METHOD AND APPARATUS FOR THE AUTOMATIC DETERMINATION OF POTENTIALLY WORM-LIKE BEHAVIOUR OF A PROGRAM

FIELD OF THE INVENTION

This invention relates generally to methods and apparatus for analyzing undesirable software entities, such as those known as worms, and relates more particularly to methods and apparatus for automatically identifying potentially worm-like behaviour in a software program.

BACKGROUND OF THE INVENTION

A Computer Virus can be defined as a self-replicating program or software routine that spreads on a computer in a possibly modified manner without human interaction. A Computer Worm can be defined as a program that can clandestinely send a copy of itself between computers on a computer network, and which uses a network service or services to replicate.

In the field of automated computer virus detection and analysis it is often necessary to predict what type of behaviour a program will exhibit so that the program can be replicated and analyzed in the environment most appropriate for the program.

Software can be dynamically analyzed to identify potentially-important behaviours (such as worm-like behaviour). Such behaviour may only be displayed when the software is executed in an environment where the software has, or appears to have, access to a production network and/or to the global Internet. The software may be executed in a real or in an emulated network environment that includes a monitoring component and an emulation component. The monitoring component serves to capture and/or record the behaviours displayed by the software and/or other components of the system, and the emulation component gives the software being analyzed the impression that it is executing with access to a production network and/or to the global Internet. The software being analyzed is effectively confined to the analysis network environment, and cannot in fact read information from, or alter any information on, any production network or the global Internet.

It would be desirable to provide a capability to specify the identity of computer worms outside of such an environment. While it may be possible to use such an environment for the replication of both

computer software viruses and worms, it may be inefficient, as the worm replication environment assumes the presence of a real or an emulated network that in practice can be expensive to implement.

Thus an ability to predict if a sample of software is a potential worm, outside of the network environment, would reduce the number of samples sent to the worm replication environment, and result in a significant improvement to the efficiency of automated replication and analysis systems.

SUMMARY OF THE INVENTION

The present invention accordingly provides methods and apparatus for the automatic determination of the behavioural profile of a program suspected of having worm-like characteristics. In a first aspect a method includes analyzing data processing system resources required by the program and, if the required resources are not indicative of the program having worm-like characteristics, running the program in a controlled non-network environment while monitoring and logging accesses to system resources to determine the behaviour of the program in the non-network environment. A logged record of the observed behaviour is analyzed to determine if the behaviour is indicative of the program having worm-like characteristics. The non-network environment may simulate the appearance of a network to the program, without emulating an actual operation of the network.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

Fig. 1 is a block diagram of a data processing system in a preferred embodiment of the present invention;

Fig. 2 is a block diagram of a controller unit in a preferred embodiment of the present invention;

Fig. 3 is a logic flow diagram that illustrates the operation of the resource analyzer component of Fig. 2 in a preferred embodiment of the present invention;

Figs. 4A and 4B, collectively referred to as Fig. 4, show a logic flow diagram that illustrates the operation of the replicator unit of Fig. 2 and a logic flow diagram that illustrates the operation of a behaviour pattern analyzer component of Fig. 2 in a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

The method disclosed herein is based on the characteristics of a worm program that enable it to spread to other computers. These characteristics include, but need not be limited to, one or more of the following: (a) using Application Program Interfaces (APIs) from dynamic link libraries (dlls) to send electronic mail to another machine; (b) the automating of common mail programs such as, but not limited to, Microsoft Outlook™ and Outlook Express™; (c) the use of address books, system registry and other system resources to determine the potential recipients of the worm and/or the location of mail programs; (d) overwriting or replacing network APIs in system libraries with worm code that, if executed, would cause the inclusion of the worm code in all mail sent from the machine as an attachment, or as a separate mail message; (e) attempting to access resources on remote drives; and (f) dropping programs that would run after the system is restarted and would cause the worm code to be sent to other machines using one of the previously described methods.

A profile of a worm is one exhibiting those characteristics that would indicate the use of one or more of the aforementioned methods. For example, a worm profile may include importing a send method from a network dll, or changing a network resource, or attempting to access a system registry for obtaining information that is descriptive of any installed electronic mail programs.

