

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

AO KASPERSKY LAB
Petitioner
v.

OPEN TEXT INC.
Patent Owner

IPR2024-00292
U.S. Patent No. 8,201,243

**DECLARATION OF MICHAEL J. DONAHOO, UNDER 37 C.F.R. § 1.68 IN
SUPPORT OF PETITION FOR *INTER PARTES* REVIEW**

TABLE OF CONTENTS

I.	INTRODUCTION	4
II.	QUALIFICATIONS AND PROFESSIONAL EXPERIENCE	5
III.	LEVEL OF ORDINARY SKILL IN THE ART	10
IV.	RELEVANT LEGAL STANDARDS	11
A.	Claim Construction	12
B.	Anticipation Under 35 U.S.C. § 102	14
C.	Obviousness Under 35 U.S.C. § 103	14
V.	OVERVIEW OF THE ‘243 PATENT	17
A.	Summary of the ‘243 Patent	17
B.	Overview of Prosecution History	19
C.	Claim Construction of ‘243 Patent	20
VI.	Technical Background Known to a Person of Ordinary Skill in the Art	20
A.	Computer Activity	20
B.	API Calls	21
C.	Kernel-Mode Driver	25
D.	Object	28
E.	Source/Externally Networked Source	29
F.	Heuristics and Heuristic Analysis	29
G.	Logging/Logs	31
VII.	OVERVIEW OF THE PRIOR ART	33
A.	OVERVIEW OF U.S. PATENT NO. 8,719,924 TO WILLIAMSON (“WILLIAMSON”)	33
B.	OVERVIEW OF U.S. PATENT APPLICATION PUBLICATION NO. 2007/0016951 TO PICCARD (“PICCARD”)	36
C.	OVERVIEW OF U.S. PATENT NO. 7,934,103 TO KIDRON (“KIDRON”)	39
VIII.	IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE	41

A.	Ground 1: 1-7 and 9-14 are obvious under 35 U.S.C. § 103 over Williamson in view of Piccard	41
1.	Reasons to Combine Williamson and Piccard	41
2.	Claim 1	49
3.	Claim 2	73
4.	Claim 3	75
5.	Claim 4	76
6.	Claim 5	78
7.	Claim 6	80
8.	Claim 7	86
9.	Claim 9	88
10.	Claim 10	89
11.	Claim 11	90
12.	Claim 12	90
13.	Claim 13	92
14.	Claim 14	93
B.	Ground 2: Claims 1-6 and 8-14 are obvious under 35 U.S.C. § 103 over Kidron in view of Williamson and Piccard	93
1.	Reasons to Combine Kidron and Williamson	93
2.	Reasons to Combine Kidron, Williamson, and Piccard	96
3.	Claim 1	98
4.	Claim 3	103
5.	Claim 6	104
6.	Claim 8	108
7.	Claim 12	109
8.	Claims 2, 4-5, 9-11, 13-14	110
IX.	NO OBJECTIVE INDICIA OF NONOBVIOUSNESS	110
X.	CONCLUSION	110

I, Michael J. Donahoo, do hereby state and declare as follows:

I. INTRODUCTION

1. I have been retained by DeWitt LLP, counsel for Petitioner, AO Kaspersky Lab in the matter of the *Inter Partes* Review of U.S. Patent No. 8,201,243 (“the ‘243 patent”) to Boney. My compensation is not contingent on the outcome of this matter or the specifics of my testimony.

2. I am also being reimbursed for reasonable and customary expenses associated with my work and testimony in this investigation. My compensation is not contingent on the outcome of this matter or the specifics of my testimony, and I have no other interest in this case or the parties thereto.

3. I have been asked to provide my opinions regarding whether claims 1-14 (the “Challenged Claims”) of the ‘243 patent are unpatentable as they would have been anticipated or obvious to a person of ordinary skill in the art (“POSITA”) at the time of the alleged invention, in light of prior art. It is my opinion that all limitations for the Challenged Claims would have been anticipated or obvious to a POSITA in light of the prior art. My opinions are based on my experience, knowledge and skill in computer security and cyber-security technology.

4. In preparation for this declaration, I have reviewed the ‘243 patent, its prosecution history, and all of the exhibits cited in this declaration and Petition. I

may rely upon these materials, my experience and knowledge, and/or additional materials to rebut arguments raised by the Patent Owner. Further, I may also consider additional documents and information in forming any necessary opinions, including documents that may not yet have been provided to me.

5. I am over the age of 18 and make this Declaration based upon personal knowledge, experience, and investigation that I have conducted. I will provide testimony concerning my statements, opinions and conclusions set forth below and may amplify or add thereto through such testimony, demonstratives, or in further written declarations.

II. QUALIFICATIONS AND PROFESSIONAL EXPERIENCE

6. My name is Michael J. Donahoo. I currently live in Waco, Texas.

7. I received a Bachelor of Science in Computer Science from Baylor University in 1991. In 1993, I received a Master of Science in Computer Science from Baylor University. I completed a Ph.D. in Computing from The Georgia Institute of Technology in 1998 with a research focus in networking.

8. From 1998 to the present, I have worked for Baylor University in the Department of Computer Science, where I am currently a professor of computer science.

9. Soon after joining Baylor, I developed completely new courses in networking at the undergraduate (senior-level) and graduate level. The courses include

instruction in network system and protocol security. I also teach a senior-level, undergraduate class on Operating Systems, which includes content that I developed on system security.

10. In Spring 2016, I developed and taught a completely new course, Introduction to Cybersecurity, cross listed for both senior-level undergraduate and graduate. This course covers the concepts and tools of cybersecurity and includes hands-on labs/exercises.

11. In Fall 2018, I developed and taught a completely new Cybersecurity Lab course for undergraduates. This course provides weekly laboratory experience in tools and defensive/offensive techniques in cybersecurity.

12. Since 2016, I have been the coach of the Baylor cybersecurity team. The team participates in a range of competitions, including Collegiate Cyber Defense Competition (CCDC) and National Cyber League (NCL). These competitions focus on the tools, technologies, and concepts in cybersecurity offense, defense, and policy. I developed materials and executed training for student contestants.

13. I am the creator and technical editor of The Practical Guide Series, published by Morgan-Kaufmann, which consists of focused, relatively fast-paced tutorial/reference books on various computing technologies. Books from this series are used in classrooms around the world. I have co-authored four books in the series, two of which are in their second edition. Three of my books focus on networking

and TCP/IP socket programming in different programming languages. In addition to basic sockets, these books provide instruction on message serialization, which details the issues and solutions involved in expressing messages as a serialized sequence of octets. These books also explore how TCP/IP subsystems operate within the operating system. Several of my books have been translated into other languages, including Japanese, Korean, Chinese, and Portuguese.

14. Since 2001, I have worked as a volunteer for the International Collegiate Programming Contest (ICPC), and I currently serve as Deputy Executive Director for the global organization. I also serve as the Chief Information Officer (CIO) for the ICPC. The ICPC is a multitier, team-based, programming competition operating under the auspices of the ICPC Foundation. Started over 50 years ago, the contest involves a global network of universities hosting regional competitions that advance teams to the ICPC World Finals, which are held around the world, including locations such as Vancouver, Honolulu, Shanghai, Prague, and, most recently, Bangladesh. Participation has grown to tens of thousands of the finest students and faculty in computing disciplines at over 3,500 universities from over 100 countries on six continents. The contest fosters creativity, teamwork, and innovation in building new software programs and enables students to test their ability to perform under pressure.

15. As ICPC CIO, I lead an international team of software developers working on enterprise cloud development to support a range of contest operations. The services are deployed in cloud (AWS). My team and I developed, deployed, and managed a cybersecurity solution, including permission models, firewalling, virtual private cloud networks, anomaly monitoring, etc.

16. Since 2018, I have served as part of the leadership in developing Baylor cybersecurity at the university level, including creation of an industry advisory board, development/submission of NSA Center of Academic Excellence (CAE) designation application, etc.

17. My research focuses on cybersecurity and networking. See my attached CV for a full listing of my writings and presentations, which is attached to this report as Exhibit 1003.

18. Further details of my qualifications are presented in my curriculum vitae, which is attached to this report as Exhibit 1003. By my education, training, and experience, as evidenced by my curriculum vitae, I am qualified as one skilled in the art of computer security and cyber-security technology.

19. I considered information from various sources in forming my opinions. Besides drawing from my experience in the computer industry, I also reviewed the '243 patent and its file history, the other documents and references as cited herein, and general technical references. The exhibits are:

- A. Ex. 1001: U.S. Patent No. 8,201,243 (issued June 12, 2012).
- B. Ex. 1002: Declaration of Michael J. Donahoo, Ph.D. under 37 C.F.R. §1.68.
- C. Ex. 1003: Curriculum Vitae of Michael J. Donahoo, Ph.D.
- D. Ex. 1004: File History of U.S. Patent No. 8,201,243.
- E. Ex. 1005: U.S. Patent No. 8,719,924 to Williamson et al. (filed March 3, 2006) (“Williamson”).
- F. Ex. 1006: U.S. Patent Publication No. 2007/0016951 to Piccard et al. (filed July 13, 2005) (“Piccard”).
- G. Ex. 1007: U.S. Patent No. 7,934,103 to Kidron (“Kidron”).
- H. Ex. 1008: WALTER ONEY, PROGRAMMING THE MICROSOFT WINDOWS DRIVER MODEL, (2nd ed. 2003).
- I. Ex. 1009: Ted Hudek & Don Marshall, *KMDF Version History*, MICROSOFT (Oct. 31, 2023), <https://learn.microsoft.com/en-us/windows-hardware/drivers/wdf/kmdf-version-history>.
- J. Ex. 1010: TROY MOTT, LEARNING CARBON, (Apple Computer, Inc. eds., 1st ed. 2001).
- K. Ex. 1011: JAMES GOSLING, FRANK YELLIN, & THE JAVA TEAM, *The Java™ Application Programming Interface, Volume 1: Core Packages in THE JAVA SERIES . . . FROM THE SOURCE*.

- L. Ex. 1012: DICTIONARY OF COMPUTING, (4th ed. 1996).
- M. Ex. 1014: Claim Construction Order, Webroot, Inc., Open Text Inc. v. AO Kaspersky Lab, 6:22-cv-00243-ADA-DTG (U.S. Dist. Ct. W. Dist. Tx. Waco Div., July 25, 2023).
- N. Ex. 1016: Plaintiffs' Responsive Claim Construction Brief Regarding Patents Plaintiffs Added by Amendment [263], Open Text Inc. v. AO Kaspersky Lab, 6:22-cv-00243-ADA-DTG (U.S. Dist. Ct. W. Dist. Tx. Waco Div., May 5, 2023).
- O. Ex. 1017: JAMES BONEY, CISCO IOS IN A NUTSHELL (2001).
- P. Ex. 1018: DAVID HUCABY, CISCO ASA AND PIX FIREWALL LOGGING, provided by Cisco Press (2005)
<https://www.ciscopress.com/articles/article.asp?p=424447&seqNum=2>.
- Q. Ex. 1019: Jennifer M. Anderson, et al., Continuous Profiling: Where Have All the Cycles Gone?, 15 ACM Transactions on Computer Systems 357 (1997).

III. LEVEL OF ORDINARY SKILL IN THE ART

20. I understand the level of a person of ordinary skill in the art at the time of the claimed invention is determined through consideration of several factors. I understand that those factors include: (1) the educational level of the inventor(s); (2)

type of problems encountered in the art; (3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of active workers in the field.

21. After considering the factors mentioned above, and based on the disclosure of the '243 patent, it is my opinion that a person of ordinary skill in the art at the relevant time for these patents would hold a bachelor's degree in computer science or a similar field, have about 2 years of experience as a software programmer or similar title, and have about 2 years of experience specifically in the computer security field. A person having more education, such as a master's degree, would require less experience, and a person not having at least a bachelor's degree would need more experience.

22. Based on my education, training, and professional experience in the field of the claimed invention, I am familiar with the skill level and abilities of a person of ordinary skill in the art at the time of the claimed invention. Additionally, I met at least these minimum qualifications to be a person having ordinary skill in the art as of the time of the claimed invention of the '243 Patent.

IV. RELEVANT LEGAL STANDARDS

23. In forming my opinions and considering the subject matter of the '243 patent and its claims in light of the prior art, I rely on certain legal principles that counsel in this case explained to me. I summarize my understanding of these concepts below.

24. I understand that the claims define the invention. I also understand that an unpatentability analysis is a two-step process. First, the claims of the patent are construed to determine their meaning and scope. Second, after the claims are construed, the content of the prior art is compared to the construed claims.

25. I understand that a claimed invention is only patentable when it is new, useful, and non-obvious in light of the “prior art.” That is, the invention, as defined by the claims of the patent, must not be anticipated by or rendered obvious by the prior art.

A. Claim Construction

26. I understand that patent claims are generally interpreted according to their ordinary and customary meaning to a person of ordinary skill in the art taking into consideration the so-called “intrinsic evidence” of the patent consisting of (1) the claim language; (2) the specification and drawings; and (3) the prosecution history. I understand that a Court has discretion to take into consideration so-called “extrinsic evidence” including references (prior art and non-prior art) as well as definitions from dictionaries and treatises to the extent they do not contradict intrinsic evidence.

27. I understand that claim terms may be explicitly defined in the patent specification, or they may be implicitly defined through consistent usage in the specification. I also understand that the scope of claim terms may be limited by statements in the specification or prosecution history where the application clearly disavows or disclaims subject matter in a clear and unmistakable manner.

28. I understand that a patent is required to provide a written description of the invention, and the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, to make and use the claimed subject matter.

29. I understand that a patent claim is indefinite if, when read in light of the specification and the prosecution history, the claim fails to inform, with reasonable certainty, those skilled in the art about the scope of the invention.

30. It is my understanding that, although the terms of a claim may appear to be definite, inconsistency with the specification disclosure or prior art teachings may make an otherwise definite claim take on an unreasonable degree of uncertainty. It is also my understanding that indefiniteness may result from inconsistent prosecution history statements where the claim language and specification on their own leave an uncertainty that, if unresolved, would produce indefiniteness.

31. It is also my understanding that a fundamental rule of claim construction is that terms in a patent are construed with the meaning with which they are presented in the patent, and that if the specification reveals a special definition given to a claim term by the patentee that differs from the meaning it would otherwise possess, then the patentee's definition governs.

B. Anticipation Under 35 U.S.C. § 102

32. I understand that a patent claim is invalid as anticipated if a single piece of prior art teaches every element of the claims, viewed from the perspective of a person of ordinary skill in the art. I also understand that an anticipatory reference does not have to recite word for word what is in the anticipated claims. Anticipation can also occur when a claimed limitation is inherent in the relevant reference. I have been advised that if the prior art necessarily functions in accordance with, or includes, the claims limitations, it can anticipate even though the limitation is not expressly disclosed.

C. Obviousness Under 35 U.S.C. § 103

33. I understand that a patent claim is invalid as obvious if the claimed invention would have been obvious to a person of ordinary skill in the art at the time the claimed invention was made. This means that even if all of the elements of the claim cannot be found in a single prior art reference that would anticipate the claim, a person of ordinary skill in the field who knew about all the prior art would have come up with the claimed invention. I understand that in an obviousness determination, the person of ordinary skill in the art is presumed to have knowledge of all material prior art. I understand that whether a claim is obvious is based upon the determination of several factual issues. A patent is invalid as obvious when a POSITA (1) would have been motivated to combine the teachings of the prior art to

achieve and the claimed invention and (2) would have had a reasonable expectation of successfully producing the claimed invention through the combination.

34. In considering obviousness, I understand that one must determine the scope and content of the prior art. I understand that, in order to be considered as prior art to a patent being considered, a prior art reference must be reasonably related to the claimed invention of that patent. A reference is reasonably related if it is in the same field as the claimed invention or is from another field to which a person of ordinary skill in the art would look to solve a known problem.

35. I understand that one must determine what differences, if any, existed between the claimed invention and the prior art.

