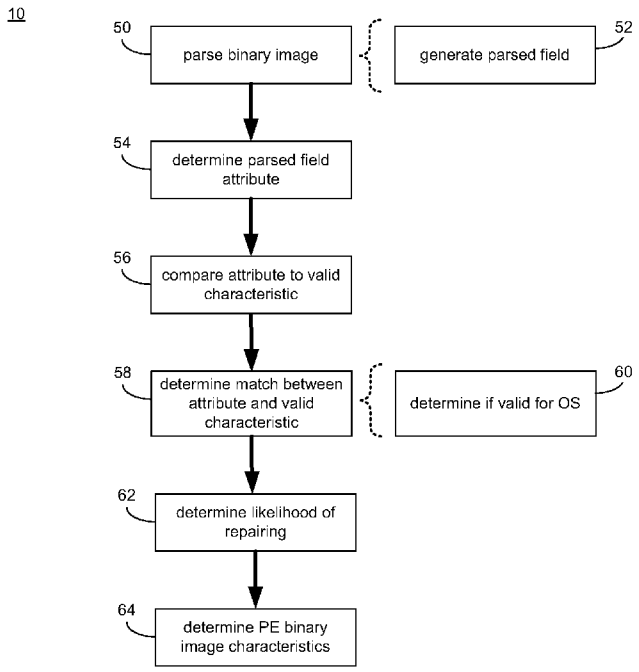

FIG. 2

(57) **Abstract**: A portable executable file is analyzed by parsing a binary image of the portable executable file to generate a parsed field. An attribute of the parsed field is determined. The attribute of the parsed field is compared to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification. It is determined if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

WO 2011/014623 A1

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

PORTABLE EXECUTABLE FILE ANALYSIS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. provisional patent application Serial No. 61/229,497, filed on 29 July 2009, the entire disclosure of which is incorporated herein by reference.

TECHNICAL FIELD

[0002] This application generally relates reverse engineering software, and more particularly relates to unpacking software and validity analysis of software files.

BACKGROUND OF THE DISCLOSURE

[0003] Portable executable file format (PE file format), as defined by Microsoft Corporation in the "Microsoft Portable Executable and Common Object File Format Specification" is a file format for executables, object code and DLL's (dynamic link libraries). PE files are used in 32-bit and 64-bit versions of the Microsoft Windows operating systems. The PE file format is a highly versatile format that can be used in numerous operating system environments and supports various processor solutions.

[0004] Software developers may use various schemes to protect software, including PE files. For example, software packers may be utilized to compress binaries, which may decrease bandwidth usage associated with transferring the binaries and storage volume. Similarly, packers may be utilized to protect intellectual property embodied within the software and to prevent code theft. Packing may involve various schemes of compression and/or encryption that may obfuscate the contents of the executable code. Running the packed executable file may unpack the original executable code (e.g., which may include decompressing and/or decrypting) and then transferring control to the original executable code. As such, the nature of the executable code may not be known until the software is actually executing. This can be problematic, for example, if the executable code is malware or other undesirable software, as the nature of the software may not be known until it is too late.

SUMMARY OF THE DISCLOSURE

[0005] According to a first implementation, a computer implemented method includes parsing, by a computing device, a binary image of a portable executable file to generate a parsed field, and determining, by the computing device, an attribute of the parsed field. The method also includes comparing, by the computing device, the attribute of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification. The method further includes determining, by the computing device, if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

[0006] One or more of the following features may also be included. The parsed field may include one or more of a portable executable format signature, an ImageBase field, a SizeOfImage field, a FileAlignment field, a SectionAlignment field, an EntryPoint address, an import table, an import address table, an export table, a relocation table, a resource table, a thread local storage table, a load configuration table, a bound import table, a COM table, and a portable executable section table. The attribute of the parsed field may include one or more of a field identifier, a field length, and a field content.

[0007] Determining if the attribute of the parsed field matches the valid characteristic of the valid corresponding field may include determining, by the computing device, if the attribute of the parsed field is valid for a predetermined operating system. It may be determined if the parsed field does not match the valid characteristic of the valid corresponding field. If the parsed field does not match the valid characteristic of the valid corresponding field, a likelihood of modifying the parsed field that does not match the valid characteristic of the valid corresponding field to generate a valid field may be determined.

[0008] The method may also include determining, by the computing device, if the binary image of the portable executable file includes a dynamic link library, a kernel driver, or an executable object.

[0009] According to another implementation, a computer program product includes a computer readable medium having a plurality of instructions stored on it. When executed by a processor, the instructions cause the processor to perform operations including parsing a binary image of a portable executable file to generate a parsed field, and determining an attribute of the parsed field. The attribute of the parsed field

is compared to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification. It is determined if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

[0010] One or more of the following features may be included. The parsed field may include one or more of a portable executable format signature, an ImageBase field, a SizeOfImage field, a FileAlignment field, a SectionAlignment field, an EntryPoint address, an import table, an import address table, an export table, a relocation table, a resource table, a thread local storage table, a load configuration table, a bound import table, a COM table, and a portable executable section table. The attribute of the parsed field may include one or more of a field identifier, a field length, and a field content.

[0011] Determining if the attribute of the parsed field matches the valid characteristic of the valid corresponding field may include determining if the attribute of the parsed field is valid for a predetermined operating system. It may be identified if the parsed field does not match the valid characteristic of the valid corresponding field.

[0012] A likelihood of modifying the parsed field that does not match the valid characteristic of the valid corresponding field to generate a valid field may be determined. Further, it may be determined if the binary image of the portable executable file includes a dynamic link library, a kernel driver, or an executable object.

[0013] According to yet another implementation, a system includes a processor, and a memory coupled with the processor. A first software module is executable by the processor and the memory. The first software module is configured to parse a binary image of a portable executable file to generate a parsed field. A second software module is executable the by processor and the memory. The second software module is configured to determine an attribute of the parsed field. A third software module is executable by the processor and the memory. The third software module is configured to compare the attribute of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification. A fourth software module is executable by the processor and the memory. The fourth software module is configured to determine if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

[0014] One or more of the following features may be included. The parsed field may include one or more of a portable executable format signature, an ImageBase field, a SizeOfImage field, a FileAlignment field, a SectionAlignment field, an EntryPoint address, an import table, an import address table, an export table, a relocation table, a resource table, a thread local storage table, a load configuration table, a bound import table, a COM table, and a portable executable section table. The attribute of the parsed field may include one or more of a field identifier, a field length, and a field content.

[0015] The fourth software module, which is configured to determine if the attribute of the parsed field matches the valid characteristic of the valid corresponding field, may further be configured to determine if the attribute of the parsed field is valid for a predetermined operating system. A fifth software module may be executable by the processor and the memory. The fifth software module may be configured to identify if the parsed field does not match the valid characteristic of the valid corresponding field.

[0016] A sixth software module may be executable by the processor and the memory. The sixth software module may be configured to determine a likelihood of modifying the parsed field that does not match the valid characteristic of the valid corresponding field to generate a valid field. A seventh software module may be executable by the processor and the memory. The seventh software module may be configured to determine if the binary image of the portable executable file includes a dynamic link library, a kernel driver, or an executable object.