The determination of the behavioural profile of a suspected program (i.e., one suspected of having worm-like behaviour, characteristics or attributes) is performed in two stages.

During the first stage, the resources required by the program are determined. Examples of the resources indicative of a potential worm-like behaviour include, but need not be limited to: (a) the dynamic link libraries for the network access; (b) methods imported from these libraries that indicate possible attempts to send electronic mail; and

(c) dynamic link libraries and methods indicating the automation of existing mail programs such as, for example, OLE or DDE.

During the second stage the suspect program is run one or more times in a controlled non-network environment where accesses to system resources, possibly all accesses, are monitored and logged. To narrow down the results and to reduce the number of false positives, the non-network environment may be configured to appear to have some network capabilities. For example, if a suspected worm is attempting to access address books or check for the presence of electronic mail programs, the environment can provide the expected information in order to elicit a more specific worm-like response. In some cases, it may be advantageous to install an electronic mail program on the system in order to facilitate a positive response to the suspected program should it attempt to access such a program.

After the one or more program runs are completed, the behaviour profile of the program is created based on both the changes made to the system, such as, by example only, a modification of any network dll, and any attempts to access specific resources such as a registry key containing the location of the mail program or of an address book.

A preferred embodiment of this invention runs on one or more of the computer systems 100 shown in Fig. 1, which shows a block diagram of a typical computing system 100 where the preferred embodiment of this invention can be practiced. The computer system 100 includes a computer platform 102 having a hardware unit 103, and a software-analyzing program (SAP) 101, also referred to herein as a controller 101, that implements the methods disclosed below. The SAP 101 operates on the computer platform 102 and hardware unit 103. The hardware unit 103 typically includes one or more central processing units (CPUs) 104, a memory 105 that may include a random access memory (RAM), and an input/output (I/O) interface 106. Micro instruction code 107, for example a reduced instruction set, may also be included on the platform 102. Various peripheral components 130 may be connected to the computer platform 102. Typically provided peripheral components 130 include a display 109, an external data storage device (e.g. tape or disk) 110 where the data used by the preferred embodiment is stored, and where a worm resource profile 205 (see Fig. 2) resides, and a printing device 111. A link 112 may also be included to connect the system 100 to one or more other similar computer systems shown simply as the block 113. The link 112 is used to transmit digital information between the computers 100 and 113. The link

112 may also provide access to the global Internet 113a. An operating system (OS) 114 coordinates the operation of the various components of the computer system 100, and is also responsible for managing various objects and files, and for recording certain information regarding same, such as date and time last modified, file length, etc. Associated with the OS 114 in a conventional manner is a registry 114A, the use of which is discussed below, and system initialization files (SYS_INIT_FILES) and other files, such as dlls 114B. Lying above the OS 114 is a software tools layer 114A containing, for example, compilers, interpreters and other software tools. The interpreters, compilers and other tools in the layer 116 run above the operating system 114 and enable the execution of programs that use methods known to the art.

One suitable and non-limiting example of computer system 100 is the IBM IntelliStation™ (trademark of the International Business Machines Corporation). An example of a suitable CPU is a Pentium™ III processor (trademark of the Intel Corporation); examples of an operating systems are Microsoft Windows™ 2000 (trademark of Microsoft Corporation) and a Redhat build of GNU/Linux; examples of an interpreter and a compiler are a Perl interpreter and a C++ compiler. Those skilled in the art will realize that one could substitute other examples of computing systems, processors, operating systems and tools for those mentioned above.

The SAP or controller 101 operates in accordance with the teachings of this invention to determine possible worm-like behaviour resulting from the execution of a suspected malicious program or undesirable software entity, referred to herein generically as a sample 115.

A presently preferred, but non-limiting, embodiment of the SAP or controller 101 uses one or multiple computers for implementing a controller subsystem or unit 200 and a replicator subsystem or unit 201, both of which are shown in Fig. 2. A worm-like behaviour analyzer 202, which forms a part of the controller 200, uses some of the currently existing tools 114A in order to determine the dlls and the imports used by the suspected program or sample 115.