36. I understand that a patent claim composed of several elements is not proved obvious merely by demonstrating that each of its elements was independently known in the prior art. In evaluating whether such a claim would have been obvious, one may consider whether a reason has been identified that would have prompted a person of ordinary skill in the art to combine the elements or concepts from the prior art in the same way as in the claimed invention. There is no single way to define the line between true inventiveness on the one hand (which is patentable) and the application of common sense and ordinary skill to solve a problem on the other hand (which is not patentable). For example, market forces or other design incentives may be what produced a change, rather than true inventiveness.

37. I understand that one may consider whether (1) the change was merely the predictable result of using prior art elements according to their known functions, or whether it was the result of true inventiveness; (2) there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; (3) the innovation applies a known technique that had been used to improve a similar device or method in a similar way; (4) the claimed invention would have been obvious to try, meaning that the claimed innovation was one of a relatively small number of possible approaches to the problem with a reasonable expectation of success by those skilled in the art; (5) the invention merely substituted one known element for another known element in order to obtain predictable results; (6) the invention merely applies a known technique to a known device, method, or product to yield predictable results; or (7) known work in the field may have prompted variations of use of the same inventions in the same or different fields due to market forces or design incentives that would have been predictable to a person of ordinary skill in the art.

38. I understand that one must be careful not to determine obviousness using the benefit of hindsight; many true inventions might seem obvious after the fact. I understand that the determination should be based on the position of a person of ordinary skill in the field at the time the claimed invention was made and should not consider what is known today or what is learned from the teaching of the patent.

39. I understand that so-called “secondary considerations of non-obviousness,” must be considered in an obviousness analysis. I understand that an analysis cannot use hindsight in determining whether a patent claim is obvious. I understand that secondary considerations of non-obviousness include: (a) a long-felt but unresolved need for the invention; (b) commercial success of the invention; (c) copying of the invention; (d) praise and recognition of the invention by others; (e) licensing of the rights to the invention; and (f) unexpected results.

V. OVERVIEW OF THE ‘243 PATENT

40. The ‘243 patent is entitled, “Backwards Researching Activity Indicative of Pestware.” Ex. 1001, Cover. I understand that the ‘243 patent was filed on April 20, 2006, as application number 11/408,146 and does not claim priority or benefit to any prior filed application. For the purpose of my analysis, I have assumed the effective filing date is the filing date of application, April 20, 2006.

A. Summary of the ‘243 Patent

41. The ‘243 patent relates to researching an identity of a source of activity that is indicative of pestware. Ex. 1001, Abstract.

42. The ‘243 patent discloses protecting against “trojans, spyware, and adware,” collectively referred to as “malware,” “spyware” or “pestware.” Ex. 1001, 1:43-45. In general, its systems and methods operate as depicted in FIG. 4 below.

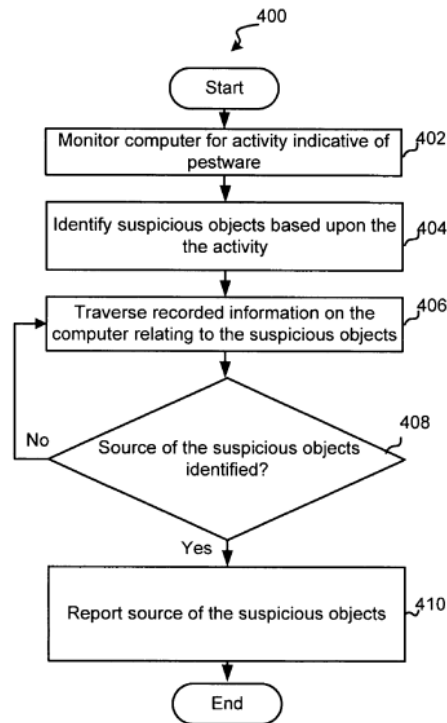


FIGURE 4

Ex. 1001, FIG. 4.

43. Exemplary claim 1 recites these method steps as monitoring, using a kernel-mode driver, the computer for API call activity that is indicative of pestware, and storing a history of at least a portion of the API calls in an activity log; analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold; identifying an object residing on the computer that is a suspected pestware object; accessing recorded history of externally networked sources that the computer received files from to identify a particular externally networked source that the suspected pestware object originated

from; and reporting the identity to an externally networked pestware research entity to research whether the particular externally networked source is a source of pestware. Ex. 1001, claim 1.

B. Overview of Prosecution History

44. It is my understanding that during the prosecution of the ‘243 patent there were a number of office actions rejecting the claims and a number of amendments were made to the claims.

45. In particular, in response to a non-final Office Action of February 25, 2010, Applicant amended exemplary independent claim 1 to recite “externally networked” source(s). Ex. 1004, pp. 166-167.

46. In response to a non-final Office Action of May 9, 2011, Applicant amended claim 1 to add “reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.” Ex. 1004, p. 70.

47. In a Notice of Allowability of February 9, 2012, the Examiner amended claim 1 with Applicant’s authorization, including “monitoring, using a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the monitoring includes monitoring API calls and storing a history of at least a portion of the API calls in an activity log; analyzing, heuristically, computer activity to determine

whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware;” and “accessing, in response to the identifying an object.” Ex. 1004, p. 47.

48. It is my opinion that Williamson and Piccard and Kidron, Williamson, and Piccard expressly disclose the allegedly patentable limitations cited above.

C. Claim Construction of ‘243 Patent

49. I understand that in the parallel litigation in the Western District of Texas the court construed contested terms in the ’243 patent under their plain and ordinary meaning with the exception of “pestware.” Specifically, I understand the Court defined “pestware” as “a trojan, spyware, adware, or malware program that collects information about a person or an organization.” Ex. 1014, p. 3. I have been informed to apply plain and ordinary meaning as proposed by Webroot and adopted by the Western District of Texas court for terms other than “pestware” and the court’s construction of “pestware.”

VI. Technical Background Known to a Person of Ordinary Skill in the Art

A. Computer Activity

50. Computer activity refers to a behavior, action, or event carried out by a computer. More specifically, the behaviors, actions, and events are performed by, for example, a process. The Oxford Dictionary of Computing defines a process as

“a stream of activity.” Ex. 1012, p. 378. Therefore, each step (behavior, action, event) performed by a process would be a computer activity.

51. When a user of a computer invokes a program (e.g., Google maps on their phone or Microsoft Word on their laptop), a “process” is launched by the operating system on which the program is running. For instance, if the user is running Google Maps and Facebook on their phone, then two processes would be launched, one for each of the invocations of each program. The software code associated with an invocation of an application is executed in a process.

52. The instructions for the software code associated with the invocation of the application are executed in a “process.” A person of skill in the art would understand the execution of the instructions to be computer activity.

53. One of skill in the art would understand computer activities include a large range of behaviors, actions, and events such as requesting file creation, requesting a read or write to/from the network, requesting resources or data from the operating system or another application, etc. Typically, such activities on a computer are made with API calls (or are considered API calls).

B. API Calls

54. An API call is a request for services or data through an API. API stands for application programming interface. An API is an interface that is defined in terms of a set of functions, and enables a program to gain access to facilities. Ex. 1012, p.

17. To call is “to transfer control to a []subroutine or []procedure, with provision for return to the instruction following the call at the end of execution of the subroutine/procedure.” Ex. 1012, p. 58. Therefore, an API call allows a process (instance of an application) to request data, resources, or services from, for example, another process (instance of an application), the operating system, or a library.

55. An interface is a known definition of how to request a service and receive the results. We encounter interfaces in everyday life. For example, if I want to put gas in my car, I go to a gas station (access), pull up to a gas pump (name), select a grade of gasoline (parameter), and receive gasoline refined to a particular standard (response).

56. Similarly, programmers carry out their tasks by utilizing the services of libraries and systems. Programmers access these services through a published interface generally referred to as an application programming interface. An application programming interface typically contains the service access instructions, names, parameters, and response specification. Consider a service for getting the current time. The interface might define the access (e.g., URL and protocol), name (e.g., getTime), parameter (e.g., timezone), and response format (e.g., Apr 2 08:30:37 CDT 2023). APIs are used by developers to program an application.

57. Accordingly, an API is essentially a set of functions that can be “called” to access a service. For example, Carbon was an API developed by Apple to enable smooth evolution of applications to newer versions of the Mac OS.

58. In Learning Carbon by Apple Computer Inc API is defined as “the set of functions and data structures defined by one piece of software, such as an operating system service, for use by client software, such as applications and device drivers. For example, you would access one programming interface to enable your application to print and another to manipulate your application’s menus.” Ex. 1010, p. 1. Likewise, the inventors of Java wrote a two-volume series on APIs for writing Java applications stating that “[t]he Java Application Programming Interface describes the libraries that every programmer can use to write Java applications and applets.” Ex. 1011, Back Cover.

59. To invoke a service or function of the API, a developer “calls” the service through the API. Therefore, an API call is the actual invocation of service via the API interface.

60. As indicated above, the activities performed on a computer are often API calls requesting data, system resources, and execution of system/application functions. Such API calls can be monitored by a kernel-mode driver as the driver can receive OS notifications of system calls or even be directly involved in the API calls/system

calls. As activities may be carried out by API calls and API calls can be monitored by a kernel-mode driver, a kernel-mode driver can monitor computer activities.

61. To carry out activities on a computing device, computational resources on a computing device (e.g., memory, CPU, disk space) are shared by the operating system (OS) and a set of users. For example, memory is divided into chunks that are allocated to different processes associated with different applications and different users, all operating on the same computer.

62. Part of the OS' job is to isolate the different processes and activities such that they cannot interfere with one another. Such interference might include Process A performing the activity of accessing or even modifying the memory for Task B. To prevent such interference, processes typically execute in a restricted mode (often called "user mode"). Processes operating in user mode cannot perform operations that would allow the task to interfere with the resources of another process.

63. If a process needs to perform activities prohibited in user mode, the OS may act on behalf of the process to carry out privileged activities. To do so, the OS operates in a less restricted mode, called "kernel mode." In kernel mode the OS can execute restricted activities that cannot be executed by processes in user mode. The various modes (i.e., user mode and kernel mode) can be likened to the permission given to a store cashier and manager. The cashier can perform some operations (e.g., sales) but not others (e.g., refunds). When the privileged operation is needed, the

cashier summons the manager who permits the privileged operation by entering a code, swiping a card, etc.

64. Activities of a process requesting services and data (function execution) that are performed all within “user mode” are typically referred to as API calls. Activities of a process for services and data (function execution) that requires kernel mode are referred to as system calls that are initiated in user mode as API calls.

C. Kernel-Mode Driver

65. A driver is a software component that implements the OS’ standardized, programmatic interface for a particular subsystem. Drivers can be configured to operate in either user-mode and/or kernel-mode. A kernel-mode driver, as the name indicates, is a type of driver capable of operating with kernel-mode access with less restrictive memory and instruction execution access than a user-mode driver. Kernel-mode drivers allow the OS to utilize a standardized interface to interact with a variety of subsystems with a permission model that allows the OS to carry out its responsibilities.

66. As indicated above, computational resources on a computing device (e.g., memory, CPU, disk space) are shared by the operating system (OS) and a set of users. Part of the OS’ job is to isolate the different processes such that they cannot interfere with one another. To prevent such interference, processes typically execute in user mode. Processes operating in user mode cannot perform activities (i.e., API

calls) that would allow the process to interfere with the resources of another process. For activities (i.e., API calls) not allowed in user mode, the OS may act on behalf of the process in kernel-mode to carry out the activity.

67. The OS needs to operate over a variety of systems and devices (subsystems), each of which has its own way of operating and communicating. Requiring the OS to understand how to communicate with all possible subsystems would be unwieldy; consequently, OSs define standard programmatic interfaces for talking to classes of subsystems (e.g., hard drive).

68. A driver allows a particular subsystem to interact with the OS' standardized, programmatic interface. In effect, a driver acts as a translator between the standardized actions of the operating system ("OS") and the specialized commands of the specific subsystem. For example, a driver might translate the OS print commands into the specific printer control language for an HP printer.

69. As indicated above, a kernel-mode driver is a driver capable of operating in kernel-mode. Kernel-mode drivers allow the OS to utilize a standardized interface to interact with a variety of subsystems with a permission model that allows the OS to carry out its responsibilities.

70. The Windows XP operating system, for example, uses kernel-mode drivers for a range of functions as shown below in Figure 1-4:

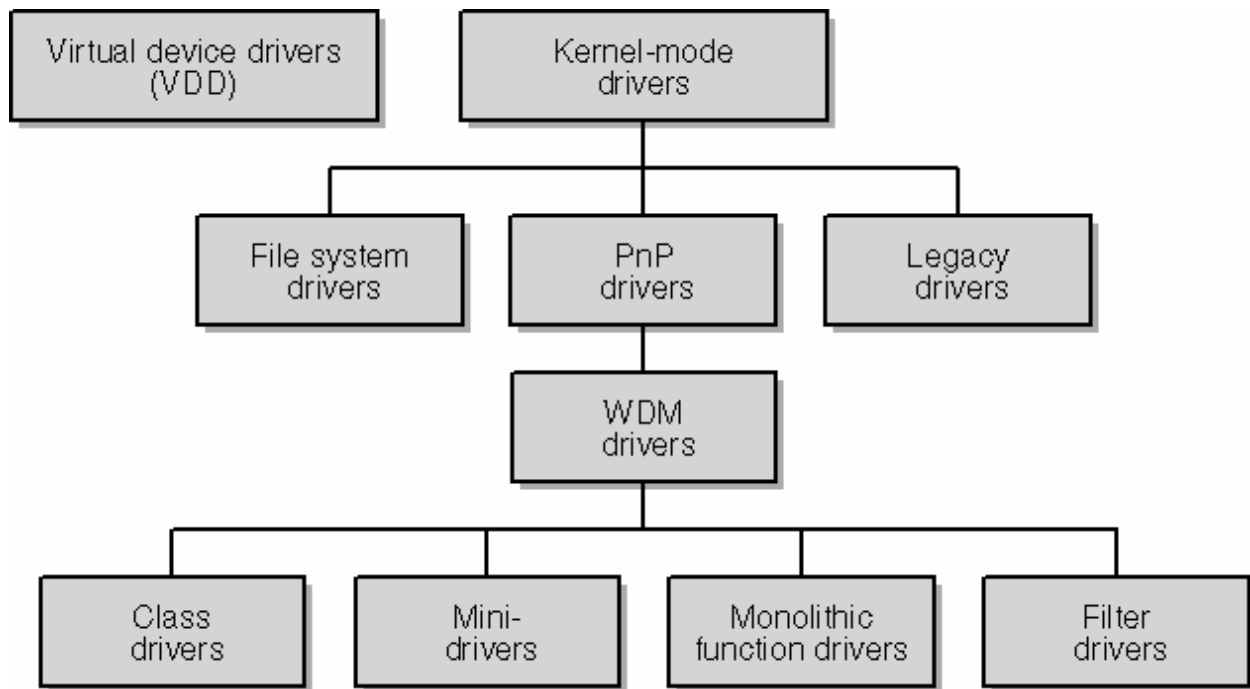


Figure 1-4. *Types of device drivers in Windows XP.*

Ex. 1008, FIG. 1-4.

71. As Oney further describes, “[f]ile system drivers implement the standard PC file system model (which includes the concepts of a hierarchical directory structure containing named files) on local hard disks or over network connections. These, too, are kernel-mode drivers.” Ex. 1008, p. 6 (emphasis omitted). In this instance, the kernel-mode driver provides a standardized interface for the PC file system model that is local or across the network. The OS operates over a standard interface for different hard disk models or even for storage across the network.

72. In fact, kernel-mode drivers are so critical that Microsoft developed the Kernel-Mode Driver Framework (KMDF) for use in Windows 2000 and versions of Windows XP. *See* Ex. 1009.

73. Accordingly, one of skill in the art would understand a kernel-mode driver to be a particular type of driver that communicates with the OS so that the OS can utilize a standardized interface to interact with a variety of subsystems. One of skill in the art would understand a kernel-mode driver to operate with less restrictions than a user-mode driver. One of skill in the art would appreciate that by monitoring the calls made to the interface of a kernel-mode driver, one could observe the activities (i.e., API calls) taken by processes executing on a computer, including actions taken by pestware.