[0017] The details of one or more implementations are set forth in the accompanying drawings and the description below. Other features and advantages will become apparent from the description, the drawings, and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] FIG. 1 schematically depicts a computing device that may execute one or more of a validity check process, a repair process and a automated unpacking process.

[0019] FIG. 2 is a flow diagram of a process performed by the validity check process of FIG. 1.

[0020] FIG. 3 is a flow diagram of a process performed by the repair process of FIG. 1.

[0021] FIG. 4 is a flow diagram of a process performed by the automated unpacking process of FIG. 1.

DETAILED DESCRIPTION OF EXEMPLARY EMBODIMENTS

[0022] As will be appreciated by one skilled in the art, the present invention may be embodied as a system, method or computer program product. Accordingly, the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment (including firmware, resident software, micro-code, etc.) or an embodiment combining software and hardware aspects that may all generally be referred to herein as a "circuit," "module" or "system." Furthermore, the present invention may take the form of a computer program product embodied in one or more computer-readable (i.e., computer-usable) medium(s) having computer-usable program code embodied thereon .

[0023] Any combination of one or more computer-readable medium(s) may be utilized. The computer-readable medium include a computer-readable storage medium, which may be, for example, but is not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, a device, or any suitable combination of the foregoing. Exemplary computer readable storage medium may include, but is not limited to, a portable computer diskette, a hard disk, a solid state disc drive, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), an optical fiber, a portable compact disc read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer-readable storage medium may be any medium that can contain, or store a program for use by or in connection with an instruction execution system, apparatus, or device.

[0024] Computer program code for carrying out operations of the present invention may be written in an object oriented programming language such as Java, Smalltalk, C++ or the like. However, the computer program code for carrying out operations of the present invention may also be written in conventional procedural programming languages, such as the "C" programming language or similar programming languages. The program code may execute entirely on a single computing device, e.g., as a stand-alone software package, and or may be at least partly executed on multiple computing

devices that may be remote to one another. In the latter scenario, remote computing devices may be connected to one another through a local area network (LAN) or a wide area network (WAN), or the connection may be made to one or more remote computing devices (for example, through the Internet using an Internet Service Provider).

[0025] The present invention is described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0026] These computer program instructions may also be stored in a computer-readable memory that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable memory produce an article of manufacture including instruction means which implement the function/act specified in the flowchart and/or block diagram block or blocks.

[0027] The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide steps for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0028] Referring to FIG. 1, there is shown validity check process 10, repair process 12, and automated unpacking process 14 that may each reside on and may be executed by computing device 16. While each of validity check process 10, repair process 12, and automated unpacking process 14 are shown residing on computing device 12, this is intended for illustrative purposes only, as one or more of validity

check process 10, repair process 12, and automated unpacking process 14 may reside on a separate computing device.

[0029] Examples of computing device 16 may include, but are not limited to: a personal computer, a server computer, a series of server computers, a mini computer, and a mainframe computer. Computing device 16 may run an operating system, for example, Microsoft® Windows® XP, or Red Hat® Linux®, for example. Various additional / alternative computing devices and operating systems may equally be utilized. For example, computing device 16 may be part of a distributed computing network with one or more of validity check process 10, repair process 12, and automated unpacking process 14 being executed, in whole or in part, on another computing device coupled with computing device 16 via a data network (e.g., a LAN, a WAN, the Internet, etc.).

[0030] As will be discussed below in greater detail, validity check process 10 may parse a binary image of a portable executable file to generate a parsed field. Validity check process 10 may also determine an attribute of the parsed field. Further validity check process 10 may compare the attribute of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification. Validity check process 10 may also determine if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

[0031] Further, and as will also be discussed below in greater detail, repair process 12 may identify an invalid field of a portable executable file. Repair process 12 may also determine a likelihood of repairing the invalid field of the portable executable file. Repair process 12 may generate a repair model for repairing the invalid field of the portable executable file. Repair process 12 may repair the invalid field of the portable executable file is repaired based upon, at least in part, the repair model.

[0032] Similarly, and as will also be discussed below in greater detail, automated unpacking process 14 may set a debugging breakpoint at an original entry point address of a packed portable executable file. Automated unpacking process 14 may also execute a debugging process for the packed portable executable file to obtain a debugged portable executable file in memory. Automated unpacking process 14 may also collect one or more of import address table data and relocation table data during execution of the debugging process for the packed portable executable file. Automated unpacking process 14 may copy the debugged portable executable file in

memory to a storage medium, and may terminate the debugging process.

[0033] The instruction sets and subroutines of validity check process 10, repair process 12, and automated unpacking process 14, which may include one or more software modules, and which may be stored on storage device 18 coupled to computing device 16, may be executed by one or more processors (not shown) and one or more memory modules (not shown) incorporated into computing device 16. Storage device 18 may include but is not limited to: a hard disk drive; a solid state drive, a tape drive; an optical drive; a RAID array; a random access memory (RAM); and a read-only memory (ROM).

[0034] Due to the fact that PE (portable executable) files contain executable code, it may be desirable to perform file validation prior to execution of the binary object (e.g., the binary image of the PE file). Validity check process 10 may analyze a PE binary image prior to execution to determine if the PE file is a valid binary image. A valid binary image may refer to a file that can be used by a given operating system, either as an image that contains executable code or other type of multimedia information.

[0035] As discussed above, and referring also to FIG. 2, validity check process 10 may parse 50 a binary image of a portable executable file (e.g., PE binary image 20, residing on storage device 18, shown in FIG. 1) to generate 52 a parsed field. Validity check process 10 may also determine 54 an attribute of the parsed field. Further, validity check process 10 may compare 56 the attribute of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification. Validity check process 10 may determine 58 if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

[0036] Validity check process 10 may parse 50 PE binary image 20 to generate 52 a parsed field. Validity check process 10 may parse 50 PE binary image 20 to generate 52 a plurality of fields consistent with PE file format 100. For example, validity check process 10 may generally parse 50 PE binary image into a portable executable format signature, an ImageBase field, a SizeOfImage field, a FileAlignment field, a SectionAlignment field, an EntryPoint address, an import table, an import address table, an export table, a relocation table, a resource table, a thread local storage table,

a load configuration table, a bound import table, a COM table, and a portable executable section table.

[0037] While several fields have been indicated, these are intended only for illustrative purposes only, as validity check process 10 may parse 50 PE binary image 20 into various additional / alternative fields selected based upon design criteria and user need. Additionally, parsing 50 PE binary image 20 to generate 52 one or more parsed fields, may include, but is not limited to, physically isolating each field (e.g., copying each field into a separate file, database field, or the like), individually reading each field, associating an offset with the beginning (and/or ending) of each field, or the like. As such, parsing 50 PE binary image 20 to generate 52 one or more parsed fields may allow individual examination of each field.

[0038] As discussed above, validity check process 10 may also determine 54 an attribute of the parsed field. The attribute determined 54 by validity check process 10 may include one or more of a field identifier, a field length, and a field content. For example, validity check process 10 may determine 54 that PE binary image 20 includes an ImageBase field having a value of 0x00400000, a SectionAlignment field having a value of 0x1000, and a FileAlignment field having a value of 0x200.