Fig. 2 is a detailed block diagram of a controller unit 200 and a replicator unit 201 in the preferred embodiment of this invention, and shows a worm-like behaviour analyzer 202 and the environment where the preferred embodiment is run. The environment includes several computing systems, one of which is a controller 200 and another one of which is a replicator 201.

It should be noted that while the unit 201 is referred to herein as a replicator, its primary function is not to replicate a worm, i.e., provide additional instance of the worm. Instead its primary purpose is to create a system environment, more precisely an emulated system environment, in which to exercise the sample 115, one or more times in one or more ways, so as to determine the behaviour of the sample 115 as it pertains to and induces changes in the state of the system environment, as well as to obtain a record or log of the activities of the sample 115 when operating in the (emulated) system environment. Any changes in the system state, and the log of the activities of the executed sample 115, are compared to system state changes and activities known to be associated with worm-like behaviour and, if a match occurs, the sample 115 is deemed to exhibit worm-like characteristics. At this point it may be desirable to attempt to replicate the suspected worm to gain further instances thereof for analysis and subsequent identification.

Exercising the sample 115 in a number of "ways" implies, for the purposes of this invention, running the sample program via different system APIs (e.g., system and/or CreateProcess) one or several times, and also exercising a GUI, if the program has a GUI. Exercising the sample 115 in a number of ways may also be accomplished by running the sample program, then restarting the system and running the program again. These techniques are not intended to be exhaustive of the "ways" in which the sample 115 can be exercised.

The worm-like behaviour analyzer 202 includes a resource analyzer 203, also referred to herein as a static analyzer or as a static determination unit, and a behaviour pattern analyzer 204, also referred to herein as a dynamic analyzer or as a dynamic determination unit. The behaviour pattern analyzer 202 uses tools 206 and 207 that determine a list of dynamic link libraries 114B required by the sample 115 and the methods imported from the dynamic link libraries 114B, respectively. An example of a tool 206, 207 that can be used to determine which dynamic link libraries 114B are required by a program is known as Microsoft DEPENDS.EXE, described in an article "Under the Hood" by Matt Pietrek in the Feb. 1997 issue of the Microsoft Systems Journal and available through the Microsoft Developer Network. An example of a tool 206, 207 that can be used to determine the imports of the sample program 215 is known as DUMPBIN.EXE, which forms a part of the Microsoft Developer Studio™ version 6.0. Other tools for performing the same or similar functions can be found or written by those skilled in the art.

The resource analyzer 203 (static determination) uses these tools to create a profile of the resources used by the suspected program or sample 115, and compares the results to the content of the worm resource profile 205. A typical worm resource profile 205 may include the network dlls 114B such as, as non-limiting examples, WSOCK32.DLL, INETMIB1.DLL, WINSOCK.DLL, MSWSOCK.DLL, WININET.DLL, MAPI32.DLL, MAPI.DLL and WS2_32.DLL, as well as a dll indicating use of OLE automation such as, for example, OLE32.DLL, as well as the list of methods imported from these dynamic link libraries. These imported methods can include, but need not be limited to, the "send", "sendto" and WSAsend methods imported from WSOCK32.DLL, the CoCreateInstance and CoCreateInstanceEx methods from OLE32.DLL, or the DDEConnect method from USER32.DLL.

The behaviour pattern analyzer 204 (dynamic determination) creates the behaviour profile of a suspected program using the results of the run of the sample 115 on the replicator 201, and compares these results to a worm behaviour profile 208. A typical worm behaviour profile 208 includes a list of changes to the system and/or attempts at file and registry access that are indicative of worm-like behaviour. The worm behaviour profile 208 list may include, but need not be limited to, the following elements: (a) changes to one or more network dlls, but no non-network dlls; (b) creation of one or more files with the VBS extension; (c) creation of any new files and corresponding changes to/creation of the system initialization files 114B that would cause a replacement of any of the network dlls with a new file. An example of this latter scenario is the creation of the file wininit.ini in the Windows directory on a Windows system, where the created wininit.ini file contains instructions such as "WSOCK32.DLL=SOME.FILE", where SOME.FILE is a new file created by the program. The worm behaviour profile 208 list may further include (d) a record of attempts to access address books or/and registry keys corresponding to the location of an electronic mail program .