D. Object

74. In computing, an object is some component containing data that resides in storage (e.g., memory, disk). For example, an executable is a file object that is stored on a computer's hard drive, where some of the data is a set of instructions to execute. When an executable file (object) is executed, a new object called a process may be created containing the execution state and some of the instructions. While waiting to execute the process is loaded into memory.

75. Objects can contain instructions for carrying out computer activities. For example, an executable file includes instructions to execute when invoked. One

skilled in the art would understand that some objects can be a component for carrying out computer activities, such as an executable file or a process. Pestware, for example, could be expressed as an object that carries out unwanted computer activities such as data collection.

E. Source/Externally Networked Source

76. Objects (i.e., executable files, scripts, etc..) for execution on a computer need to originate from some source (i.e., a website, an email, a CD-ROM, file stored on the computer hard drive, etc.).

77. One skilled in the art would understand an externally networked source is a source that is outside (i.e., external) of but accessible by the computing device. A common example of this is a web source (web page) where your browser is acquiring data (i.e., executable pestware file) over a network (i.e., Internet). Whereas, executing a file stored on the internal hard drive of the computing device would not be an externally networked source. Of course, one skilled in the art would appreciate that a file stored in the hard drive may have come from an externally networked source originally.

F. Heuristics and Heuristic Analysis

78. A heuristic is a problem-solving strategy or method with no guarantee of optimal performance. As defined by the Oxford Dictionary of Computing, a heuristic is “‘a rule of thumb,’ based on []domain knowledge from a particular application,

that gives guidance in the solution of a problem.” Ex. 1012, p. 224. We are often looking for ways to accomplish a task that quickly produces a good result. When looking for a rental on AirBNB, we may decide to only consider rentals with at least 4.7 stars and a minimum of 100 reviews. This does not guarantee the optimal experience; however, it does allow for rapid decision making with limited information. Such a decision-making guideline is a heuristic.

79. In algorithms, we can use heuristics to make a decision, given incomplete information. For example, if a user submits “hurricanes” to a search engine and there is an active hurricane named Norma near the user, the search engine may present news about Norma and information about storms. Alternatively, if a football team named the Hurricanes just won the championship, the search engine may present news about the team and information about football. Here, the search engine would be using a heuristic that if a search term is ambiguous or not specific enough, select the response most closely matching recent news.

80. If there are many factors in a decision (e.g., news and user’ previous searches), our heuristic may need to determine how much each factor influences the decision. The measure of influence for a factor is called the “weight.”

81. Accordingly, a heuristic analysis is the process of solving (analyzing) a problem based on a set of one or more heuristics (rules).

82. Distinguishing between pestware and non-pestware objects based on computer activities carried out by an object can be difficult, as both gather and process data. As such, we can look for patterns in activities indicative of pestware. An object carrying out such suspected activities is a suspected pestware object.

83. In the case of deciding if an object is pestware, we might analyze a range of object activities, and have a set of heuristic rules for making a decision on whether the object should be marked as pestware. Examples of the heuristic rules may be if analyzed activities are sufficiently like the activities of known malware and if the object was recently installed/modified.

G. Logging/Logs

84. Logging is the act of recording a record of events (problems, errors, information, activity, etc.) of operations on a computing system. Accordingly, a log contains a record of data about computer activity that is stored (at least temporarily) for later use/analysis. A log consists of one or more entries, each of which represents a single activity occurrence and includes information related to the single activity occurrence.

85. Log entries may be stored in a range of storage types, from memory to files to databases. For example, in Cisco routers, logs can be sent to a variety of destinations, including console, files, and memory. An administrator can log to memory using the *logging buffered* command (“all the log messages will be stored

in the logging buffer instead of scrolling past on the screen. The default size of the logging buffer is platform-specific, but you can change the size by adding a byte count to the end of the logging buffered command. The buffer size can be from 4096 to 4294967295 bytes.” Ex. 1017, p. 383. The logging buffered command can also be used for Cisco firewall logging. In this example, one can “[l]og to the firewall's internal buffer. Internal buffered logging uses a 4096-byte circular memory buffer to store the most recent messages.” Ex. 1018, p. 8. When this circular memory buffer fills, new messages prompt the deletion from memory of the oldest message. In another example discussing Digital Continuous Profiling Infrastructure, Anderson et al. describes temporarily logging to memory. Ex. 1019, pp. 366, 371, FIG. 5. Such logs contain a trail of activities that may be used to debug a service error, understand a security compromise, etc. and may be recorded only as long as necessary to utilize the information. For example, consider the following

log entry from an access log:

Oct 12 1:23:12: Failed password for admin from 123.45.67.89 port 3425.

An administrator or pestware analyzer who observed many of these failed login attempts in a short timeframe might conclude that somebody is trying to guess the admin password.

VII. OVERVIEW OF THE PRIOR ART

A. OVERVIEW OF U.S. PATENT NO. 8,719,924 TO WILLIAMSON (“WILLIAMSON”)

86. Williamson is a U.S. Patent filed on March 3, 2006, and issued on May 6, 2014.

87. Williamson is directed to analyzing activities to find software performing unwanted actions on a computer. Cf. Ex. 1001, 1:35-39, and Ex. 1005, 1:49-53. Williamson recognizes the deficiencies in current technology, particularly in uniquely identifying malicious code. Ex. 1005, 1:16-36, 2:1-8. Williamson relates to detecting harmful software and malicious computer programs by creating and using classifiers to determine whether the program is acting maliciously. Ex. 1005, 1:49-57. An exemplary system that detects various classes of malware is shown in FIG. 2 below:

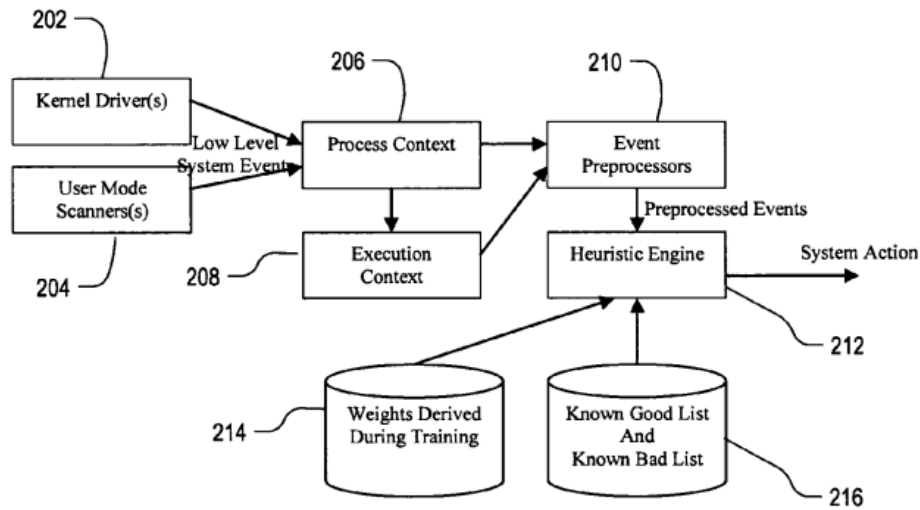


FIG. 2

Ex. 1005, FIG. 2.

88. A POSITA would have recognized that Williamson and the ‘243 patent are related to the fields of detecting, controlling, and removing security threats to a computer. Cf. Ex. 1001, 1:35-39, and Ex. 1005, 1:49-53.

89. A POSITA would have also recognized that Williamson describes solutions to the same problems as the ‘243 patent such as using definitions to detect malware. Cf. Ex. 1001, 1:62-2:3 (pestware definitions are sole and cumbersome to create), and Ex. 1005, 1:29-37, 5:34-35, 2:1-3 (“instead of trying to uniquely identify the malicious code, the idea is to detect and constrain the behavior of the malicious program”).

90. A POSITA would have also recognized that Williamson describes solving the problem in the same manner as the ‘243 patent. The ‘243 patent describes the invention as identifying malicious programs through monitoring activity indicative of pestware. Ex. 1001, 1:55-2:3, 2:19-26. Williamson summarizes its focus on identifying malware through behavioral analysis while recognizing pestware definitions are sole and cumbersome to create. Ex. 1005, 1:49-2:18. Both patents further describe functionality of identifying some software component as the source of unwanted activity. To carry out their function, both patents assess if activity rises to the level of identification as potentially harmful. The ‘243 patent describes identifying malware based on computer activity using heuristics to see if a threshold is exceeded in order to determine if the computer activity indicates potentially harmful software. Ex. 1001, 10:14-20. Likewise, Williamson also uses behavior to assess the probability of potentially harmful software. Ex. 1005, 15:49-55. Williamson uses activity to define potentially malicious behavior. Ex. 1005, 9:1-2, 10:41-42.

91. Further, a POSITA would have also recognized that Williamson and the ‘243 patent recite related structures, including kernel drivers (e.g., ‘243 ref. 116 in Fig. 1 “activity monitor is realized by a kernel-mode driver,” ‘243 5:42-43, and Williamson ref. 202 in Fig. 2) and heuristics (e.g., ‘243 ref. 110 in Fig. 1 and Williamson Fig. 2). *See*, Ex. 1001 FIG. 1, 5:42-43, and Ex. 1005 FIG. 2. Williamson describes a

Kernel Driver and User Mode Scanners that hook various API calls. Ex. 1005, 4:65-67. Low level system events associated with API calls are evaluated by a heuristic engine using a classifier based on weights generated during training, and the process associated with the event is determined to be either a malware or not based on the result. Ex. 1005, 5:5-33.

B. OVERVIEW OF U.S. PATENT APPLICATION PUBLICATION NO. 2007/0016951 TO PICCARD (“PICCARD”)

92. Piccard is a U.S. Patent Application filed July 13, 2005, and published on January 18, 2007.

93. Piccard is directed to systems and methods for managing malware through the collection of information from a history log on the protected computer. Ex. 1006, ¶¶ [0005]-[0007]. For example, “[i]f the malware detection module 118 determines that the protected computer 102 includes malware, the history log module 120 collects information from one or more history logs maintained by the application programs.” Ex. 1006, ¶ [0017]. After identifying malware, the “malware removal module 124 operates to remove the malware on the protected computer 102.” Ex. 1006, ¶ [0019].

94. A POSITA would have recognized that Piccard, like the ‘243 patent, describes detecting, controlling, and removing security threats to a computer. Ex. 1001, 1:35-39; Ex. 1006, ¶¶ [0005]-[0007]. The ‘243 patent specifically recites “detecting,

controlling and/or removing pestware.” Ex. 1001, 1:37-38. Piccard specifically recites “identifying sources of malware.” Ex. 1006, ¶ [0001].

95. A POSITA would have also recognized that Piccard describes solutions to the same problems as the ‘243 patent. Both Piccard and the ‘243 patent recite almost identical “Background of the Invention” sections. Cf. Ex. 1001, 1:43-45 (“Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as ‘malware,’ ‘spyware’ or ‘pestware’”), and Ex. 1006, [0002] (“Personal computers and business computers can be vulnerable to attack by computer programs such as keyloggers, system monitors, browser hijackers, dialers, Trojans, spyware, and adware, which are collectively referred to as ‘malware’ or ‘pestware’”).

96. Another problem recognized by both Piccard and the ‘243 patent is using definitions to detect malware. Cf. Ex. 1001, Abstract, 1:62-2:3 (pestware definitions are sole and cumbersome to create), and Ex. 1006, ¶ [0003] (“it is often difficult to initially locate malware in order to generate definitions, particularly since malware can evolve. It would be desirable to identify sources of malware, such that definitions can be rapidly generated or updated to account for new or evolving malware”).

97. A POSITA would have also recognized that Piccard further describes solving problems in the same manner as the ‘243 patent. Both Piccard and the ‘243 patent describe functionality of identifying some software component as unwanted. The

‘243 patent identifies “a suspected pestware object.” Ex. 1001, 10:18-19. Similarly, Piccard recites “detecting malware on a protected computer.” Ex. 1006, Claim 1. To carry out this function, both Piccard and the ‘243 patent utilize a history of activities in one or more logs to detect malware and communicate such detection for the purpose of identifying a malware source. Both Piccard and the ‘243 patent focus on utilizing analysis of computer activities to find software performing unwanted actions on a computer. The ‘243 patent describes identifying unwanted software through monitoring activity indicative of pestware. Ex. 1001, 1:55, 2:19-26. Likewise, Piccard describes identifying sources of malware through analysis of a history log, which is a “record of events related to operation of that application program.” Ex 1006, Abstract, ¶ [0014]. The ‘243 patent further explains these functions by reciting “identifying, based upon the activity, an object residing on the computer that is a suspected pestware object” where the activity is from an activity log, and “reporting the identity of the particular externally networked source to an externally networked pestware research entity.” Ex. 1001, 10:12-19, 10:26-28. Williamson claims similar functionality, reciting “collecting information from a history log of the protected computer” and “directing the protected computer to convey the information to a host computer, such that the information can be used to identify a source of the malware.” Ex. 1006, Claim 1.

**C. OVERVIEW OF U.S. PATENT NO. 7,934,103 TO KIDRON
("KIDRON")**

98. Kidron is a U.S. Patent filed on April 15, 2003, and published on April 26, 2011.

99. Kidron is directed to systems and methods for detecting and countering known and unknown malicious code by analyzing computer activity to identify irregularities. Ex. 1007, 4:16-21. A POSITA would have recognized that Kidron, like the '243 patent, describes detecting, controlling, and removing security threats to a computer. Cf. Ex. 1001, 1:35-39, and Ex. 1007, 3:11, 3:25-40. Kidron specifically recites "[m]alicious code deployment of unknown nature . . . may be detected through pattern recognition technology," and that "[a]fter detection of malicious code is confirmed, an abstract, the remote control core may be employed to promptly deploy changes, patched, or any possible action needed to handle the threat." Ex. 1007, 4:58-62, 5:12-15. Similarly, the '243 patent specifically recites "detecting, controlling and/or removing pestware." Ex. 1001, 1:37-38.

100. A POSITA would have also recognized that Kidron describes solutions to the same problems as the '243 patent. Like the '243 patent, Kidron recognizes the deficiencies in current technology, particularly in uniquely identifying malicious code. For example, the '243 patent recites that "[c]urrent pestware removal software uses definitions . . . [t]hese definitions are often slow and cumbersome to create."

Ex. 1001, 1:63-66. Likewise, Kidron recites that “[m]ost conventional security mechanisms rely on detecting fixed, known virtual signatures . . . [h]owever, the signature files . . . must appropriately be updated before new malicious code can be detected.” Ex. 1007, 2:50-56. Kidron further recognizes that some pestware is able to change its signature and therefore evade signature based technologies. Ex. 1007, 2:15-19.

101. A POSITA would have also recognized that Kidron further describes solving problems in the same manner as the ‘243 patent. Both Kidron and the ‘243 patent describe functionality of detecting unwanted software components without relying on signature technology. Kidron recites “[d]etection of malicious code may be accomplished through monitoring for irregular operations.” Ex. 1007, 4:18-20. Kidron further discloses that a pattern recognition processor monitors activities for “irregular behavior patterns, and generates an alert after an irregularity in local behavior pattern is detected.” Ex. 1007, 4:47-49. Similarly, the ‘243 patent describes identifying malware based on computer activity using heuristics to see if a threshold is exceeded in order to determine if the computer activity indicates potentially harmful software. Ex. 1001, 10:14-20.

102. Further, a POSITA would have also recognized that Kidron and the ‘243 patent recite related structures, including kernel drivers and heuristics. *See*, Ex. 1001, 5:42-43, FIG. 1, and Ex. 1007, 4:58-62, 5:47-51, 6:36-47. Kidron describes that a

client includes an operating system signal and the client may be a compact extension to an operating system kernel. Ex. 1007, 5:37-48. A POSITA would have recognized Kidron to be describing a kernel driver. Kidron further discloses that the pattern recognition processor may use “heuristic scanning,” and other pattern recognition technology such as “neural nets, clustering techniques, decision tree techniques, etc.” Ex. 1007, 4:61-62, 6:42. A POSITA would have recognized that neural nets, clustering techniques, and decision tree techniques all essentially “learn” heuristics which they use to analyze new data.