[0039] Validity check process 10 may compare 56 the one or more determined 54 attributes of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification. A valid corresponding field may include a field that is required or allowed by the "Microsoft Portable Executable and Common Object File Format Specification" published by Microsoft Corporation (PECOFF), and which corresponds to a parsed field. For example, a valid corresponding field for the parsed ImageBase field may be the ImageBase field allowed as an option Windows specific field by PECOFF. A valid characteristic of a valid corresponding field may include a characteristic that is allowable by PECOFF. For example, PECOFF may specify acceptable field identifiers, field lengths and field contents of an accepted PE binary image. For example, PECOFF may define a default ImageBase value of 0x00400000, and may require that the value be a multiple of 64 K. Similarly, PECOFF may specify that the SectionAlignment field have a value that is greater than or equal to the FileAlignment. Further, PECOFF may specify that the FileAlignment have a value that is a power of

2 between 512 and 64 K, with a default value of 512. Accordingly, the foregoing may be example of valid characteristics for the identified fields.

[0040] Validity check process 10 may determine 58 if the attribute of the parsed field matches the valid characteristic of the valid corresponding field based upon, at least in part, the comparison 56 between the attribute of the parsed field and a valid characteristic of a valid corresponding field. Continuing with the above stated example, the determined 54 attribute for the FileAlignment field of PE binary image 20 was 512. As also discussed above, PECOFF may specify that the FileAlignment field have a value that is a power of 2 between 512 and 64 K. Accordingly, validity check process 10 may determine 58 that the attribute of parsed FileAlignment field (e.g., having a value of 512) matches a valid characteristic of a valid FileAlignment field.

[0041] Determining 58 if the attribute of the parsed field matches the valid characteristic of the valid corresponding field may include determining 60 if the attribute of the parsed field is valid for a predetermined operating system. Again, continuing with the above-stated example, validity check process 10 may have determined 54 a ImageBase field value of 0x00400000 for PE binary image 20. This determined value may be the default value for the operating systems Windows NT, Windows 2000, Windows XP, Windows 95, Windows 98, and Windows Me. However, the ImageBase field default value for Windows CE is 0x00010000, per PECOFF. Accordingly, validity check process 10 may determine 60 that the parsed ImageBase field for PE binary image 20 is not valid for Windows CE.

[0042] Validity check process 10 may determine 58 if the parsed field does not match the valid characteristic of the valid corresponding field. Continuing with the above-stated example, PECOFF specifies that the SectionAlignment field have a value that is greater than, or equal to, the FileAlignment. Further, validity check process 10 may have determined 54 a SectionAlignment field attribute of 256 and a FileAlignment field attribute of 512 for PE binary image 20. Accordingly, as the determined 54 SectionAligment field attribute (i.e., 256) is not greater than or equal to the determined 54 FileAlignment field attribute (i.e., 512) for PE binary image 20, validity check process 10 may determine 58 that the parsed SectionAlignment field does not match a valid characteristic of a valid corresponding field (i.e., the determined 54 SectionAlignment field attribute is not greater than or equal to the

determined 54 FileAlignment field attribute). Accordingly, validity check process 10 may provide an indicator (e.g., may provide an indicator in a graphical user interface, not shown).

[0043] If the parsed field does not match the valid characteristic of the valid corresponding field, validity check process 10 may determine 62 a likelihood of modifying the parsed field that does not match the valid characteristic of the valid corresponding field to generate a valid field. Validity check process 10 may determine 62 the likelihood of modifying the parsed field to generate a valid field based upon, at least in part, the number and nature of errors in a field that does not match the valid characteristic of a valid corresponding field. For example, and continuing with the above-discussed example, the parsed SectionAlignment field of PE binary image 20 does not match a valid characteristic of a valid SectionAlignment field because the value is less than the value of the FileAlignment field. Validity check process 10 may determine 62 a relatively strong likelihood of being able to modify the SectionAlignment filed of PE binary image 20 to generate a valid characteristic as the parsed SectionAlignment field of PE binary image 20 includes a single well defined error (namely, the value is less than the FileAlignment field). It may, for example, be possible to modify the SectionAlignment field to include a value that is greater than or equal to the FileAlignment field. Although some recursive testing may be necessary to modify the SectionAlignment field of PE binary image 20 to achieve a valid field it may be reasonably likely that such a modification may be achieved.

[0044] The likelihood of modifying a field to generate a valid field may be determined 62 based upon, at least in part, one or more empirically determined rules. The one or more empirically determined rules may be based upon, at least in part, various possible types of errors that may occur in various fields, and the possible modifications that may be implemented to correct the errors. As such, an error type in a given field for which there may be relatively few possible modifications that may generally result in a valid field, validity check process 10 may determine 62 a relatively high likelihood of modifying the field to generate a valid field. Conversely, for an error type having many possible modifications, many of which may not result in a valid field, validity check process 10 may determine 62 a relatively low likelihood of modifying the field to generate a valid field. Similarly, if the number of

detected errors between the parsed field and a valid corresponding field are relatively large, validity check process 10 also determine 62 a relatively low likelihood of modifying the field to generate a valid field.

[0045] Validity check process 10 may also determine 64 if the binary image of the portable executable file includes a dynamic link library, a kernel driver, or an executable object. Validity check process 10 may determine 64 the nature of PE binary image 20 based upon, for example, one or more of the included fields, the content of the included fields, or the like, by evaluating the parsed fields relative to possible valid characteristics and possible valid fields. Various additional / alternative characteristics of PE binary image 20 may similarly be determined. For example, validity check process 10 may determine one or more of an environment in which the PE file may execute, if the PE file is a console or other application with a graphical user interface, if the PE file includes dependencies and whether the dependencies exist on the target system, if the PE file includes depend functions and whether the dependent function exist in libraries available on the target system, etc.

[0046] As briefly mentioned above, validity check process 10 may provide an output indicating the various parsed fields, field attributes, validity of the fields, nature of the PE file, etc. In an embodiment, validity check process 10 may provide a graphical user interface, through which the various outputs may be rendered. Additionally / alternatively validity check process 10 may provide an output to a database, file, etc., which may be consumed by a user via an appropriate program, such as a database application. Various other suitable outputs will be appreciated by those having skill in the art.

[0047] PE binary images may be come damaged through various mechanisms. For example, PE files may become damaged when the files are transferred from one media to another. Similarly, errors may be introduced by software packers (e.g., UPX, PECompact, ASPack, etc.). Error introduced by software packers may render some files valid only for certain versions of operating systems that support PE file formats. Accordingly, repair process 12 may be implemented to repair damaged PE files.

[0048] With reference also to FIG. 3, repair process 12 may identify 100 an invalid field of a portable executable file (e.g., PE binary image 20 shown in FIG. 1). Further, repair process 12 may determine 102 a likelihood of repairing the invalid

field of the portable executable file. Repair process 12 may generate 104 a repair model for repairing the invalid field of the portable executable file. Repair process 12 may repair 106 the invalid field of the portable executable file based upon, at least in part, the repair model generated 104 by repair process 12.