Both the worm resource profile 205 and the worm behaviour profile 208 are preferably static in nature, and are preferably created before the sample 115 is run on the replicator 201.

The replicator 201 is invoked by the controller 200 prior to the behaviour pattern analysis operation discussed above. The replicator 201 includes a replication controller (RC) 209, behaviour monitors 210 and an optional network behaviour simulator 211.

The network behaviour simulator 211 operates in conjunction with the behaviour monitors 210 to create an appearance of a network-like behaviour to elicit certain worm-like behaviours of the sample 115. For example, the network behaviour simulator 211 operates to provide a false network address, such as a false IP address to the sample 215 when the behaviour monitor 210 detects a request for an IP address by the sample 115. In this case the sample 115 may request the local IP address to ascertain if the system has network capabilities before displaying worm-like behaviour, and a local IP address may be provided to the sample 115 as an inducement for the sample 115 to display worm-like behaviour.

In a similar fashion, the environment in which the sample 115 is run can be made to exhibit the presence of system resources and/or objects that do not, in fact, exist. For example, the sample 115 may request information about a specific file, and it may then be advantageous for the environment to respond with the requested information as if the file exists, or to create the file before returning it to the sample 115 as an inducement for the sample to display worm-like behaviour. That is, a known non-network environment can be made to exhibit at least one of non-existent local network-related resources and local network-related objects to the program.

Figs. 3 and 4 illustrate the overall control flow during the execution of the controller 200 and the replicator 201. In Fig. 3 the sample 115 is first delivered to the controller 200 at step 301, which then passes the sample 115 to the resource analyzer 203 at step 302. The resource analyzer 203 determines which dynamic link libraries 114B are accessed by the sample 115 at step 303, and compares the accessed dlls to those in the worm resource profile 205 (step 304).

If the dll usage matches the worm resource profile 205, the methods imported from these dlls are determined at step 305. If these methods match those contained in the worm resource profile 205 (step 306), the sample 115 is classified as a potential worm. If neither the dll usage or the imported methods match the worm resource profile 205, as indicated by a No indication at either of steps 304 and 306, the sample 115 is passed to the replicator 201, shown in Fig 4A, for replication and the subsequent behaviour pattern determination.

Fig. 4 illustrates the flow of control through the replicator 201 (Fig 4A) and the behaviour pattern analyzer 204 (Fig 4B). After the replication environment is initialized at step 401, the sample is sent to

the replicator 201 at step 402 and is executed at step 403. Control then passes to the behaviour pattern analyzer 204 (Fig 4B) that examines any changes to the system at step 404, and compares detected changes to those in the worm behaviour profile 208 at step 405. If there is a match the sample 115 is declared to be a potential worm, otherwise the behaviour pattern analyzer 204 analyzes the activity reported by the behaviour monitors 210 at step 406 and attempts at step 407 to match the reported sample behaviour to the activity patterns (worm-like behaviour patterns) listed in the worm behaviour profile 208.

If either the changes to the system analyzed at step 404 or the activity reported by the behaviour monitors at step 406 contains any of the patterns listed in the worm behaviour profile 208, the sample is classified as a potential worm, otherwise the sample 115 is classified as not being a worm. If classified as a worm at step 407, the sample 115A can be provided to a worm replication and analysis system for further characterization.

The method described above can be embodied on a computer readable medium, such as the disk 110, for implementing a method for the automatic determination of the behavioural profile of the sample program 115 suspected of having worm-like characteristics. The execution of the computer program causes the computer 100, 200 to analyze computer system resources required by the sample program 115 and, if the required resources are not indicative of the sample program 115 having worm-like characteristics, further execution of the computer program causes the computer 100, 200, 201 to run the program in a controlled non-network environment while monitoring and logging accesses to system resources to determine the behaviour of the program in the non-network environment. The further execution of the computer program may cause the computer to simulate the appearance of a network to the sample program, without emulating the operation of the network.