VIII. IDENTIFICATION OF HOW THE CLAIMS ARE UNPATENTABLE

A. Ground 1: 1-7 and 9-14 are obvious under 35 U.S.C. § 103 over Williamson in view of Piccard

1. Reasons to Combine Williamson and Piccard

103. A POSITA would have understood that Williamson’s basic “idea is to detect and constrain the behavior of the running program. The behavior-based mechanism runs the malicious program, and from its running behavior determines whether the program is behaving maliciously.” Ex. 1005, 2:2-6. A POSITA would have recognized that allowing the running of malicious programs (or potentially malicious programs) endangers the computer system because malicious operations of the malicious program can harm the computer system prior to detection. Williamson itself describes an example of endangering of the computer system to include self-

propagation, allowing unauthorized access, recording information, and recording user actions. Ex. 1005, 1:62-67. Thus, if the behavior is malicious, Williamson has already endangered the computer by allowing execution of malicious operations of the malicious program.

104. A POSITA would have been motivated to improve Williamson and prevent malicious operations from sources of suspected malware from executing in the first place and thus avoiding harm to the computer system.

105. Piccard explicitly solves the problem of the risk to the system of Williamson by attempting to preemptively (i.e., prior to execution) detect malware. Piccard approaches this in several ways. First, files are inspected after download to “detect a presence of malware that is downloaded using a Web browser.” Ex. 1006, ¶ [0006]. This is accomplished by analyzing downloaded files. “For example, the malware detection module 118 can analyze files that are downloaded using the Web browser 114 to determine whether those files include malware.” Ex. 1006, ¶ [0016]. Piccard works “to detect a presence of malware that is downloaded using a Web browser (e.g., the Web browser 114) (block 200).” Ex. 1006, ¶ [0022].

106. Piccard continues, a malware detection module (e.g., the malware detection module 118) detects the presence of malware on a protected computer (e.g., the protected computer 102) by monitoring the protected computer on a periodic or some other basis. It is also contemplated that operation of the malware detection module

can be triggered based on a particular event, such as a Web browsing event. For example, once a file is downloaded using the Web browser, the malware detection module can analyze the file to detect the malware in the file without execution or loading. Ex. 1006, ¶ [0023]. “In the illustrated embodiment, the malware detection module detects the presence of the malware on the protected computer based on a set of malware definitions. In particular, the set of malware definitions can include representations of known malware, and the malware detection module can scan files of the protected computer to detect the malware in one of the files. For example, the set of malware definitions can include a hash value or a digital signature of known malware. Such as one that is generated using Message Digest 5 (‘MD5’). In this example, the malware detection module can generate a hash value of a particular file to be analyzed and compare the hash value of that file with a set of hash values of known malware to determine whether there is a sufficient match. As another example, the set of malware definitions can include a Cyclical Redundancy code (‘CRC’) of a portion of known malware. In this example, the malware detection module can generate a CRC of a particular file to be analyzed and compare the CRC of that file with a set of CRCs of known malware to determine whether there is a sufficient match.” Ex. 1006, ¶ [0023]. Piccard therefore, addresses the problem in Williamson of allowing potentially dangerous execution of malware to enable

detection by analyzing the files prior to execution, including by the use of the described MD5 hash and CRC.

107. Second, Piccard maintains a blacklist of malware sources, warning users when they attempt to download from these sites. For example, “[a]s illustrated in FIG. 1, the database 126 also includes a blacklist of sources of malware. For example, the blacklist of sources of malware can include a list of malware distribution sites to alert a user about those Web sites that are known to distribute malware or that are suspected of distributing malware.” Ex. 1006, ¶[0015]. Piccard provides protection without dangerous execution by helping users avoid downloading potential malware from blacklisted sites.

108. A POSITA would have readily understood another deficiency of Williamson related to continuous training and update. Williamson operates by training on new malware and normal programs to derive the characteristics and weights used to detect malware. “FIG. 1 shows that a malware database 101, **constructed from the ongoing collection of new malware 102, is used to train the system 103.**” Ex. 1005, 3:62-64 (emphasis added). Williamson enables detecting “new and never seen instances of malware” through learning “from existing instances of malware in an automated fashion” Ex. 1005, 8:4-6. Training information is then “delivered to the customer machine, as the system running protecting against new malware 110.” Ex. 1005, 2:37-38. Initially, “the values for the weights are calculated during training”

and then “are shipped along with the software.” Ex. 1005, 4:62-63. To be used on the protected computer according to the “overall architecture of the system is shown in FIG. 2.” Ex. 1005, 4:63-64. Williamson acknowledges that “new types of malware (e.g. mail worms, Trojans, MacroViruses, etc. . . .) require new behavior detectors added to the anti-malware program” through “incremental updates . . . as the types of programs written evolve, changes would be needed to the characteristics and classifier,” and require the feature of being “updatable and extensible for new type of malware through a standardized process” Ex. 1005, 2:25-28; 5:38-40; 8:16-17. From this, Williamson concludes that “the list of characteristics and the classification mechanisms are extensible, and a mechanism is provided whereby new characteristics and classification types can be added to the system as and when needed.” Ex. 1005, 5:40-44. In particular, the heuristic engine 212 from Fig. 2 needs to be updated with such training results from identified new known malware. A POSITA would have understood that regular and timely updates to training data would be required to maintain effectiveness. Williamson acknowledges that “[a]s a rule of thumb, more training and more diverse training is better.” Ex. 1005, 6:35-36. So, ongoing training data is imperative.

109. Piccard directly addresses this shortcoming in Williamson through continuous improvement of malware detection via update using data collected by protected computers. The Reporting Module 122 in Fig. 1 “reports information relating to the

set of Web sites to a *remotely located host computer*.” Ex. 1006, ¶ [0028] (emphasis added). “This information can identify the set of Web sites as potential malware distribution sites” and “this information can include a representation of the malware or can identify suspicious activities related to the malware.” Ex. 1006, ¶ [0028]. The *remotely located host computer* can analyze this information, “a new or updated set of malware definitions can be generated,” and “the new or updated set of malware definitions can be provided to the protected computer.” Ex. 1006, ¶ [0028].

110. This approach by Piccard improves “the efficiency at which sources of malware can be identified” through a “decentralized” approach. This is accomplished by enabling collaboration between a multitude of protected computers (102) to identify malware (i.e., “[T]he illustrated embodiment allows automated collection and reporting of relevant information once malware is detected on the protected computer 102”). Ex. 1006, ¶ [0020]. Such information is reported to separate system (i.e., remotely located host computer), analyzed, and a “new or updated set of malware definitions can be provided” to the malware detection systems. Ex. 1006, ¶ [0028].

111. A POSITA would understand that Piccard’s reporting system can be utilized to provide continuous input for novel and timely training to Williamson 103, Fig. 1. Further, Piccard’s enables “the new or updated set of malware definitions” to be

“provided to the protected computer.” Ex. 1006, ¶ [0028]. This capability could be used to update Williamson’s Heuristic Engine 212 in Fig. 2.

112. A POSITA would have found it obvious to modify Williamson’s heuristic engine to incorporate Piccard’s malware detection module, history log module, and reporting module. As described, a specific motivation in Williamson is that its heuristics engine analysis can evolve (i.e., the list of characteristics and the classification mechanisms are extensible, and a mechanism is provided whereby new characteristics and classification types can be added to the system as and when needed). New characteristics and classification types can be readily incorporated to identify sources suspected of malware, as in Piccard. Ex. 1005, 5:39-43.

113. Williamson further discloses that one of the characteristics of computer activity that is evaluated is whether the executable accessed the network, “either as a client accessing services on other machines, or listening on a certain network port. **Malicious programs need to use the network to communicate with their controllers, send out information, receive software updates etc.**” Ex. 1005, 8:62-66 (emphasis added). Williamson already contemplates evaluation of a networked source of the malware.

114. Williamson’s system modified by Piccard would therefore operate to identify malware prior to malware execution as in Piccard and implement Williamson’s analysis to identify both identified sources of suspected malware and unknown code

using updated training on new known malware as malware. The combined system would identify malware using e.g., MD5, CRC, blacklists and as applied by the modified heuristic engine as a result of updated classification developed through training on new known malware.

115. A POSITA would have recognized that this combination would have improved Williamson's known system of detecting various classes of malware with Piccard's known identification of malware prior to execution and identification of new sources of suspected malware. This combination would have been well within the abilities of a POSITA, requiring only programming updates without undue experimentation for execution on the hardware of Williamson. Ex. 1005, FIG. 3, ref. 310; Ex. 1006, ¶ [0016]. In particular, the known malware detection module 118, the history log module 120, and the reporting module 122 that operate to facilitate identification of sources of malware of Piccard could have been readily programmed into the system of Williamson. Likewise, a POSITA would have recognized that database 126 of Piccard could have readily been integrated with the combined system, as in the blacklist database example, because simple database calls would likewise have been well within the programming abilities of a POSITA. Moreover, Williamson's heuristics engine already accesses whitelists/blacklists stored in known good list and known bad list 216. Ex. 1005, FIG. 2, 4:29-30. Using such list

data would require only basic reprogramming to checks made by the modified system.

116. Therefore, a POSITA would have appreciated a reasonable expectation of success making software modifications that require no more than the ordinary skill of a POSITA. The modified Williamson/Piccard system and method would operate normally, by detecting known malware prior to execution and identifying suspected malware based on behaviors.

2. Claim 1

[1.0] A method for identifying an origin of activity on a computer that is indicative of pestware comprising:

117. I understand that a preamble typically does not limit a claim. However, to the extent that the preamble is limiting, it is my opinion that it is disclosed. Williamson describes methods to addressing the problem of “malicious code running on computers,” and in one example, a “behavior based mechanism” that “determines whether the program is behaving maliciously. This allow[s] unknown code to be identified as malware.” Ex. 1005, 1:61-62, 2:4-7. In an example system, Williamson discloses systems and methods for identifying sources of malware. A system includes a malware detection module configured to determine that a protected computer includes malware. Ex. 1005, Abstract, 14:18-21.

118. The ‘243 Patent explains that “trojans, spyware, and adware” are “collectively referred to as ‘malware,’ ‘spyware’ or ‘pestware.’” Ex. 1001, 1:43-54.

119. Williamson describes methods for addressing “worms and viruses which are programs that self-propagate; trojans or backdoors which are programs that allow unauthorized access to machines, as well as record information (passwords etc.); and spyware which is software to record user actions.” Ex. 1005, 1:61-67. In another example, Williamson describes “some malicious programs (particularly adware) install themselves in [the Program Files] directory.” Ex. 1005, 9:56-58. In yet another example, “[t]rojans are used as an example of a particular malware.” Ex. 1005, 2:28-30. A POSITA would have recognized that the examples identified by Williamson teach at least one or more types of “pestware” as described by the ’243 Patent because all are categorized under the ’243 Patent definition of “pestware” and further under the construction of “pestware” as “a trojan, spyware, adware, or malware program that collects information about a person or an organization.” Ex. 1014, pg. 3. More particularly, Williamson’s examples to address “record[ing] information (passwords etc.)” and “software to record user actions,” correspond to the collection of information about a person or an organization. Ex. 1005, 1:61-67.

120. Williamson directly discloses identifying an origin of activity, for example in claims 1 and 10, “identifying the software application as harmful.” Ex. 1005, 15:59, 16:52-53, 16:55-56, 16:59-60. Here, Williamson identifies the harmful software application as pestware. In another example, Williamson describes identification of

a Trojan based in claim 3, “identifying the software application as a Trojan Horse.”
Ex. 1005, 16:3.

121. As explained above, Williamson in view of Piccard discloses the method for identifying an origin of activity on a computer that is indicative of pestware as claimed. It is my opinion that it would have been obvious to modify Williamson in view of Piccard for the reasons discussed above, and in particular to identify the source of the malicious code, which is explicitly disclosed by Piccard (“directing the protected computer to convey the information to a host computer, such that the information can be used to identify a source of the malware.”). Ex. 1006, ¶ [0007].

[1.1] monitoring, using a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the monitoring includes monitoring API calls and storing a history of at least a portion of the API calls in an activity log;

122. It is my opinion that Williamson discloses each part of this limitation.

123. As I have explained, a driver is a software component that implements the OS’ standardized, programmatic interface for a particular subsystem. A kernel-mode driver is a type of driver capable of operating with kernel-mode access. Williamson describes “a Kernel Drive [sic] 202 and User Mode Scanners 204 that hook various components of the operating system. The kernel driver 202 looks at the requests made by programs running on the protected system, such as system calls, graphics, and windows calls,” as highlighted below in annotated Figure 2. Ex. 1005, 4:63-5:2.

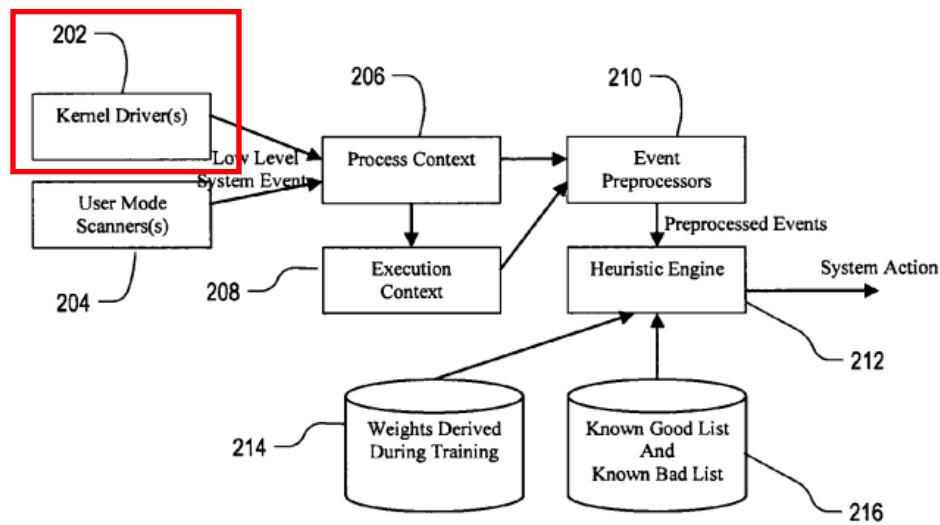


FIG. 2

Ex. 1005, FIG. 2 (annotated).

124. A POSITA would have understood kernel driver and kernel-mode driver to be equivalent terms. As explained by Williamson above, a kernel-mode driver is a software component 1) executing with elevated privileges (i.e., above user-mode) and 2) implementing an API. More particularly, Williamson describes, “[t]he kernel driver 202 looks at the requests made by programs running on the protected system, such as system calls, graphics, and windows calls,” which means it can observe API calls made by programs where such observation requires special privileges because monitoring such system calls, graphics, and windows calls requires heightened access. Ex. 1005, 4:67-5:2.

125. The kernel driver monitors requests by observing events. For example, “[w]hen these components detect an interesting operation that generates a Low Level System event, the events are passed to a process content 206.” Ex. 1005, 5:5-7. As I previously explained, computer activity includes behaviors carried out by a computer. These events of Williamson are examples of computer activity (see “events are any type of resource, action, or status request that a running program makes from the operating system or any of its extensions”) Ex. 1005, 5:8-10. Williamson’s heuristic engine 212 uses this event record to determine if such activity is indicative of pestware (“[o]n every new event passed to the event preprocessor 210 . . . the characteristics are calculated by the heuristic engine 212 . . . and the process is determined to be either a malware or not based on the result.”) Ex. 1005, 5:28-33.

126. The system calls, graphics, and windows calls monitored using hooks by kernel driver 202 are examples of API calls. As I explained above, to invoke a service of the API, a developer “calls” the service in an Application Programming Interface. Accordingly, an API call is the actual invocation of service via the API.