[0049] Repair process 12 may identify 100 an invalid field of PE binary image 20 utilizing a variety of mechanisms. For example, repair process 12 may receive 108 an indicator from validity check process 10 (described in detail above) indicating the validity of PE binary image 20 and/or subset parts of PE binary image 20 (i.e., the validity of the various fields of PE binary image 20). Repair process 12 may receive 108 the indicator directly from validity check process 10. Additionally / alternatively, in an embodiment in which validity check process 10 may generate a validity report (e.g., in the form a file, database entries, or the like), repair process 12 may identify 100 an invalid field (and/or a plurality of invalid fields) by accessing 110 the validity report and interpreting the contents thereof.

[0050] In further embodiments, repair process 12 may identify 100 an invalid field of PE binary image 20 by performing 112 one or more validity checks on PE binary image 20. For example, repair process 12 may perform one or more validity checks on PE binary image 20 in a manner similar to that discussed above with reference to validity check process 10. For example, repair process 12 may generally parse PE binary image 20 into a plurality of fields, and may compare attributes of the plurality of fields to valid characteristics of valid corresponding fields. As discussed above, valid characteristics of valid corresponding fields may be specified by PECOFF. Accordingly, the validity of a field (and/or of PE binary image 20 as a whole) may be determined based upon, at least in part, whether the various fields and attributes comply with PECOFF. Therefore, repair process 12 may identify 100 an invalid field as a field having an attribute that does not comply with a valid characteristic of a valid corresponding fields as specified by PECOFF.

[0051] When identifying 100 an invalid field, repair process 12 may examine all fields of PE binary image 20, and/or may give special attention to the most crucial fields of PE binary image 20. Examples of fields that may be particularly important (e.g., which may have the greatest impact on the executability of PE binary image 20) may include, but are not limited to, PE format signatures, PE specific fields (e.g., ImageBase, SizeOfImage, FileAlignment, SectionAlignment, and EntryPoint

address), and PE specific tables (e.g., Import table, Import address table, Export table, Relocation table, Resource table, Thread local storage table, Load configuration table Bound import table, COM table, and PE section tables).

[0052] As discussed above, repair process 12 may determine 102 a likelihood of repairing the invalid field (or multiple invalid fields) of PE binary image 20. Repair process 12 may determine 102 a likelihood or repairing the invalid field of PE binary image 20 based upon, at least in part, determining 114 the number and characteristics of attributes of the invalid field that do not match a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification (e.g., PECOFF). For example, it will be appreciated that various errors may have a higher likelihood of being repairable than other errors. Similarly, a PE file having relatively few errors may have a higher likelihood of being repairable than a PE file having a relatively large number of errors.

[0053] Repair process 12 may determine 102 the likelihood of repairing an invalid field including comparing the identified 100 invalid field(s) (including the attributes of the invalid fields that fail to comply with PECOFF) against a library of possible errors and likelihood of repairing the error. The library (e.g., library 22 residing on storage device 18) may include empirically derived data of various errors that have previously been encountered and whether it was possible to repair the error to obtain an executable file.

[0054] As mentioned above, repair process 12 may generate 104 a repair model for repairing the identified 100 invalid field(s). The repair model generated 104 by repair process 12 may include one or more algorithms for repairing the one or more identified 100 invalid field. Similar to determining 102 a likelihood of repairing the invalid field, repair process 12 may generate 104 the repair model based upon, at least in part, one or more empirically derived rules (e.g., which may be included in library 22). For example, repair process 12 may generate 104 a repair model for repairing an invalid SizeOfImage field having an abnormal value, in which the rule may include recalculation of a correct SizeOfImage value. Similarly, repair process 12 may generate 104 a repair model for repairing an invalid entry point section that does not include an executable attribute, in which the rule may include correcting the section attributes. In a further example, repair process 12 may generate 104 a repair model for repairing an invalid resource table data that cannot be physically located, in which

the rule may include temporarily removing the invalid resource table values in the PE header. Additionally, library 22 may include rules that are empirically derived based upon a comparison between different operating system versions and the way the different operating system versions process the PE file format.

[0055] Repair process 12 may repair 106 the invalid field of the portable executable file based upon, at least in part, the repair model generated 104 by repair process 12. Some errors (i.e., invalid fields) may be repaired "on disk" by modifying PE binary image 20 residing on storage device 18. Accordingly, repair process 12 may repair 106 the invalid field may by statically repairing 116 the invalid field. Statically repairing 116 the invalid field may include modifying 118 the image of the portable executable file (e.g., PE binary image 20) on storage device 18. Repair process 12 may store 120 the modified PE image on storage device 18.

[0056] For example, and continuing with the above example, in which the PE field SizeOfImage was identified 100 as being invalid for having an abnormal value, repair process 12 may statically repair the SizeOfImage field of PE binary image 20. For example, repair process may recalculate a correct SizeOfImage value. Repair process 12 may modify PE binary image 20 to include the correct SizeOfImage value. Repair process 12 may store modified PE binary image 20 on storage device 18.

[0057] In addition to errors that may be repaired "on disk," other errors may be repaired in memory. The determination as to what errors may be repaired "on disk" on what errors may be repaired in memory may be based upon, at least in part, the empirically derived rules (e.g., which may reside in library 22). For errors that may be repaired in memory, repair process 12 may dynamically repair 122 the invalid field. To dynamically repair 122 an invalid field, repair process 12 may execute 124 the portable executable file (e.g., PE binary image 20). Repair process 12 may further modify 126 the portable executable file residing in memory (e.g., in RAM) during execution, in which the portable executable file residing in memory during execution is based upon the portable executable file (e.g., based upon PE binary image 20).

[0058] Further, repair process may repair 106 the invalid field by disabling 128 the invalid field. For example, repair process may temporarily disable 128 an invalid field by removing 130 the invalid field from an image of the portable executable file (e.g., PE binary image 20) stored on storage device 18 prior to execution of the portable executable file.

[0059] In some embodiments, repair process 12 may repair 106 an invalid field by disabling 128 the invalid field and dynamically repairing 122 the invalid field. For example, and referring to the above example in which resource table data could not be physically located, repair process 12 may temporarily remove 130 the invalid resource table values in the PE header. Repair process 12 may then execute 124 PE binary image 20 (e.g., repair process 12 may execute an unpacker of PE binary image 20) up to the original entry point of the portable executable file. The original entry point may include the first instruction of code of the portable executable file before the portable executable file was protected (e.g., packed). Once execution of PE binary image 20 reaches the original entry point the process memory may be dumped 132 to storage device 18. That is, the process memory associated with the execution of PE binary image 20 residing in RAM may be saved to storage device 18. The resource table data acquired from memory during unpacking of PE binary image 20 may be reverted to an original state, and a new PE file based upon, at least in part, the dumped process memory may be stored. Accordingly, a valid PE file (i.e., a PE file in compliance with PECOFF) may be achieved.

[0060] In an embodiment a PE binary image 24 may include a packed portable executable file. A packed portable executable file may include portable executable file (consistent with PECOFF, discussed herein above) that may include one or more software protections, such as compression, encryption, combinations of compression and encryption, etc. Automated unpacking process 14 may, generally, execute a debugging process for the packed portable executable file, and may utilize various breakpoints and callbacks to collect import address table filling data, as well as various other data that may be used to build an unprotected, valid portable executable file based upon packed (e.g., protected) PE binary image 24.