Note that foregoing description implies that only if the system resource requirements of the sample do not indicate a potential worm (static determination), that the replicator 201 is operated (dynamic determination). In the presently preferred embodiment this is the case, as the static determination process is typically much less computationally expensive than the dynamic determination process, and will typically execute much faster. However, it is also possible to execute both processes or subsystems when a potential worm is indicated by the first, with the second process or subsystem serving to verify the result of the

first. It should be further noted that in some cases it may not be desirable to require that both the static and dynamic determination processes reach the same conclusion as to the worm-like nature of a particular sample 115, as the generation of false negative results might occur. It is thus presently preferred that one of the static or dynamic processes is relied on to declare a particular sample to be worm-like, enabling the system to switch to processing the sample as a potential virus if worm-like behaviour is not indicated.

As was noted above, the sample 115 can typically be processed faster for the static determination executed by resource analyzer 203 (outside of the emulated environment) than for the dynamic determination executed by behaviour pattern analyzer 204.

CLAIMS

1. A method for the automatic determination of potentially worm-like behaviour of a program, comprising:

determining a behavioural profile of the program in an environment that does not emulate the operation of a network;

comparing the determined behavioural profile against a profile indicative of worm-like behaviour; and

providing an indication of potentially worm-like behaviour based on the result of the comparison.

2. A method as in claim 1, where the behavioural profile is determined by determining what system resources are required by the program.

3. A method as in claim 1, where the behavioural profile is determined by determining what system resources are referred to by the program.

4. A method as in claim 1, where the determination of the behavioural profile comprises:

executing the program in at least one known non-network environment,

using an automated method for examining the environment and determining what changes, if any, have occurred in the environment; and

recording any determined changes as said behavioural profile.

5. A method as in claim 4, where the known non-network environment has an ability to appear to exhibit network-related capabilities.

6. A method as in claim 5 where, in response to the program seeking to determine if the environment has network capabilities, providing a network address to the program as an inducement for the program to display worm-like behaviour.

7. A method as in claim 4, where the known non-network environment exhibits at least one of non-existent local network-related resources and local network-related objects to the program.

8. A method as in claim 4 where, in response to the program seeking to determine information about a file, responding as if the file exists as an inducement for the program to display worm-like behaviour.

9. A method as in claim 4 where, in response to the program seeking to determine information about a file, creating the file before returning the file to the program as an inducement for the program to display worm-like behaviour.

10. A method as in claim 4 where, in response to the program seeking to determine information about an electronic mail program, returning the information to the program as an inducement for the program to display worm-like behaviour.

11. A method as in claim 4 where, in response to the program seeking to determine information about an electronic mail address book, returning the information to the program as an inducement for the program to display worm-like behaviour.

12. A method as in claim 1, where determining the behavioural profile of the program includes exercising the program in a plurality of ways.

13. A method as in claim 1, where determining the behavioural profile of the program comprises determining a dynamic link library usage of the program.

14. A method as in claim 1, where determining the behavioural profile of the program comprises determining the identities of methods imported by the program from a library.

15. A method as in claim 1, where determining the behavioural profile of the program comprises determining what resources are requested by the program.

16. A method as in claim 15, where said resources comprise at least one of dynamic link libraries for network access, methods imported from dynamic link libraries that indicate a possible attempt to send electronic mail, and dynamic link libraries and methods indicating an automation of an electronic mail program.

17. A method as in claim 1, where determining the behavioural profile of the program comprises determining if system resources requested by the program match certain predetermined system resources and, if so, executing the program in a controlled environment, determining if system changes made during execution of the program match certain predetermined

system changes and, if so, declaring the program to be a possible worm, and if not, determining if activities reported by a program behaviour monitor match certain predetermined activities and, if so, declaring the program to be a possible worm.

18. A data processing system comprising at least one computer for executing a stored program for making an automatic determination of potentially worm-like behaviour of a program, comprising:

means for determining a behavioural profile of the program in an environment that does not emulate the operation of a network;

means for comparing the determined behavioural profile against a stored profile indicative of worm-like behaviour; and

means for providing an indication of potentially worm-like behaviour based on the result of the comparison.

19. A computer program product comprising instructions which, when executed on a data processing system having a non-volatile memory storage device, causes said system to carry out a method as claimed in any of claims 1 to 17.