127. Williamson describes many example behaviors carried out by API calls. For example, to “turn off a firewall;” “load a kernel module;” or spawn “one or more processes,” Williamson describes accomplishing through API calls (e.g., system

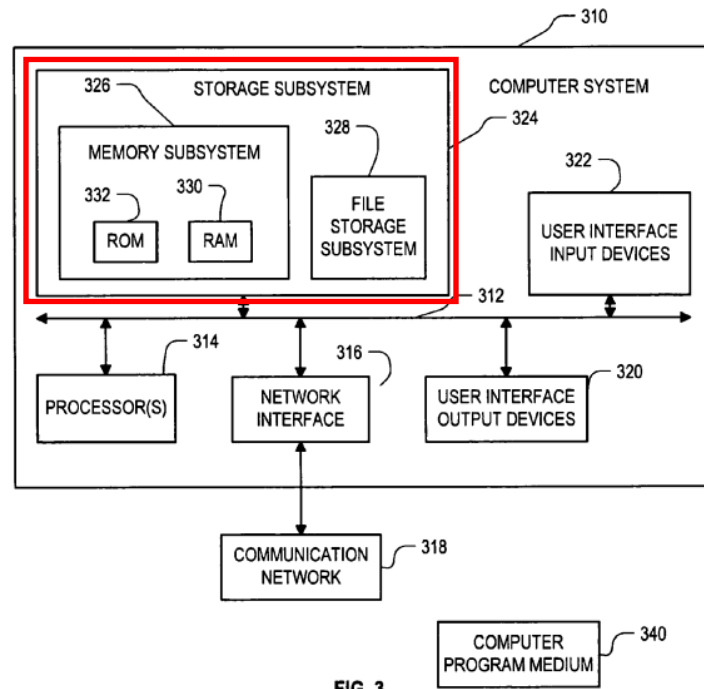
calls). Ex. 1005, 16:40-42. If done through the command line, the command executes the API call.

128. As I have also explained, logging is the act of recording a record of events (problems, errors, information, activity, etc.) of operations on a computing system. A log consists of one or more entries, each of which represents a single activity occurrence. Log entries may be stored in a range of storage types, from memory to files to databases.

129. Williamson discloses storing a history of at least a portion of the API calls in an activity log. When the kernel driver 202 detects “an interesting operation that generates a Low Level System event, the events are passed to a process context 206.” Ex. 1005, 5:5-7. These events are “kept at the process context,” or “applied to the executable context 208” Ex. 1005, 5:16, 19. A POSITA would understand that to keep the event, it must be stored. The events are then “passed to the event preprocessor 210 from the process context 206 or the execution context 208.” Ex. 1005, 5:28-29. The Event Preprocessor 210 then stores these events until the Preprocessed Events can be passed to the Heuristics Engine 212 (e.g., see Fig. 2 above). Williamson must store these events to carry out various operations on the event. For example, in Williamson method stores the behavior to execute the various “determining” and “identifying” steps. Ex. 1005, 17:9-08, 18:28-45.

130. As explained above, API calls are an example of an event described in Williamson because an API call is a type of resource, action, or status request that a running program makes from the operating system or any of its extensions.

131. Williamson also explains that “events that are kept at the process context.” Ex. 1005, 5:16. Williamson further discloses storage of data constructs such as the process context. “Storage Subsystem 324 stores the basic programming and data constructs that provide the functionality of certain embodiments of the present invention.” Ex. 1005, 14:59-61. A POSITA would have understood that events must be stored for processing. More particularly, a POSITA would have understood that such storage would occur in at least 206, 208, 210, and 212 of Fig. 2. Williamson’s “STORAGE SUBSYSTEM” of Fig. 3 provides a number of locations where a portion of the history of API call events, representing computing activities, are stored for such processing, as highlighted in annotated Figure 3 below.



Ex. 1005, FIG. 3 (annotated).

132. A POSITA would have understood that the events kept at the process context, for example, by storage subsystem 324 could be stored in volatile RAM 330 and/or file storage subsystem 328. This description is consistent with the storage as described by the '243 Patent, "file storage device includes an activity log 120." Ex. 1001, 3:10-11. Specifically, at least 206, 208, and 210 store events in storage in order to subsequently process the events.

[1.2] analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware;

133. It is my opinion that Williamson discloses each part of this limitation.

134. As I explained above, Williamson's kernel driver 202 monitors computer activity, including API calls, to be recorded as events associated with the activity. *Supra*, Section VIII.A.2. [1.1]. As I explained above, computer activity refers to a behavior, action, or event carried out by a computer. Williamson describes a heuristics engine to analyze the events. Annotated Figure 2 below depicts the heuristic engine.

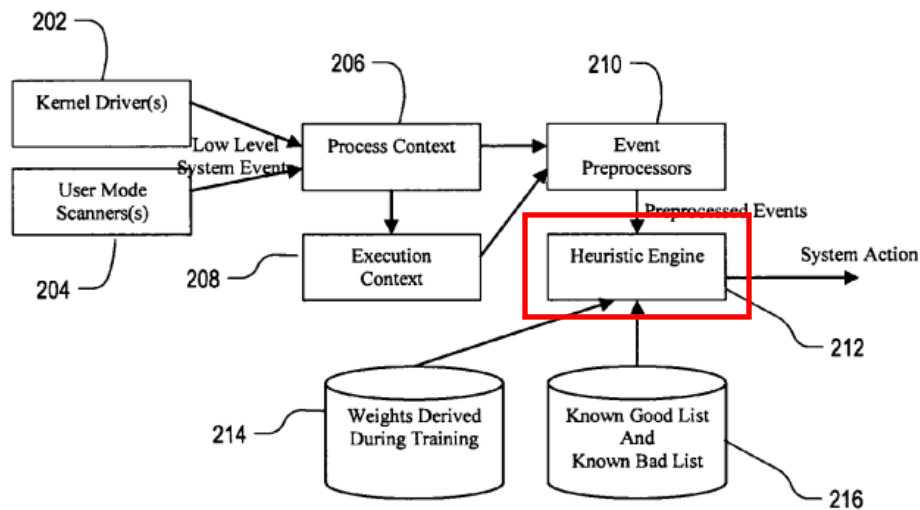


FIG. 2

Ex. 1005, FIG. 2 (annotated).

135. On every new event passed to an “event processor 210 from the process context 206 or the execution context 208, the characteristics are calculated by the heuristic engine 212 based on weights generated during training 214, based on [Evaluation Equation] (see Bayes Classifier) and the process is determined to be

either a malware or not based on the result.” Ex. 1005, 5:28-33. In this example, the characteristics calculated by the heuristic engine 212 based on weights is an analysis of computer activity because characteristics of the activity are evaluated (e.g., using weights). A resulting indication of malware or not is a determination that the activity is indicative of pestware.

136. Williamson’s heuristics engine 212 implements a heuristic Bayes Classifier, a well-known classifier “trained based on the data and samples as shown in FIG. 1” “that calculates the probability that a class of programs (e.g. trojans) will exhibit a particular characteristic (e.g. have a visible window).” Ex. 1005, 5:45-46, 55-58.

137. The classifier in Williamson uses “the key premise . . . that most of malware of the same type exhibit similar behaviors. These behaviors are categorized and detectors for them are created and weights are automatically assigned to through a learning process.” Ex. 1005, 2:12-16. “One embodiment uses a Bayes classifier to learn the weights (tuning parameters) from known malware of a particular type and from normal programs.” Ex. 1005, 2:19-21. Williamson’s “classifier is trained by calculating $P(x_i|\text{normal})$ and $P(x_i|\text{Trojan})$ for each characteristic. Those values are then used to predict what type a new program is.” Ex. 1005, 6:38-39. These values are combined in a summation over the various characteristics’ probability ratios.

$$\sum_i \alpha_i \log [P(x_i|c1)/P(x_i|c2)] + \alpha_0 \log [P(c1)/P(c2)] > 0$$

α_i is a weighted factor used to determine the “strength of each characteristic” that generates the best prediction over the training data. Ex. 1005, 6:49.

138. Williamson teaches evaluation of multiple weighted factors against a threshold. In describing a two-class problem (c_1 and c_2), Williamson teaches “individual values of $P(x_i|c_j)$ can be estimated from the data, and for a new data point this equation can be evaluated. If the function is >0 , then it is predicted to belong to class c_1 , else c_2 .” Ex. 1005, 6:19-6:22. In this example, evaluation (e.g., known malware or pestware) of the probability function P is against the threshold 0. *See*, Ex. 1005, 5:47-6:22. Thus the heuristic Bayes Classifier can operate on one or more (weighted) factors associated with an activity (e.g., “a data vector X that has data elements x_i , e.g. $x_1, x_2, x_3 \dots x_n$ ”) Ex. 1005, 5:54-55.

139. Williamson’s “classifier is trained by calculating $P(x_i|\text{normal})$ and $P(x_i|\text{Trojan})$ for each characteristic. Those values are then used to predict what type a new program is.” Ex. 1005, 6:38-39. These values are combined in a summation over the various characteristics’ probability ratios.

$$\sum_i \alpha_i \log [P(x_i|c_1)/P(x_i|c_2)] + \alpha_0 \log [P(c_1)/P(c_2)] > 0$$

α_i is a weighted factor used to determine the “strength of each characteristic” that generates the best prediction over the training data. Ex. 1005, 6:49.

140. In one particular example of weighted factors associated with an activity, Williamson teaches, “some of the characteristics are based on which directory in the file system the malicious code runs from. A particular directory may be commonly used by different forms of malware, thus resulting in the classifier learning that executing from that directory is a good indication of maliciousness.” Ex. 1005, 6:50-52. Here, the characteristic of the activity is the file system directory. Weighted factors associated with execution from a given file system directory thereby associate a strength to the characteristic of execution from that directory.

141. Williamson also teaches evaluation of a single weighted factor against a threshold. Williamson distributes alpha weights based on training for “determining an updated probability that the software application is malicious based on [a] behavior, and in response to the updated probability exceeding a threshold value . . . identifying the software application as harmful.” Ex. 1005, Claim 1. In this case, the “updated probability” is evaluated as to whether it exceeds a “threshold value” to determine harmfulness, or indicative of pestware.

142. Here, Williamson’s updated probability based on a behavior in claim 1 refers to evaluating an individual aspect of program behavior. More particularly, the aspect of program behavior is an example of computer activity. As I explained above, computer activity is behavior carried out by a computer, and that computer activities

include a large range of behaviors such as file creation and reading/writing to/from the network, etc.

143. Further, Williamson's evaluation uses a weighted characteristic. Williamson treats as equivalent, "aspects of the program behavior" being measured to "characteristics." Ex. 1005, 2:9-10. Williamson teaches several examples of such characteristics, including 1) Surviving a reboot, 2) Stealth, 3) Disguised executable, 4) Network, 5) Keylogger, 6) Installation actions, 7) Injecting code into other processes), Ex. 1005, 3:27-45. A POSITA would have understood that the above characteristics necessarily require computer activities to be performed. Williamson's behavioral approach relies on heuristics and individual probabilities of computer activity as associated with malware. Ex. 1005, 2:30-32. Williamson teaches several other characteristics and that, "[d]ifferent classes of malware will require different characteristics. For example, for mass-mailers (viruses or worms that spread themselves via email), the following characteristics could be considered in addition to those above 1) Searching the files system. 2) Accessing email address books stored on the computer. 3) Sending large volumes of email to different recipients. 4) Querying the Domain Name System (DNS) to find the addresses of computers that accept mail for particular domains, so called MX records." Ex. 1005, 3:66-4:12. In still other examples, "[f]or viruses that infect files, e.g., by writing themselves into the executable on the file system, the following characteristics would be important

1) Searching the file system for files to infect. 2) Modifying existing executables.”

Ex. 1005, 4:15-19.

144. Though Williamson’s described updated probability exceeding a threshold value is additionally taught in claim 1, it is my opinion that it would have been obvious to modify the heuristic engine 212 to include evaluation against thresholds for individual activities.

145. Williamson recognizes that “malicious code spreads quickly.” Ex. 1005, 1:30-31. Accordingly, a POSITA would have recognized by the effective date of the ‘243 patent that timely and efficient detection of malware is critical to any malware detection system. More particularly, without timely and efficient detection, a computer system is vulnerable to any number of threats. Evaluating fundamental action malware behavior against a threshold for malware improves detection speed and efficiency.

146. As I have explained, a heuristic analysis is the process of analyzing a problem based on a set of one or more heuristics. In the case of deciding if an object is pestware, a range of object activities can be analyzed according to a set of heuristic rules for making a decision on whether the object should be marked as pestware. Williamson explains that one of the key premises behind its technology is that most malware of the same type exhibit similar behaviors. Ex. 1005, 2:12-14. Williamson also explains particular emphasis on individual characteristics of a behavior, such as

“\alpha_i allows the ‘strength’ of each characteristic to be independently varied.”

Ex. 1005, 6:49-50. Williamson further teaches uniqueness of some behaviors. “Some of these characteristics would be rare in normal programs (e.g. having a keylogger).”

Ex. 1005, 3:49-50. Therefore, certain key behaviors should be emphasized. Evaluating a fundamental characteristic of malware against a threshold for that behavior characteristic would speed up malware detection. Claim 1 of Williamson indeed teaches that, “each of the plurality of behaviors defines a specific action that the software application performs.” Ex. 1005, Claim 1. Moreover, Williamson also describes “per-characteristic granularity, e.g. certain programs may be ‘allowed’ to exhibit certain characteristics without that contributing to their overall score.” Ex. 1005, 4:26-29.

147. A POSITA would have had a reasonable expectation of success in making this modification. Williamson explains both combined characteristics evaluation against a probability threshold and individual characteristic evaluation against a probability threshold. Ex. 1005, 3:51-59 (“combination mechanism for the characteristics . . . to calculate an overall score for each class”), 4:49-52 (“extra set of weights that vary the importance of the individual probabilistic calculations, so for example increasing the importance of the keylogger characteristic”), Claim 1 (“updated probability that the software application is malicious based on” “the specific action that the software application performs”). Indeed, this is just combining already-present prior art

elements according to known methods already described expressly in Williamson to predictably yield a system for evaluating one or more factors of computer activity.

148. It is my opinion that it would have been obvious to modify Williamson to evaluate computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware.

[1.3] identifying, based upon the activity, an object residing on the computer that is a suspected pestware object;

149. It is my opinion that Williamson discloses each part of this limitation.

150. As I have explained, a computer object is some component containing data that resides in storage (e.g., memory, disk). For example, an executable is a file object that is stored on a computer's hard drive, where some of the data is a set of instructions to execute. When instructions from such an object are executed, a new object called a process may be created containing the execution state and some of the instructions. Accordingly, the object of the '243 can be a running program or executable or process instance of an executable.

151. Williamson describes systems and methods to identify a suspected pestware object associated with the monitored computer activity. For example, Williamson teaches monitoring requests made by programs running on the protected system. Low Level System events (any type of resource, action, or status request that a

running program makes from the operating system or any of its extensions) applied to the process context 206 “only apply to **that particular instance of a running program or executable**. The events that are applied to the executable context 208 will apply to **every running process instance of an executable**.” Ex. 1005, 5:5-20 (emphasis added). Williamson uses events captured by the kernel driver 202 as indications of computer activities to determine if a process or executable is a pestware object. The Low Level System events applied to the process context is an evaluation based upon the computer activity. And, the described “particular instance of a running program or executable” and “every running process instance of an executable” are example objects, as I have previously explained. Ex. 1005, 5:17-19.

152. Heuristic engine 212 evaluates corresponding characteristics of an event using a classifier and the executable or “the process is determined to be either a malware or not based on the result.” Ex. 1005, 5:30-33. Here, the determination of malware (or not) of a particular process is identifying the process as a suspected pestware object according to an evaluation of the process activity.

153. Identifying behavior indicative of pestware necessarily identifies the process conducting the behavior. Typically, processes are associated with executables, so identifying the process also identifies an associated executable.

154. Williamson explains identifying the specific pestware object and its location on the computer. For example, “[t]he above four characteristics

[EXEC_FROM_CACHE, EXEC_FROM_WINDIR, EXEC_FROM_PGM_FILES, OTHER_PATH] are a report of where in the file system the executable resides . . . [I]t provide[s] some hints about the type of executable that is running.” Ex. 1005, 12:42-54. In particular, the example executables’ locations are relative to the computer upon which the object resides (e.g., computer cache location, computer WinDir directory, computer Program Files location, or another computer path).