[0061] Referring also to FIG. 4, in general automated unpacking process 14 may set 150 a debugging breakpoint at an original entry point address of a packed portable executable file (e.g., packed PE binary image 24, shown in FIG. 1). Automated unpacking process 14 may also execute 152 a debugging process for the packed portable executable file to obtain a debugged portable executable file in memory (e.g., in RAM). Automated unpacking process 14 may collect 154 one or more of import address table data and relocation table data during the execution 152 of the debugging process for the packed portable executable file. Automated unpacking process 14

may copy 156 the debugged portable executable file stored in memory to a storage medium (e.g., storage device 18). Automated unpacking process 14 may terminate 158 the debugging process at the original entry point of the portable executable file.

[0062] As discussed, automated unpacking process 14 may set 150 a debugging breakpoint at an original entry point address of packed PE binary image 24. The original entry point of packed PE binary image 24 may be the first instruction of the executable code before the file was protected. Setting 150 a debugging breakpoint at the original entry point address of packed PE binary image 24 may allow the execution of packed PE binary image 24 to be suspended prior to control being passed to the executable file embodied within packed PE binary image 24. As used herein, "execution of packed PE binary image" and "executing packed PE binary image" may refer to the execution of the file embodied by the packed PE binary image and the executing PE file embodied by the packed PE binary image. Automated unpacking process 14 may determine 160 the ImageBase field data of the packed portable executable file and AddressOfEntryPoint data of the packed portable executable file. The ImageBase field data and the AddressOfEntryPoint data may be loaded from packed PE binary image 24. The original entry point address of packed PE binary image 24 may be the sum of the ImageBase data and the AddressOfEntryPoint data. Determining the original entry point may additionally include other numeric calculations, which may be based upon, at least in part, the software packer layout itself. Automated unpacking process 14 may load various additional data from packed PE binary image 24, such as, but not limited to, ImageBase data, SizeOfImage data, and PE section data.

[0063] Automated unpacking process 14 may initialize 162 the debugging process. Initializing 162 the debugging process may include creating a debugging process based upon, at least in part, packed PE binary image 24. That is, initializing 162 the debugging process may establish a debugging environment in which packed PE binary image 24 may be executed. In the initialized debugging process, automated unpacking process 14 may set 150 a debugging breakpoint on the original entry point. The debugging breakpoint set on the original entry point will be called once the debugged process finishes loading, before execution of the first instruction of the executable file embodied within packed PE binary image 24.

[0064] Automated unpacking process 14 may execute 152 the initialized debugging process. That is, packed PE binary image 24 may be executed within the established debugging environment. Automated unpacking process 14 may collect 154 one or more of import address table data and relocation table data. Collecting 154 one or more of import address table data and relocation table data may include running the debugging process until it reaches the import address table filling code. In part, automated unpacking process 14 may collect 154 one or more of import address table data and relocation table data by setting 164 one or more debugging breakpoints associated with a LoadLibrary call, a GetModuleHandle call, and a GetProcAddress call. Additional breakpoints may also be associated with a part of the software packer that relocates the file in memory. Breakpoints associated with a LoadLibrary call, a GetModuleHandle call, and a GetProcAddress call may be set, in some embodiments, during initialization 162 of the debugging process.

[0065] Packed PE binary image 24 executing within the debugging process may utilize a LoadLibrary API call or a GetModuleHandle API call in order to load a dependent dynamic link library. Setting 164 one or more breakpoints associated with a LoadLibrary call or a GetModuleHandle call may result in a callback to automated unpacking process 14 when executing packed PE binary image 24 loads a dynamic link library. In response to the breakpoint callback associated with a LoadLibrary call or a GetModuleHandle call, automated unpacking process 14 may collect 154 the name of the dynamic link library being loaded by executing packed PE binary image 24.

[0066] Similarly, packed PE binary image 24 executing within the debugging process may utilize a GetProcAddress API call to find the locations of necessary API's (application programming interfaces). Setting 164 one or more breakpoints associated with a GetProcAddress API call may result in a callback to automated unpacking process 14 when executing packed PE binary image 24 loads the addresses of necessary API's. In response to the breakpoint callback associated with a GetProcAddress API call, automated unpacking process 14 may collect 154 the API addresses being located by executing packed PE binary image 24. Executing packed PE binary image 24 may call GetProcAddress API at two locations, e.g., for string API locating and ordinal API locating. Automated unpacking process 14 may set 164 a breakpoint associated with each GetProcAddress API call. Automated unpacking

process 14 may add the locations of API's located by the GetProcAddress API calls to the last collected dynamic link library.

[0067] Automated unpacking process 14 may copy 156 a debugged PE file from memory (e.g., RAM) to a computer readable medium, such as storage device 18. Once packed PE binary image 24, executing within the debugging environment, reaches the original entry point of the executable file embodied therein, unpacking of the file may be substantially complete. That is, the file may be decompressed and/or decrypted, or the like (depending upon the nature of the protections associated with packed PE binary image 24). As such, at this point an unpacked PE file may reside in memory associated with computing device 16 (e.g., PE in memory 26, shown in FIG. 1). Automated unpacking process 14 may copy unpacked PE in memory 26, e.g., to a file residing on storage medium 18. As such, automated unpacking process 14 may create stored PE 28, which may be at least a portion of an unpacked portable executable file based upon, at least in part, packed PE binary image 24.

[0068] Automated unpacking process 14 may paste 166 one or more of an import address table, based upon, at least in part, the collected 154 import address table data, and a relocation table, based upon, at least in part, the collected 154 relocation table data into the debugged portable executable file (e.g., stored PE 28). For example, automated unpacking process 14 may construct one or more of an import address table and a relocation table based upon, at least in part, the import address table data and the relocation table data collected 154 by automated unpacking process 14 during execution 152 of the debugging process (e.g., which may include executing packed PE binary image 24 within a debugging environment).

[0069] Pasting 166 one or more of an import address table, based upon, at least in part, collected 154 import address table data, and a relocation table, based upon, at least in part, collected 154 relocation table data into the debugged portable executable file (e.g., stored PE 28) may include adding 168 a new section to the debugged portable executable file. For example, stored PE 28 may not include a section for an import address table and/or a section for a relocation table. Accordingly, automated unpacking process 14 may make space for an import address table and/or a relocation table within stored PE 28 (e.g., by adding 168 an appropriate section within stored PE 28 for an import address table and/or a relocation table). Automated unpacking

process 14 may then paste 166 the import address table and/or the relocation table into the appropriate locations of stored PE 28.

[0070] Once the import address table and/or the relocation table have been pasted 166 into stored PE 28, automated unpacking process 14 may realign 170 the debugged PE file (e.g., stored PE 28). Generally, realigning 170 the debugged PE file may include compacting the file and verifying that the file is a valid image, e.g., which may include verifying that the physical sizes of the individual PE sections of the file are correct and as small as possible. Additionally, automated unpacking process 14 may make all section attributes of the debugged PE file (e.g., stored PE 28) read, write, and execute. As such, automated unpacking process 14 may create a valid PE file that may substantially resemble packed PE binary image 24 prior to packing (i.e., prior to modifying the file with software protections and/or compression).

[0071] With the unpacking process complete, automated unpacking process 14 may terminate 158 debugging of packed PE binary image 25 at the original entry point.

[0072] While various discrete processes have been discussed herein above, such separate discussion is intended for ease of explanation. The various discrete processes (and / or portions thereof) may include modules of a larger application that may interoperate with one another. Additionally, the various features and steps of the processes may be utilized in combination with features and steps of other processes described herein. Accordingly, the present disclosure should not be construed as being limited to the discrete processes as described above.

[0073] A number of implementations have been described. Nevertheless, it will be understood that various modifications may be made. Accordingly, other implementations are within the scope of the following claims.

What is claimed is:

1. A computer implemented method comprising:

parsing, by a computing device, a binary image of a portable executable file to generate a parsed field;

determining, by the computing device, an attribute of the parsed field;

comparing, by the computing device, the attribute of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification; and

determining, by the computing device, if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

2. The computer implemented method of claim 1, wherein the parsed field includes one or more of a portable executable format signature, an ImageBase field, a SizeOfImage field, a FileAlignment field, a SectionAlignment field, an EntryPoint address, an import table, an import address table, an export table, a relocation table, a resource table, a thread local storage table, a load configuration table, a bound import table, a COM table, and a portable executable section table.

3. The computer implemented method of claim 1, wherein the attribute of the parsed field includes one or more of a field identifier, a field length, and a field content.

4. The computer implemented method of claim 1, wherein determining if the attribute of the parsed field matches the valid characteristic of the valid corresponding field further includes determining, by the computing device, if the attribute of the parsed field is valid for a predetermined operating system.

5. The computer implemented method of claim 1, further comprising identifying if the parsed field does not match the valid characteristic of the valid corresponding field.

6. The computer implemented method of claim 5, further comprising determining, by the computing device, a likelihood of modifying the parsed field that does not match the valid characteristic of the valid corresponding field to generate a valid field.

7. The computer implemented method of claim 1, further comprising determining, by the computing device, if the binary image of the portable executable file includes a dynamic link library, a kernel driver, or an executable object.

8. A computer program product comprising a computer readable medium having a plurality of instructions stored thereon, which, when executed by a processor, cause the processor to perform operations comprising:

parsing a binary image of a portable executable file to generate a parsed field;

determining an attribute of the parsed field;

comparing the attribute of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification; and

determining the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

9. The computer program product of claim 8, wherein the parsed field includes one or more of a portable executable format signature, an ImageBase field, a SizeOfImage field, a FileAlignment field, a SectionAlignment field, an EntryPoint address, an import table, an import address table, an export table, a relocation table, a resource table, a thread local storage table, a load configuration table, a bound import table, a COM table, and a portable executable section table.

10. The computer program product of claim 8, wherein the attribute of the parsed field includes one or more of a field identifier, a field length, and a field content.

11. The computer program product of claim 8, wherein determining if the attribute of the parsed field matches the valid characteristic of the valid corresponding field further includes determining if the attribute of the parsed field is valid for a predetermined operating system.

12. The computer program product of claim 8, further comprising identifying if the parsed field does not match the valid characteristic of the valid corresponding field.

13. The computer program product of claim 12, further comprising determining a likelihood of modifying the parsed field that does not match the valid characteristic of the valid corresponding field to generate a valid field.

14. The computer program product of claim 8, further comprising determining if the binary image of the portable executable file includes a dynamic link library, a kernel driver, or an executable object.

15. A system comprising:

a processor;

a memory coupled with the processor;

a first software module executable by the processor and the memory, the first software module configured to parse a binary image of a portable executable file to generate a parsed field;

a second software module executable the by processor and the memory, the second software module configured to determine an attribute of the parsed field;

a third software module executable by the processor and the memory, the third software module configured to compare the attribute of the parsed field to a valid characteristic of a valid corresponding field based upon, at least in part, a portable executable file format specification; and

a fourth software module executable by the processor and the memory, the fourth software module configured to determine if the attribute of the parsed field matches the valid characteristic of the valid corresponding field.

16. The system of claim 15, wherein the parsed field includes one or more of a portable executable format signature, an ImageBase field, a SizeOfImage field, a FileAlignment field, a SectionAlignment field, an EntryPoint address, an import table, an import address table, an export table, a relocation table, a resource table, a thread local storage table, a load configuration table, a bound import table, a COM table, and a portable executable section table.

17. The system of claim 15, wherein the attribute of the parsed field includes one or more of a field identifier, a field length, and a field content.

18. The system of claim 15, wherein the fourth software module, configured to determine if the attribute of the parsed field matches the valid characteristic of the valid corresponding field, is further configured to determine if the attribute of the parsed field is valid for a predetermined operating system.

19. The system of claim 15, further comprising a fifth software module executable by the processor and the memory, the fifth software module configured to identify if the parsed field does not match the valid characteristic of the valid corresponding field.

20. The system of claim 19, further comprising a sixth software module executable by the processor and the memory, the sixth software module configured to determine a likelihood of modifying the parsed field that does not match the valid characteristic of the valid corresponding field to generate a valid field.

21. The system of claim 15, further comprising a seventh software module executable by the processor and the memory, the seventh software module configured to determine if the binary image of the portable executable file includes a dynamic link library, a kernel driver, or an executable object.

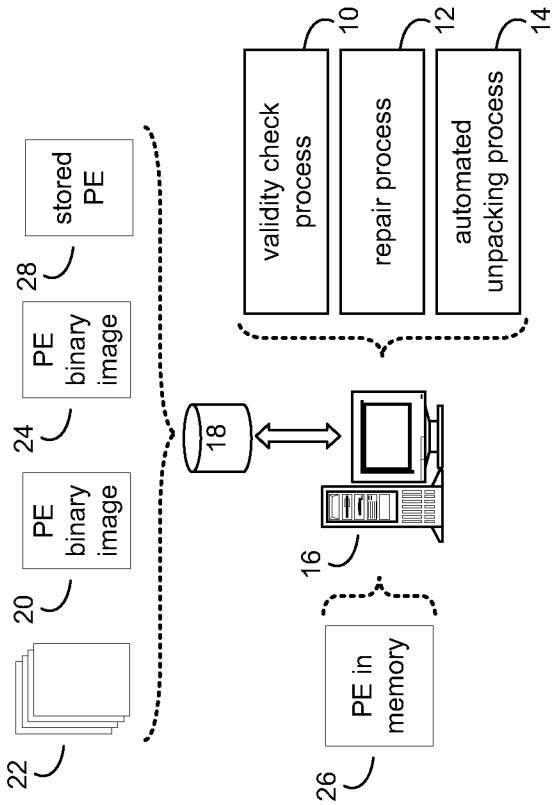
validity check process
10
repair process
12
automated unpacking process
14
18
16
PE in memory
26
22
20
PE binary image
24
PE binary image
28
stored PE