[1.4] accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from; and

155. It is my opinion that Williamson in view of Piccard discloses each part of this limitation.

156. As I have explained above, Williamson discloses identifying a suspected pestware object (specific instance of a running program or executable or process instance of an executable) residing on the computer, based upon monitored API call activity. *Supra*, Section VIII.A.2. [1.0]-[1.3].

157. Williamson does not explicitly disclose accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from. Piccard discloses this element.

158. Piccard identifies suspected pestware objects using computer activity like Williamson. For example, Williamson characterizes detecting activity including if “the executable modified . . . portions of the registry” is “suspicious.” Ex. 1005, 10:60-63. Piccard discloses similar methods for detecting malware on a protected computer. Ex. 1006, Abstract. Similar to Williamson’s methods that operate based upon monitored computer activity, Piccard likewise discloses that its malware detection module can “monitor activities of the protected computer to detect the presence of the malware on the protected computer.” Ex. 1006, ¶[0024]. Piccard discloses detecting activities including “suspicious activities related to third-party cookies” or “entries or modifications of registry files of an operating system.” Ex. 1006, ¶[0024].

159. Piccard discloses that malware identification results in accessing at least a portion of the recorded history, i.e., “If **malware detection module 118 determines that the protected computer 102 includes malware, the history log module 120 collects information from one or more history logs** maintained by the application programs 112.” Ex. 1006, ¶ [0017] (emphasis added).

160. Piccard discloses history logs that provide a recorded history for access by a history log module. Ex. 1006, ¶ [0014]. In one example, Piccard explains that “Web browser 114 maintains a history log 116, which serves to provide a record of Web browsing events.” Ex. 1006, ¶ [0014]. Here, the history log record of web browsing

events is a portion of recorded history. In another example, “it is also contemplated that the application programs 112 can maintain **a common history log to provide a record of events** for all of the application programs 112.” Ex. 1006, ¶ [0014] (emphasis added). Here, the recorded history in the common history log contains the identity of the externally networked sources from which the computer received files.

161. As I have explained, an externally networked source is a source that is outside (i.e., external) of but accessible by the computing device. In particular, as described by Piccard, “when a user visits a Web site, the Web browser 114 records an entry that is indicative of that Web site in the history log 116. For example, when a file is accessed and downloaded from a Web site, the Web browser 114 can record a Web address of the file in the history log 116.” Ex. 1006, ¶ [0014]. The Web address provides a reference to the identity of a particular externally networked source. To be “particular,” the identity must be of a specific source; an externally networked source of “an AWS host” would not be particular. Piccard teaches, “[f]or example, a Web address can be a Uniform Resource Identifier (‘URI’) of a file, such as a Uniform Resource Locator (‘URL’) of the file. It is also contemplated that a Web address can be defined in various other ways, such as using an Internet Protocol (‘IP’) address or any other identifier of a source of a file.” Ex. 1006, ¶ [0014]. A web URL contains a host, commonly specified by name (e.g., google.com). The name or IP address provides the identity of a particular externally networked source. Here

the history log 116 is a recorded history containing references (e.g., URL) with the identity of externally networked sources identified by “Web address” for a specific file that has been received (i.e., “downloaded”).

162. The history log module utilizes the history log of past accesses to determine the identity (web address) of the particular source (web site) that the suspected pestware object (malware) originated from. “In particular, the history log module accesses the Web browser's history log to identify the n most recently recorded Web addresses in the Web browser's history log Thus, by collecting the set of Web addresses from the Web browser's history log, the history log module can facilitate identification of the set of Web sites from which the malware may have been downloaded, such as in terms of domain names of the set of Web sites.” Ex. 1006, ¶ [0026]. Here, the most recently recorded Web addresses in the Web browser's history log is the recorded history of externally networked sources that the computer received files from because a web browser receives files from websites via web addresses. Likewise, the identification of the domain names of the set of Web sites is the identity of the externally networked source, because a domain name serves to identify a particular internet resource.

163. I understand that with reference to the claimed *externally networked source*, during prosecution, the applicant explained an “externally networked” entity means “any entity external to the computer.” Ex. 1004, p. 71. Consistent with the Patent

Owner's definition, Piccard explains and depicts, for example, in FIG. 1 below, "the Web browser 114 is operated by a user who visits various Web sites that are **included in the computer network 104**. For example, the user can access and download various files from those Web sites." Ex. 1006, ¶ [0013] (emphasis added). As described earlier, an externally networked source is a source of data that is outside of but accessible by the protected computer. The web sites recorded in the Web browser's history log are thus external to the protected computer 102. Ex. 1006, ¶ [0013]. Figure 1, as annotated below, shows the computer network 104 being external to protected computer 102.

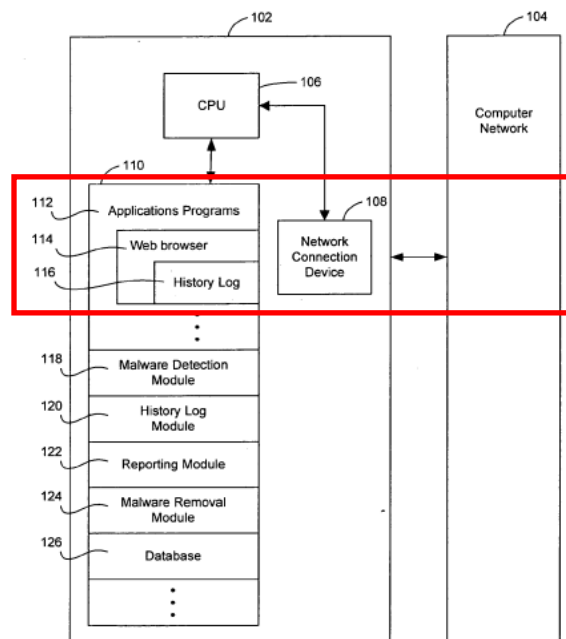


FIG. 1

Ex. 1005, FIG. 1 (annotated).

164. In another example, with reference to “additional history logs can be maintained to provide a record of other types of events, such as events related to operation of an e-mail program.” Ex. 1006, ¶ [0014]. Here, the events related to operation of the email program are the at least a portion of a recorded history. Piccard continues, “other sources of malware that can be identified include **sources that are external to the protected computer 102**, such as a **sender of an e-mail that includes the malware or an external database from which the malware was accessed or downloaded.**” Ex. 1006, ¶¶ [0031] (emphasis added). This example shows Piccard accessing a recorded history of containing the identity of an externally networked source that originated the suspected pestware object.

165. A POSITA would have been motivated to incorporate Piccard’s identification of a suspected source of malware functionality into Williamson’s heuristic engine to identify the source of the files associated with suspected malware.

[1.5] reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.

166. It is my opinion that Williamson in view of Piccard discloses each part of this limitation.

167. As discussed above, Piccard discloses identifying an identity of a particular externally networked source of a suspected pestware object using its history log

module. *Supra*, Section VIII.A.2. [1.4]. In Piccard, “the history log module 120 can identify those Web sites from which the malware may have been downloaded and, thus, can identify those Web sites as potential or suspected malware distribution sites.” Ex. 1006, ¶ [0017]. Piccard further discloses reporting the identity of the particular externally networked source. For example, “[o]nce the history log module 120 collects the information from one or more history logs, the **reporting module 122 then reports this information** to a **remotely located host computer**.” Ex. 1006, ¶ [0018] (emphasis added). Figure 2 shows Step 204 to “[r]eport that set of websites include potential malware distribution site.” Ex. 1006, FIG. 2. Piccard explains, “[t]he third operation illustrated in FIG. 2 is to report that the set of Web sites include a potential malware distribution site (block 204). In the illustrated embodiment, once the history log module identifies the set of Web sites, a reporting module (e.g., the reporting module 122) reports information relating to the set of Web sites to a remotely located host computer that is connected to the protected computer.” Ex. 1006, ¶ [0028]. Here, the “remotely located host computer” receiving the report is the externally networked pestware research entity.

168. With reference to the claimed externally networked pestware research entity, Piccard explicitly describes the host computer as “remotely located” and “included in the computer network,” thus “external to the computer.” The information relayed to this research entity “can identify the set of Web sites as potential malware

distribution sites, such as in terms of the domain names of the set of Web sites.” Ex. 1006, ¶¶ [0018], [0028]; *Supra*, Section VIII.A.2. [1.4].

169. As a specific example, Piccard explains that “the reporting module 122 can report the n most recently visited Web sites to the host computer, and **the host computer** or a user at the host computer **can evaluate those Web sites to determine whether any of those Web sites is, in fact, a malware distribution site.**” Ex. 1006, ¶ [0018] (emphasis added). This enables the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.

170. As discussed above, a POSITA would have been motivated to incorporate Piccard’s reporting module into Williamson’s heuristic engine to enable a pestware research entity to research if an externally networked source is a source of pestware. Such research could, as an example, contribute to Williamson’s “Known Good List And Known Bad List.” Ex. 1006, FIG. 2.

3. Claim 2

[2.0] The method of claim 1,

171. See [1.0]-[1.5].

[2.1] wherein the particular externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

172. As already explained, the combined Williamson/Piccard system and method discloses identification of a *reference to an identity of a particular externally*

networked source. As explained by Piccard, “by collecting the set of Web addresses from the Web browser’s history log, the history log module can facilitate **identification of the set of Web sites** from which the malware may have been downloaded.” Ex. 1006, ¶ [0026] (emphasis added); *Supra*, Section VIII.A.2. [1.4].

173. Piccard explains that a “[w]eb browser 114 records an entry that is indicative of that Web site in the history log 116. For example, when a file is accessed and downloaded from a Web site, the Web browser 114 can record a Web address of the file in the history log 116.” Ex. 1006, ¶ [0014]. The Web site is a server that is a source, which is accessed from the computer via a network. Each of these Web addresses contain information to identify a particular source is external to the claimed computer and the data may include pestware objects; therefore, the Web addresses captured by the history log are identifiers that identify particular network sources. Piccard explains that “a Web address can be a **Uniform Resource Identifier (‘URI’) of a file**, such as a **Uniform Resource Locator (‘URL’) of the file**. It is also contemplated that a Web address can be defined in various other ways, such as using an **Internet Protocol (‘IP’) address or any other identifier of a source of a file**.” Ex. 1006, ¶ [0014] (emphasis added). “[B]y collecting the set of Web addresses from the Web browser’s history log, the history log module can facilitate identification of the set of Web sites from which the malware may have been downloaded.” Ex. 1006, ¶ [0026].

174. I have been informed that in the parallel litigation, the Patent Owner asserts that “unrecited group members” may fall within the scope of this claim. Ex. 1016, pp. 25-28. Piccard explicitly describes that “[i]t is also contemplated that a Web address can be defined in various other ways, such as using an Internet Protocol (‘IP’) address or any other identifier of a source of file.” Ex. 1006, ¶ [0014]. Therefore, Piccard discloses that any other identifier identifying the externally networked source can likewise be utilized.

4. Claim 3

[3.0] The method of claim 2

175. See [1.0]-[1.5] and [2.0]-[2.1].

[3.1] wherein the identifying includes accessing an activity log that includes information that relates the activity to the suspected pestware object.

176. As I explained above with respect to [1.2], Williamson discloses identifying a process as a suspected pestware object based on evaluation by heuristic engine 212. Williamson’s behavioral approach utilizes that “most of malware of the same type exhibit similar behaviors. These behaviors are categorized and detectors for them are created and weights are automatically assigned to through a learning process.” Ex. 1005, 2:13-16. Williamson’s monitoring of after-effects of system operations cause low-level system events to be generated. In turn, the events are passed to a process context. Ex. 1005, 5:2-16. A POSITA would have understood

that events that are kept at the process context comprises an activity log as claimed.

Ex. 1005, 5:16-18.

177. Williamson explains accessing the events stored by the process context 206. In one example, “on every new event passed to the event preprocessor 210 from the process context 206 or the execution context 208, the characteristics are calculated by the heuristic engine 212 based on weights generated during training . . . and the process is determined to be either a malware or not based on the result.” Ex. 1005, 5:27-33. Here, the event preprocessor 210 accesses the events in the activity log stored by the process context 206.

178. As I have also explained, heuristic engine 212 evaluates events from the process context (via the preprocessor) for characteristics associated with particular behaviors of particular instance of a running program or executable as malware/pestware. The particular behaviors of the particular instance therefore includes information that relates the activity to the suspected pestware object because the activity of the object instance is what is evaluated.

5. Claim 4

[4.0] The method of claim 1,

179. See [1.0] - [1.5].

[4.1] wherein the recorded history resides in a browser cache, which includes information about files downloaded to the computer and a source of each of the files.

180. As explained above, the combined Williamson/Piccard system and method discloses a recorded history of externally networked sources that the computer received files from. In one example, Piccard explains that “Web browser 114 maintains a **history log 116** which serves to provide a record of Web browsing events.” Ex. 1006, ¶ [0014]; *Supra* Section VIII.A.2. [1.4].

181. The ‘243 Patent explains that the browser cache contains “objects cached in connection with use of a browser of the protected computer.” Ex. 1001, 6:27-28. This cache “may be searched to identify the name of the file, and if the originating URL is stored in connection with the file, the URL is identified as the source of the file.” Ex. 1001, 7:50-52.

182. Piccard explains that, “when a user visits a Web site, the Web browser 114 records an entry that is indicative of that Web site in the history log 116. For example, when a file is accessed and downloaded from a Web site, the Web browser 114 can record a Web address of the file in the history log 116.” Ex. 1006, ¶ [0014]. Piccard’s history log 116 records information about the files downloaded to the computer. The Web address provides the sources of each of the downloaded files. Piccard’s “Web address can be a URL of a file that is downloaded from a Web site. For example, a Web address can have the following format:

http://www.DomainName.com/Subdirectory/FileName.html, where ‘http://’ specifies a communication protocol used to download a file, ‘www.DomainName’ specifies a domain name of a Web site from which the file was downloaded, ‘/Subdirectory/’ specifies a subdirectory within the Web site from which the file was downloaded, and ‘FileName.html’ specifies a name of the file.” Ex. 1006, ¶ [0026]. Piccard’s history contains the name of files accessed by the browser, or information about the files downloaded to the computer, as well as the URL used to access them, or a source of each of the files. This history can be searched to identify the file and associated URL. This is consistent with the description of the browser cache of the ‘243 patent.

183. A POSITA would have recognized that Williamson’s system as modified by Piccard utilizes a history log in browser cache and includes a domain name as a source of a file and a file name as information about a file downloaded to the computer when the web site was accessed.

6. Claim 5

[5.0] The method of claim 1

184. See [1.0]-[1.5].

[5.1] wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

185. As explained above, Williamson in view of Piccard discloses a recorded history. In one example, Piccard explains that “Web browser 114 maintains a **history log 116**, which serves to provide a record of Web browsing events.” Ex. 1006, ¶ [0014]. *Supra*, Section VIII.A.2. [1.4].

186. Piccard explains “each of the application programs 112 maintains a separate **history log**, which serves to provide a **record of events** related to operation of that application program.” Ex. 1006, ¶ [0014] (emphasis added). Here, the record of events is an event log. In another example, Piccard explains, “Web browser 114 maintains a **history log 116**, which serves to provide a **record of Web browsing events**.” Ex. 1006, ¶ [0014] (emphasis added). here, the history log with Web browsing events is a browser history. Piccard also details browser cache as a log where recorded history resides. *Supra*, Section VIII.A.5. [4.1]. A POSITA would have understood that the example records of events are stored in a log selected from at least a browser history, an event log, and browser cache.

7. Claim 6

[6.0] A system for identifying a source of activity on a computer that is indicative of pestware including:

187. I understand that a preamble typically does not limit a claim. However, to the extent that the preamble is limiting, it is my opinion that it is disclosed.

188. Williamson discloses systems and methods for identifying sources of malware. In one embodiment, a system includes a malware detection module configured to determine that a protected computer includes malware. Ex. 1005, Abstract, 14:18-21. The block diagram of a computer system 310 suitable for use with embodiments of the present invention of Figure 3 is reproduced below, depicting Williamson's computer system. Williamson's "[c]omputer system 310 typically includes at least one processor 314," which is fundamental to the operation of the computer system. Ex. 1005, 14:20-21.