FIG. 1

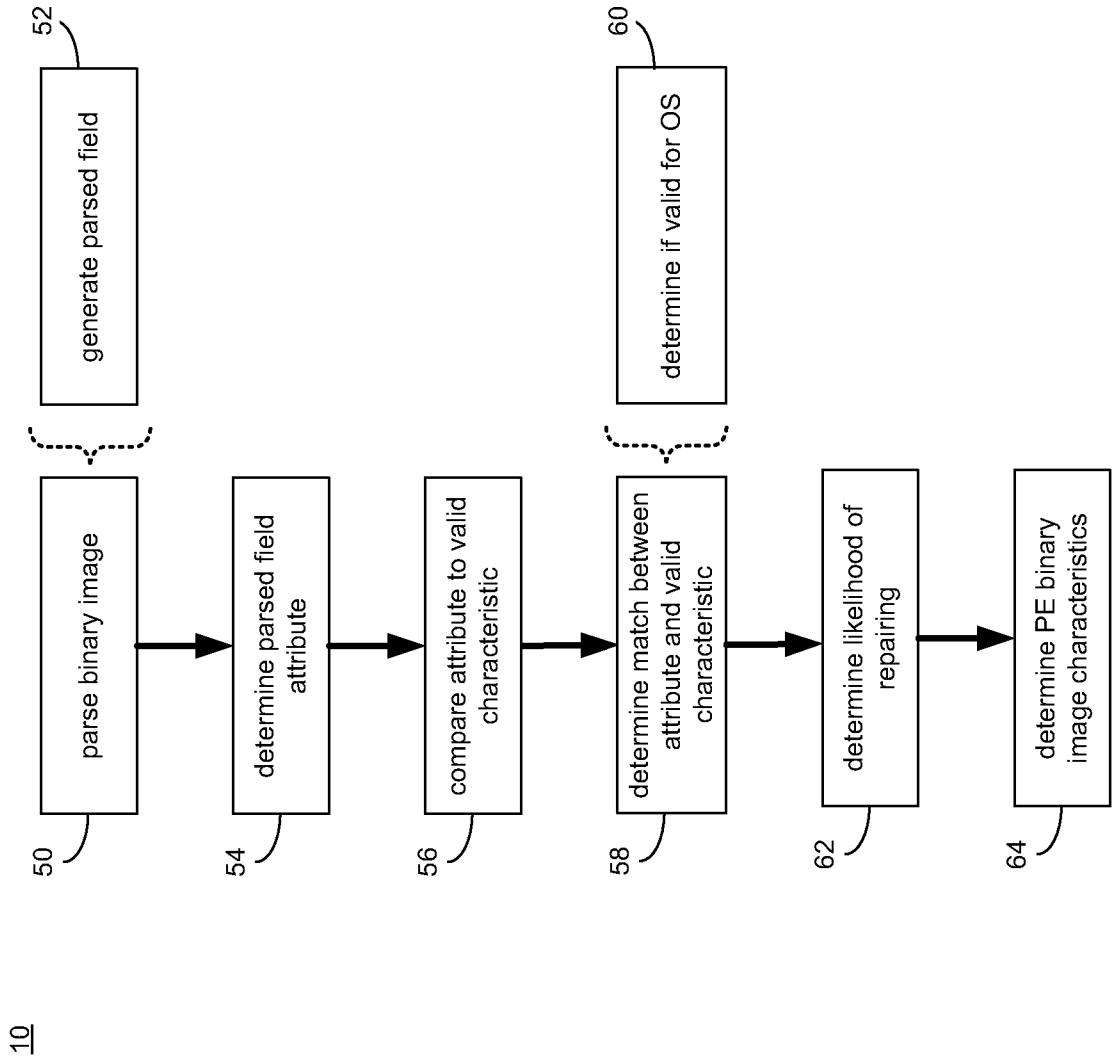
10
50
parse binary image
52
generate parsed field
54
determine parsed field attribute
56
compare attribute to valid characteristic
58
determine match between attribute and valid characteristic
60
determine if valid for OS
62
determine likelihood of repairing
64
determine PE binary image characteristics

FIG. 2

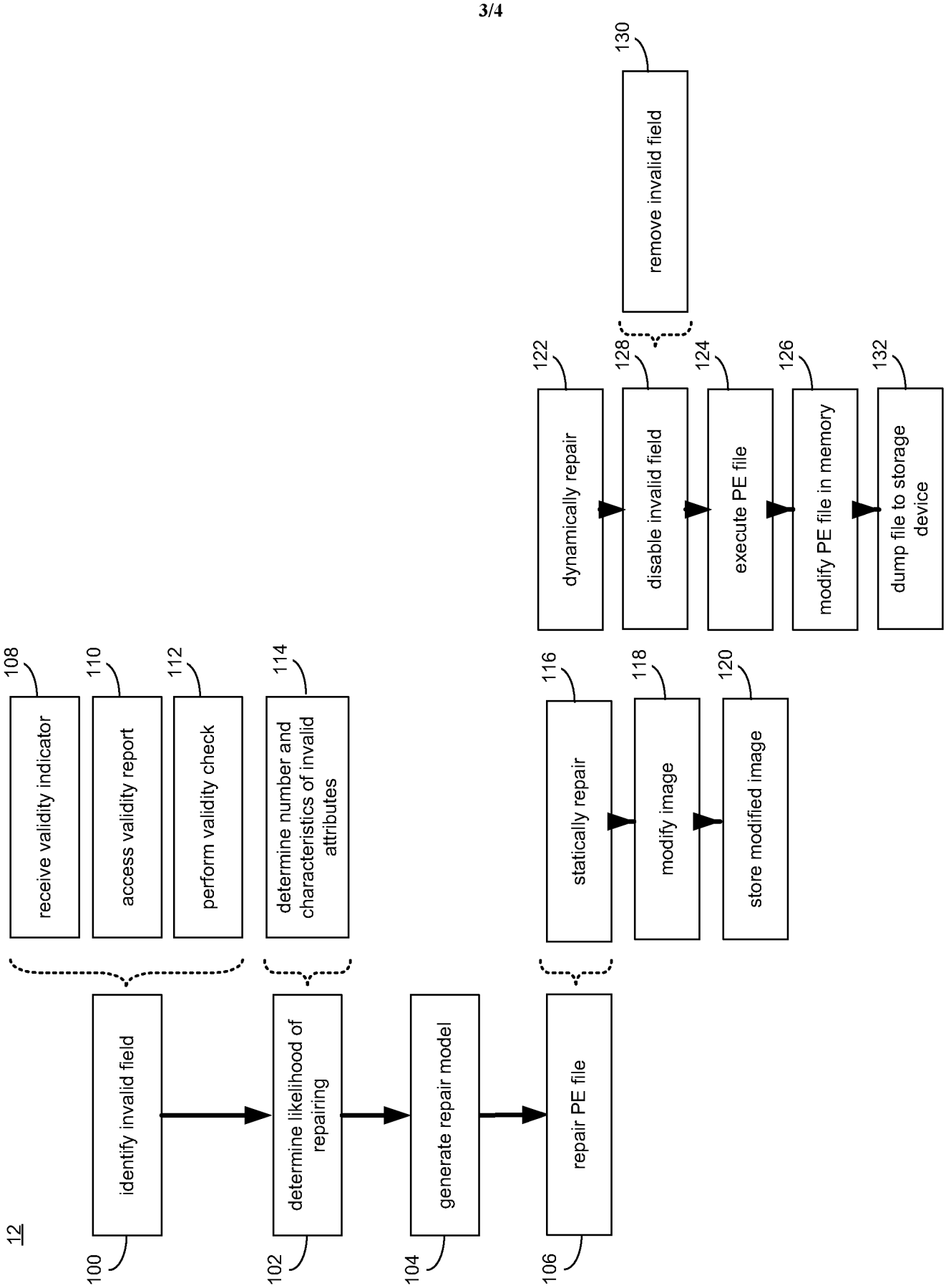
12
identify invalid field
100
receive validity indicator
108
access validity report
110
perform validity check
112
determine likelihood of repairing
102
determine number and characteristics of invalid attributes
114
generate repair model
104
repair PE file
106
statically repair
116
modify image
118
store modified image
120
dynamically repair
122
disable invalid field
128
remove invalid field
130
execute PE file
124
modify PE file in memory
126
dump file to storage device
132

FIG. 3

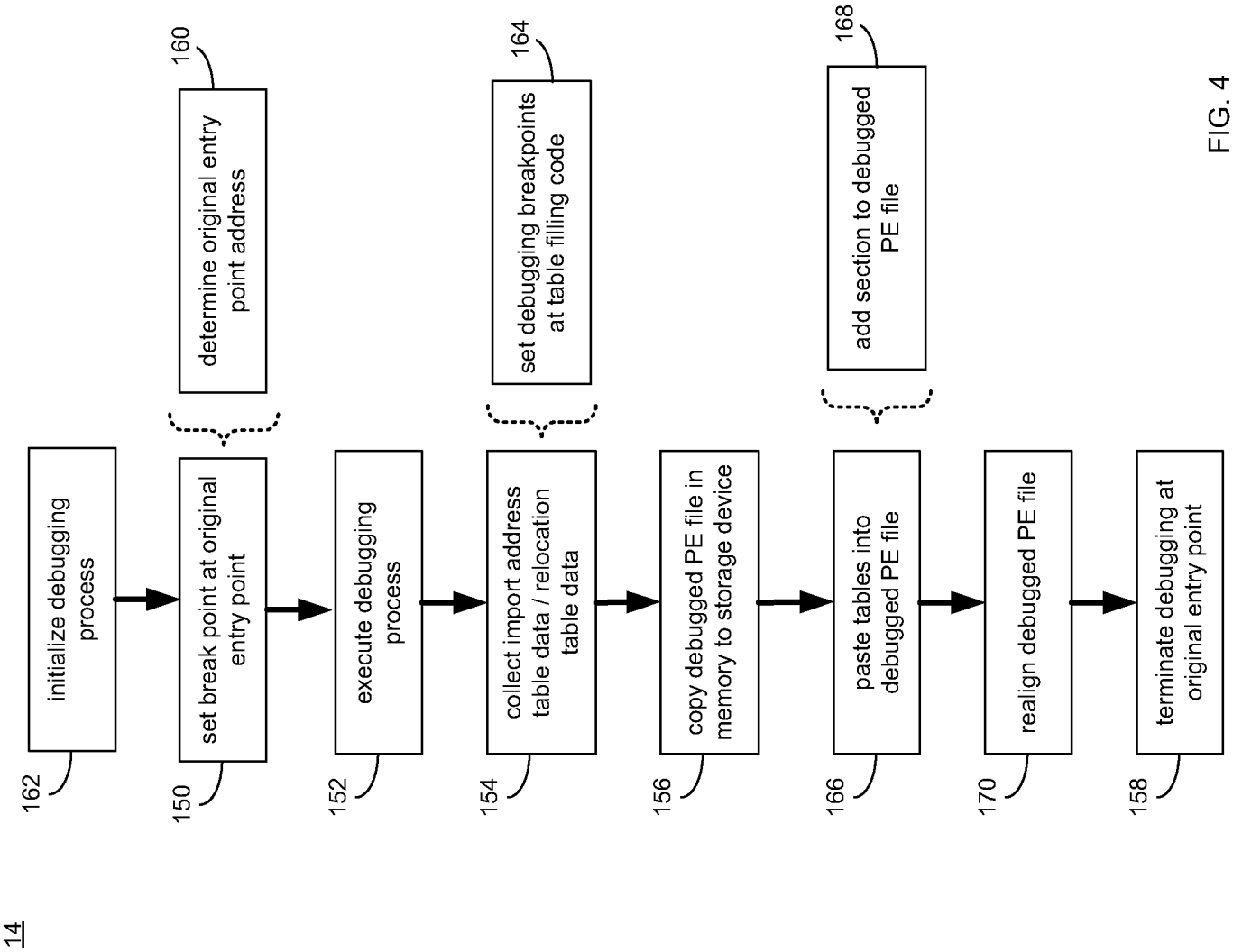
14
initialize debugging process
162
set break point at original entry point
150
determine original entry point address
160
execute debugging process
152
collect import address table data / relocation table data
154
set debugging breakpoints at table filling code
164
copy debugged PE file in memory to storage device
156
paste tables into debugged PE file
166
add section to debugged PE file
168
realign debugged PE file
170
terminate debugging at original entry point
158

FIG. 4

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2010/043660

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F11/07 G06F21/00
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2009/013405 A1 (SCHIPKA MAKSYM [GB]) 8 January 2009 (2009-01-08) * abstract; figures 2,4 paragraph [0051] - paragraph [0075] paragraph [0115] - paragraph [0130]	1-21
X	US 2009/133125 A1 (CHOI YANG SEO [KR] ET AL) 21 May 2009 (2009-05-21) * abstract; figures 2-5 paragraph [0015] paragraph [0077] - paragraph [0084] paragraph [0092] - paragraph [0098]	1-21

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
3 December 2010	13/12/2010

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Rousset, Antoine

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2010/043660

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	WEBER M ET AL: "A toolkit for detecting and analyzing malicious software", COMPUTER SECURITY APPLICATIONS CONFERENCE, 2002. PROCEEDINGS. 18TH ANN UAL 9-13 DEC. 2002, PISCATAWAY, NJ, USA,IEEE, 9 December 2002 (2002-12-09), pages 423-431, XP010627054, ISBN: 978-0-7695-1828-2 * abstract; figures 5,6 page 424, column 2, line 14 - page 425, column 2, line 10 page 428, column 1, line 21 - page 429, column 1, line 22	1-21
A	UNKNOWN: "Microsoft Portable Executable and Common Object File Format Specification", INTERNET CITATION, February 1999 (1999-02), XP002390296, Retrieved from the Internet: URL:http://n1rv4n4.iz4u.net/uploadfiles/PECOFF60.pdf [retrieved on 2006-07-13] the whole document	1-21

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/US2010/043660

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2009013405 A1	08-01-2009	WO 2009007686 A1	15-01-2009
US 2009133125 A1	21-05-2009	KR 20090052596 A	26-05-2009