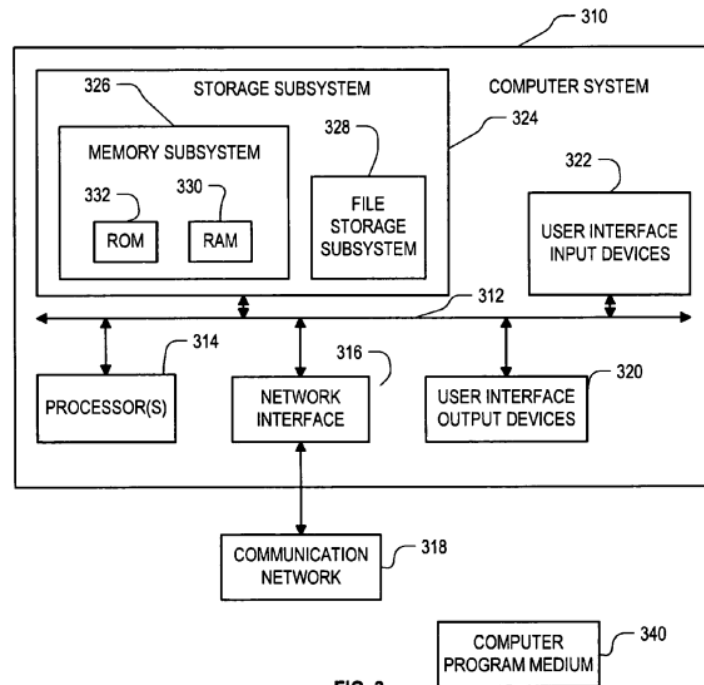


FIG. 3

Ex. 1005, FIG. 3. I also incorporate by reference my opinion regarding [1.0] above.

[6.1] an activity monitor configured to monitor API calls and to store a history of at least a portion of the API calls in an activity log, wherein the activity monitor includes a kernel-mode driver adapted to intercept the API calls;

189. It is my opinion that Williamson discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.1].

190. As discussed above with respect to claim 1, Williamson describes an activity monitor as a “*Kernel Drive* [sic] 202 and User Mode Scanners 204 that **hook various components of the operating system**. The *kernel driver* 202 looks at the requests made by programs running on the protected system, such as system calls, graphics, and windows calls.” Ex. 1005, 4:63-5:2 (emphasis added), FIG. 2. Williamson’s activity monitor includes kernel-mode driver 202 and user mode

scanners 204, which collect events. The Kernel Driver 202 hooking intercepts API calls (e.g., system calls), enabling monitoring. *Supra*, Section VIII.A.2. [1.1].

191. Williamson explains that low level system events of API calls are “kept at the process context” (*store a history of at least a portion of the API calls in an activity log*), and further that data constructs (such as the process context) are stored by storage subsystem 324. Ex. 1005, 5:16, 14:59-61, FIG. 3. The system events are stored in at least the process context 206, execution context 208, event preprocessor 210, and as preprocessed events for consumption by the heuristics engine. *Supra*, Section VIII.A.2. [1.1].

[6.2] a heuristics module configured to identify an activity on the computer that is indicative of pestware residing on the computer and to analyze the activity to determine whether one or more weighted factors associated with the activity exceeds a threshold; and

192. It is my opinion that Williamson discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.2].

193. Williamson’s heuristic engine 212 is a module that utilizes logged computer activities to determine if some activity on the computer is indicative of pestware (or not). Williamson describes that for every new event for computer activity initiated by an API call “characteristics are calculated by the heuristic engine 212 based on weights generated during training 214, based on [Evaluation Equation] (see Bayes Classifier) and the process is determined to be either a malware or not based on the

result.” Ex. 1005, 5:28-33. Accordingly, the heuristic engine 212 analyzes the activity according to the calculated characteristic using weighted factors.

194. Williamson further describes “determining an updated probability that the software application is malicious based on [a] behavior, and in response to the updated probability exceeding a threshold value (determine whether one or more weighted factors associated with the activity exceeds a threshold) . . . identifying the software application as harmful.” Ex. 1005, Claim 1. The Heuristics Engine 212 classifies activity as indicative of pestware (or not) by assessing one or more weighted factors on computer activity as preprocessed events and classifies as pestware if the predicted pestware probability exceeds some threshold. *Supra*, Section VIII.A.2. [1.2].

[6.3.a] a research portion configured to access, in response to a prompt from the heuristics module, a first set of recorded information on the computer that relates the activity to at least one file residing on the computer,

195. A POSITA would have understood that Williamson’s heuristic engine 212 as modified by Piccard to include the functions of the malware detection module, history log module, and reporting module, as described above, would include a research portion as claimed. In particular, Piccard discloses a prompt in the form of a determination of malware consistent with the definition of a prompt in the ‘243 patent. In the ‘243 patent, “the heuristics module 110 prompts the research module 108 to initiate research into the origin of the pestware initiating the activity.” Ex.

1001, 4:53-55. In Piccard, “the illustrated embodiment allows automated collection and reporting of relevant information once malware is detected on the protected computer 102.” Ex. 1006, ¶ [0020].

196. Piccard further teaches access of recorded information on the computer, “[i]f the malware detection module 118 determines that the protected computer 102 includes malware, the history log module 120 collects information from one or more history logs maintained by the application programs 112.” Ex. 1006, ¶ [0017].

197. As described above, accessed information includes “the history log module 120 can identify which one of the application programs 112 was used to access or download the malware. Here, the first set of recorded information on the computer that relates the activity to at least one file residing on the computer is the identification of the application programs (file) used to access or download the malware.” Ex. 1006, ¶ [0017]; *Supra*, Section VIII.A.2. [1.3]-[1.4].

[6.3.b] and wherein the research portion is configured to access a second set of recorded information on the computer that relates the at least one file to an externally networked source from which the file was received; and

198. Piccard discloses access by the host computer of a second set of recorded information, as claimed. In particular, Piccard teaches “the history log module 120 can collect information from one or more predetermined history logs, such as the history log 116.” Ex. 1006, ¶ [0017]. “For example, if the malware is downloaded using the Web browser 114, the history log module 120 can access the history log

116 to identify the **n most recently visited Web sites.**” Ex. 1006, ¶ [0017]. “In such manner, the history log module 120 can identify those Web sites from which the malware may have been downloaded.” Ex. 1006, ¶ [0017]. The research module 108 therefore accesses this list of most recently visited websites in the form of URLs (second set of recorded information) to identify at least one file received from an externally networked source. *Supra*, Section VIII.A.2. [1.3]-[1.4].

[6.4] a reporting portion configured to generate a report that identifies the externally networked source of the file and to report an identity of the externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the externally networked source is a source of pestware.

199. It is my opinion that Williamson in view of Piccard discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.5].

200. A POSITA would have understood that Williamson in combination with Piccard renders obvious the claimed reporting portion. Piccard discloses a **“reporting module 122”** that **reports** this “information to a remotely-located host computer that is included in the computer network 104.” Ex. 1006, ¶ [0018] (emphasis added). Here, the reporting module 122 is the claimed reporting portion, and the remotely-located host computer is the claimed externally networked pestware research entity.

201. As I have already explained, the *n* most recently visited websites described in Piccard is the identity of the externally networked source because of URL identifiers. “For example, the reporting module 122 can **report the *n* most recently visited Web sites** to the host computer.” Ex. 1006, ¶ [0018] (emphasis added). “This information as well as any additional relevant information can be analyzed at the host computer to identify whether the externally networked source is a source of pestware.” Ex. 1006, ¶ [0018]. In Piccard, the reporting module (“reporting portion”) generates a report containing a list of one or more Web addresses as URLs. These URLs contain the identity (e.g., domain name, IP address) of the externally networked source of the file downloaded by the Web browser when accessing the URL. The report is sent to the externally networked pestware research entity (“remotely-located host computer”). This host computer can then research to see if the externally networked source is a source of pestware. *Supra*, Section VIII.A.2. [1.5].

8. Claim 7

[7.0] The system of claim 6,

202. See [6.0] - [6.4].

[7.1] wherein activity monitor is configured to store an identity of each process that made each of the API calls in the stored history, and wherein the activity monitor is configured to store an identity of each file that corresponds to each the processes that made the API calls in the stored history so as to create a relation between each API call and at least one file.

203. As I explained above, Williamson describes an activity monitor as a kernel-mode driver and user mode scanners. The kernel-mode driver “looks at the requests made by programs running on the protected system, such as system calls, graphics, and windows calls.” Ex. 1005, 4:65-5:2. Williamson further explains “[w]hen these components (e.g. kernel driver) detect an interesting operation that generates a Low Level System event, the events are passed to a process content 206.” Ex. 1005, 5:5-7. “The events that are kept at the process context only apply to that particular instance of a running program or executable.” Ex. 1005, 5:16-18. Here, events that are kept at the process context are the claimed stored history. The particular instance of a process is the identity of each process that corresponds to each of the processes. The running program or executable is the identity of each file.

204. More particularly, Williamson describes two possible types of attacks – (1) malicious code in memory (i.e., memory of a running process) and (2) malicious code in an executable file. Malicious code in memory indicates that the executable file the process was launched from is normal and the process memory has been compromised, not the executable file itself. Williamson monitors and stores events with respect to the processes and to executables corresponding to the processes, and

to do so, stores identities of each process and each corresponding file. A POSITA would understand that the operating system would know the executable file associated with a process since the operating system gets process instructions from the executable file. The identity of each process and each corresponding executable file is a relation between each API call and at least one file.

9. Claim 9

[9.0] The system of claim 6,

205. See [6.0] - [6.4].

[9.1] wherein the first set of recorded information and the second set of recorded information reside in the same log file on a file storage device of the computer.

206. As I explained above, a POSITA would have understood Williamson's modified system to disclose or render obvious the first and second sets of recorded information. *Supra*, Section VIII.A.7. [6.3.a]-[6.3.b]. Piccard further teaches that the first and second sets are recorded in the same history log file 116 on a file storage device of the computer. Recall that Piccard teaches the claimed first set, "[i]dentification of which one of the application programs 112 was used to access or download the malware can be based on . . . information from one or more predetermined history logs, such as the history log 116." Ex. 1006, ¶ [0017]. Piccard teaches the claimed second set, "the history log module 120 can access the history log 116 to identify the n most recently visited Web sites." Ex. 1006, ¶ [0017].

207. Annotated Figure 1 of Piccard is copied below, which illustrates that the first set of recorded information and the second set of recorded information reside in the same log file on storage of the computer. In particular, the history log 116 is stored on “memory 110” storage device of “protected computer 102.”

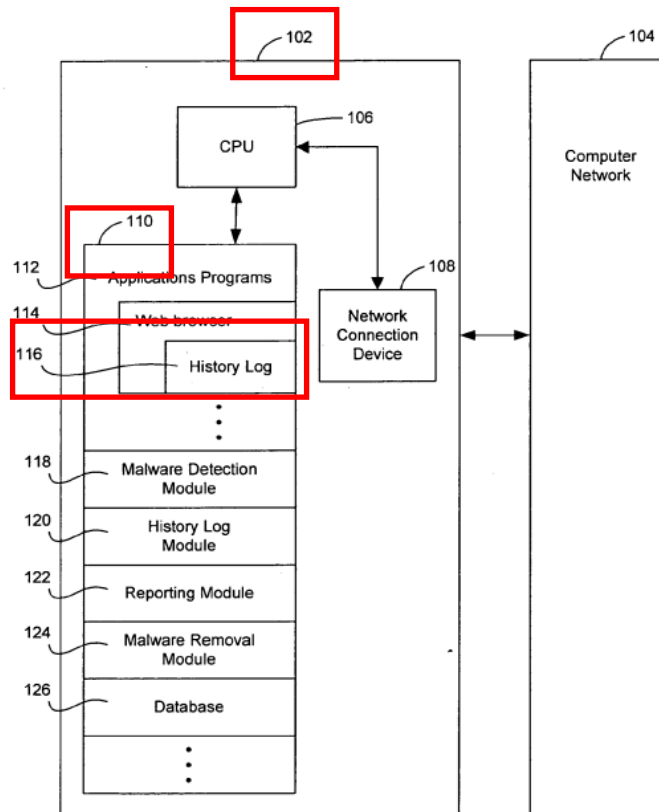


FIG. 1

Ex. 1006, FIG. 1.

10. Claim 10

[10.0] The system of claim 6,

208. *See* [6.0] - [6.4].

[10.1] wherein the externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

209. See [2.1].

11. Claim 11

[11.0] The system of claim 6

210. See [6.0] - [6.4].

[11.1] wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

211. See [5.1].

12. Claim 12

[12.0] A non-transitory computer-readable medium including processor-executable instructions for identifying an origin of activity on a computer that is indicative of pestware, the instructions including instructions for:

212. I understand that a preamble typically does not limit a claim. However, to the extent that the preamble is limiting, it is my opinion that it is disclosed.

213. Williamson in view of Piccard discloses a method of identifying an origin of activity on a computer that is indicative of pestware. *Supra*, Section VIII.A.2. [1.0].

To the extent the preamble is limiting, Williamson teaches a non-transitory computer-readable memory that includes processor-executable instructions. A POSITA would have understood that an “anti-malware program” consists of executable instructions and such a program would need to be stored. Ex. 1005, 2:27.

Williamson provides an example “[c]omputer program medium 340 can be a

medium associated with file storage subsystem 328.” Ex. 1005, 15:15-16. Further, Williamson’s “[f]ile storage subsystem 328 provides persistent storage for program and data files.” Ex. 1005, 15:3-4. A POSITA would have understood that an anti-malware program would have instructions to carry out the method of identifying an origin of activity on a computer that is indicative of pestware.

[12.1] monitoring, with a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the instructions for monitoring include instructions for monitoring API calls and storing a history of at least a portion of the API calls in an activity log;

214. It is my opinion that Williamson discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.1].

[12.2] analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware;

215. It is my opinion that Williamson discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.2].

[12.3] identifying, based upon the activity, an object residing on the computer that is a suspected pestware object;

216. It is my opinion that Williamson discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.3].

[12.4] accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from; and

217. It is my opinion that Williamson in view of Piccard discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.4].

[12.5] reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.

218. It is my opinion that Williamson in view of Piccard discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [1.5]

13. Claim 13

[13.0] The non-transitory computer-readable medium of claim 12,

219. See [12.0] - [12.5].

[13.1] wherein the particular externally networked source is identified by an identifier selected from the group consisting of an I.P. address and a URL.

220. It is my opinion that Williamson in view of Piccard discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [2.1].

14. Claim 14

[14.0] The non-transitory computer-readable medium of claim 12,

221. See [12.0] - [12.5].

[14.1] wherein the recorded history resides in at least one log selected from the group consisting of an activity log, a browser history, browser cache, browser settings, operating system settings, an event log, a debugging log, a firewall log, file information and monitoring software logs.

222. It is my opinion that Williamson in view of Piccard discloses each part of this limitation for the same reasons I expressed above with respect to claim limitation [5.1].

B. Ground 2: Claims 1-6 and 8-14 are obvious under 35 U.S.C. § 103 over Kidron in view of Williamson and Piccard

1. Reasons to Combine Kidron and Williamson

223. In my opinion, one of ordinary skill in the art would have been motivated to combine Kidron and Williamson.

224. Kidron's system of detecting and countering malicious code operates in an enterprise network system having a server and a plurality of local machines. First, local machine operations are monitored to detect irregular local behavior patterns to generate an alert from the local machine to the server. Then, if similar alerts are received from a threshold number of local machines, as determined by the server, countermeasure operations are implemented. Ex. 1007, 3:25-40. A POSITA would have understood several shortcomings of Kidron's system. First, Kidron relies on

data from multiple local machines to trigger implementation of countermeasures, so some minimum threshold of local machines must detect irregular behavior before any response. Ex. 1007, 3:29-33. This exposes local machines to malicious code, due to delays in those machines experiencing the same malicious behavior. Second, Kidron relies on a functioning server to enact countermeasures. If the server fails or a network partition isolates the server, the service will not receive alerts and countermeasures will be delayed. A local machine that is partitioned from the server cannot receive countermeasures.

225. To increase identification speed by performing it locally and provide greater resilience against server failure and network partition, a POSITA would have been motivated to incorporate Williamson's "Systems Actions," including quarantining, malware removal, etc. Ex. 1005, 4:55-58.

226. To improve local machine identification speed and accuracy, a POSITA would have been motivated to incorporate Williamson's heuristic weighting analysis for API call activity into Kidron's similar pestware security methods, which monitors API call activity to identify and mitigate pestware. Combining Williamson and Kidron's local detection provides greater opportunity for identifying malware earlier. Such earlier identification, may allow mitigation before the malware is able to spread itself, improving safety for all local machines.

227. Williamson explicitly discloses improving performance based on adjustment of the “strength” of each characteristic. Ex. 1005, 6:40-7:14. Such adjustment of the weight given to each characteristic allows greater accuracy because the influence of a characteristic on determining malware or not focus on the more predictive characteristics by increasing their weighting. As a result, a POSITA would therefore be motivated to improve Kidron’s detection performance based on weighted factors. Kidron’s pattern recognition processor 12a would incorporate functions from Williamson’s heuristic engine 212 to perform Williamson’s weighted threshold analysis. In fact, Kidron already discloses that its pattern recognition processor may be trained based on known classifications, including heuristics. Ex. 1007, 3:26-47. Williamson’s training on malware and normal programs to derive critical characteristics and associated weights would be used to enhance Kidron’s pattern recognition to detect problems earlier in the malware’s computer activity. The modified Kidron-Williamson system would therefore implement a weighted analysis using Kidron’s pattern recognition processor trained based on Williamson’s weighted factors.

228. Kidron’s pattern recognition can be further enhanced in combination with Williamson by utilizing Kidron’s alerts to improve the characteristics and weights training of Williamson. Williamson acknowledges that “more training and more diverse training is better.” Ex. 1005, 6:35-36. Kidron’s alerts contain information

about detected irregular behavior. Specifically, the “nature of the alerts” informs the selection of countermeasure by the server, and such information may also be used as (re)training materials for Williamson’s heuristics engine.

229. This combination would have improved a known system (Kidron’s system of detecting malicious code) with known techniques (Williamson’s heuristic engine implementing weights as tuning parameters), requiring only programming updates for execution on the hardware of Kidron. Ex. 1007, FIGS. 1A-1B (local machine).

230. Therefore, a POSITA would have appreciated a reasonable expectation of success making software modifications that require no more than the ordinary skill of a POSITA. This combination would include a straight-forward integration of Williamson’s training and heuristics engine as an aspect of Kidron’s pattern recognition system. In addition, alert nature could be integrated into a process of continuous update of Williamson’s characteristics and weights.

2. Reasons to Combine Kidron, Williamson, and Piccard

231. In my opinion, one of ordinary skill in the art would have been motivated to combine Piccard with the combination of Kidron and Williamson.

232. A POSITA would have found it obvious to include Piccard’s accessing of recorded history to identify an identity of a particular externally networked source, and subsequent reporting of the entity to a pestware research entity to identify a source of the malware with the combination of Kidron and Williamson.

233. Piccard describes systems and methods for identifying sources of malware similar to the combined Kidron-Williamson system and method. Ex. 1006, ¶ [0001]. If Piccard's malware detection module determines the protected computer has malware, information is collected from a history log, and this information is reported to a host computer (e.g., server) to identify a source of the malware. Ex. 1006, ¶¶ [0017]-[0018].

234. For example, one of the stated goals of the modified Kidron-Williamson system is to take appropriate countermeasures to protect a local machine, including "pinpoint the exact flow of the malicious code through the network" or "shutting down a certain TCP/IP port" in the enterprise architecture of local machines coupled to a server machine. Ex. 1007, 7:50-55.

235. A POSITA would have therefore recognized the advantages of identifying sources of malware of Piccard in a similar enterprise environment. Additionally, Kidron's countermeasures and Williamson's quarantining and removal are important elements in mitigating the spread and damage of malware. Combining this with Piccard's use of history for determination of a malware's external network source significantly enhances spread reduction as Piccard's identified external malware sites can be blocked by protected computers.

236. It is my opinion that a POSITA would have had a reasonable expectation of success in modifying Kidron-Williamson's method with Piccard because a POSITA

would have understood that Kidron's remote control core 12c would have readily integrated the functionality of Piccard's history log module 120 and the reporting module 122. Modifying the combined Kidron-Williamson system would have required a POSITA to implement only straightforward programming updates for execution on the existing hardware of Kidron-Williamson.

3. Claim 1

[1.0] A method for identifying an origin of activity on a computer that is indicative of pestware comprising:

237. I understand that a preamble typically does not limit a claim. However, to the extent the preamble is limiting, Kidron teaches "a method of detecting and countering malicious code in an enterprise network system" as one of its "flexible methodologies for identifying and countering new computer viruses and malicious code." Ex. 1007, 3:41-42, 3:19-21, 1:41-43 ("attack by malicious software, such as viruses, Trojans, worms and other malicious computer codes, is a continuous and increasing threat to computers"). *Supra*, Section VIII.A.2. [1.0] (Trojans, spyware, and adware categorized under the '243 patent definition of "pestware"). In the '243 patent, "trojans, spyware, and adware" are "collectively referred to as 'malware,' 'spyware' or 'pestware,'" so Kidron discloses identifying the activities of pestware. Ex. 1001, 1:44-45.)

238. Kidron further discloses identifying an origin of computer activity. For example, referring to suspicious operating system calls, Kidron's system and method

can “identify the source (and other details) of operating system calls.” Ex. 1007, 7:14-22.

[1.1] monitoring, using a kernel-mode driver, the computer for activity that is indicative of pestware, wherein the monitoring includes monitoring API calls and storing a history of at least a portion of the API calls in an activity log;

239. Kidron discloses a “signal monitor module *monitors* and *logs operating system call activity* (*storing a history of at least a portion of the API calls in an activity log*), innocent or malicious (*indicative of pestware*).” Ex. 1007, 6:22-23, 6:18-19 (emphasis added).

240. Here, the signal monitor module of Kidron monitors the computer operating system call activity, whether innocent or malicious (or in other words, indicative of pestware). Further, as I explained above in my discussion of API Calls, operating system calls are API calls, which applications use to request services like file/network I/O, memory allocation, process/thread creation/termination, etc. And, as I also explained in my discussion of logs and logging, the signal monitor module logging operating system calls is a storing a history of at least a portion of the API calls in an activity log.

241. Kidron further teaches “signal monitor 12b may be provided **for hooking into the operating system on the local machine**” and “may be **closely fused into the operating system**, for example, as an **extension to the operating system kernel**.” Ex. 1007, 6:15-21 (emphasis added). Consistent with the ‘243 patent, intercepting

system calls includes integration with the kernel (in other words, a kernel-mode driver with kernel mode access). Kidron explains this feature as a “signal monitor” that is a “(compact) extension to the operating system kernel” to “audit operating system calls.” Ex. 1007, 5:47-48, 6:21. Here, the signals are system (API) calls. A POSITA would understand that in order to hook operating system calls using a closely fused extension to the operating system kernel, the signal monitor 12b would need kernel mode access.

[1.2] analyzing, heuristically, computer activity to determine whether one or more weighted factors associated with an activity exceeds a threshold so as to arrive at a determination that the activity is indicative of pestware;

242. Kidron discloses heuristic analysis of computer activity so as to arrive at a determination that the activity is indicative of pestware. As explained above, Kidron’s signal monitor module monitors and logs operating system call activity. Then, “pattern recognition processor 12a periodically *analyzes* the log of **operating system calls** to **detect any irregular patterns.**” Ex. 1007, 6:24-25. In Kidron, the operating system calls are API call. As I explained earlier, API calls are an example of computer activity. Kidron analyzes the behavior pattern of activity on the local computer, looking to detect irregular patterns indicative of pestware activity, and if detected, Kidron transmits its determination as an irregular behavior alert. Ex. 1007, 3:33-50. A POSITA would have understood that in order to determine an irregular pattern, the behavior pattern must be compared to one or more regular patterns.

Derivation from a regular pattern as an irregular pattern is activity indicative of pestware.

243. As I explained above, heuristics are rules for solving a problem, such as classification of pestware or normal. Kidron describes a number of techniques for deriving and applying such rules, including rule-based analysis and heuristic scanning (i.e., analysis). Kidron discloses several examples of heuristically analyzing the computer activity. In an example, “[m]alicious code deployment of unknown nature (that is, of which there is no known signature or behavior pattern) in an enterprise environment may be detected through pattern recognition technology (such as neural nets, clustering techniques, decision tree techniques, etc.).” Ex. 1007, 4:58-62. In other examples, the methodologies may include a combination of neural net, rule-based and state analysis techniques. Ex. 1007, 5:54-56. Also, pattern recognition processor may use “integrity checking (e.g., checksum or detecting a change to a program's file size), polymorphic code detection, interrupt monitoring (monitoring of access to restricted system resources), statistic analysis, signature and/or heuristic scanning.” Ex. 1007, 6:38-42. Accordingly, Kidron’s pattern recognition technology, neural net, rule-based and state analysis techniques, and heuristic scanning are just some of the examples Kidron describes to solve its stated problem of detecting malicious code using rules.

244. Kidron does not explicitly disclose whether one or more weighted factors associated with an activity exceeds a threshold. Williamson discloses this element. *Supra*, Section VIII.A.2. [1.2]; *supra*, Section VIII.B.1. My opinion in Ground 1 regarding [1.2] is incorporated by reference.

[1.3] identifying, based upon the activity, an object residing on the computer that is a suspected pestware object;

245. Kidron's modified method teaches pattern recognition processor analysis of computer activity. *Supra*, Section VIII.A.2. [1.2]. Protective measures can then be taken in response to irregular patterns of a suspected pestware object. Ex. 1007, 6:49-51. Kidron discloses transmitting locally assessed information to the server and receiving instructions from the server for remote management of the local workstation. Here, locally assessed information is the claimed analysis based upon the activity because the locally assessed information is, for example, operating system calls. In particular, Kidron teaches detection of irregular behavior at a local machine may trigger issuance of an alert to the network, additional confirmation of irregular behavior may trigger interventional measures. Ex. 1007, 5:9-12. Here, the irregular behavior on the local machine is behavior residing on the computer, again local operating system calls in one example. Local protective (or interventional) security measures include "blocking any access to selected files, setting a quarantine on a suspect file to prevent copying of the suspect file." Ex. 1007, 6:53-58. Here in

Kidron, the “suspect file” is an example of a suspected pestware object. Viruses infect normal files, transitioning them to pestware.

[1.4] accessing, in response to the identifying an object, at least a portion of a recorded history of externally networked sources that the computer received files from so as to identify a reference to an identity of a particular externally networked source that the suspected pestware object originated from; and

246. As I explained in Ground 1, Williamson in view of Piccard teaches this element. *Supra*, VIII.A.2. [1.4]; *supra*, Section VIII.B.2. My opinion in Ground 1 regarding [1.4] is incorporated by reference.

[1.5] reporting the identity of the particular externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the particular externally networked source is a source of pestware.

247. As explained in Ground 1, Williamson in view of Piccard teaches this element. *Supra*, Section VIII.A.2. [1.5]; *supra*, Section VIII.B.2. My opinion in Ground 1 regarding [1.5] is incorporated by reference.

4. Claim 3

[3.0] The method of claim 2

248. See Ground 2 [1.0]-[1.5] and [2.0]-[2.1]. *See also, supra*, Section VIII.A.3. [2.0]-[2.1].

[3.1] wherein the identifying includes accessing an activity log that includes information that relates the activity to the suspected pestware object.

249. As I have explained, Kidron’s modified method discloses a signal monitor that monitors and logs operating system call activity in an activity log. *Supra*,

VIII.B.3. [1.1]. Kidron further teaches that operating system calls are accessed by pattern recognition processor's analysis of the log of operating system calls to detect any irregular patterns, which may correspond to, for example, known or unknown viruses, Trojans, worms, or other malicious code (for example, with characteristics of memory resident, file, and/or macro viruses). Ex. 1007, 6:23-31. The computer activity from the activity log detected as an irregular pattern are, in fact, API calls made by the pestware objects (e.g., virus, Trojan, worm, or malicious code), thereby relating the activity to the suspected pestware object.

5. Claim 6

[6.0] A system for identifying a source of activity on a computer that is indicative of pestware including:

250. I understand that a preamble typically does not limit a claim. However, to the extent that the preamble is limiting, it is my opinion that it is disclosed. As I have explained, Kidron discloses systems for identifying sources of malware. For example, Kidron provides a system for detecting and countering known and unknown malicious code in an enterprise network. Ex. 1007, 3:25-27. Kidron further explains its advantages of new, more flexible methodologies for identifying and countering new computer viruses and malicious code, and that the system includes a local machine of "any of the known computing platforms." Ex. 1007, 7:31-35; *see*. Ex. 1007, 3:19-21; *supra*, Section VIII.B.3. [1.0]. My opinion in Ground 2 regarding [1.0] is incorporated by reference.

[6.1] an activity monitor configured to monitor API calls and to store a history of at least a portion of the API calls in an activity log, wherein the activity monitor includes a kernel-mode driver adapted to intercept the API calls;

251. As I’ve explained with respect to claim 1, Kidron describes a signal monitor that *monitors* and *logs operating system call activity*. Again, as I have already explained, the signal monitor is the claimed activity monitor, and the functionality of logging operating system call activity is storing a history of at least a portion of the API calls in an activity log, as claimed.

252. Kidron also explains “signal monitor 12b may be provided **for hooking into the operating system on the local machine**” and “may be **closely fused into the operating system**, for example, as an **extension to the operating system kernel**.”

Ex. 1007, 6:15-21 (emphasis added). Hooking is a well-known method of intercepting function calls or messages or events passed between software components. Further, as I have already explained, Kidron’s closely fused extension to the operating system kernel is the claimed kernel-mode driver. *Supra*, Section VIII.B.3. [1.1]. My opinion in Ground 2 regarding [1.1] is incorporated by reference.

[6.2] a heuristics module configured to identify an activity on the computer that is indicative of pestware residing on the computer and to analyze the activity to determine whether one or more weighted factors associated with the activity exceeds a threshold; and

253. As I explained above, Kidron’s signal monitor module monitors and logs operating system call activity. Then, “**pattern recognition processor 12a**

periodically **analyzes the log of operating system calls** to detect any irregular patterns.” Ex. 1007, 6:24-25 (emphasis added). The pattern recognition processor is a heuristics module as it can apply “heuristic scanning” and may be “rules-based” to analyze the activity on the computer in the form of the log of operating system calls.

254. Irregular behavior patterns may correspond to, for example, known or unknown viruses, Trojans, worms, or other malicious code (for example, with characteristics of memory resident, file, and/or macro viruses). Ex. 1007, 6:28-31. And that such detection of irregular behavior is at a local machine. Ex. 1007, 5:9-12. Here, the activity on the “local machine” (e.g., computer) is used by the pattern recognition processor to detect “irregular patterns” in “behavior” (i.e., computer activity) to identify activities that are indicative of pestware.

255. Kidron further discloses many examples of heuristics analysis by the signal monitor module. *Supra*, Section VIII.B.3. [1.2]. Kidron does not explicitly disclose whether one or more weighted factors associated with an activity exceeds a threshold. Williamson discloses this element. *Supra*, Section VIII.A.2. [1.2]; *supra*, Section VIII.B.1. My opinion in Ground 2 regarding [1.2] is incorporated by reference.

[6.3.a] a research portion configured to access, in response to a prompt from the heuristics module, a first set of recorded information on the computer that relates the activity to at least one file residing on the computer,

256. As I explained in Ground 1, Piccard teaches this element. A POSITA would have understood that Kidron's modified remote control core 12c would include the functionality of Piccard's history log module 120 and reporting module 122. The pattern recognition processor modified by Williamson's weighted evaluation (heuristics module) prompts the research portion (i.e., part of system identifying the activity indicative of pestware). The modified remote control core 12c uses the functionality of the history log module 120 to collect information from one or more history logs. Ex. 1006, ¶ [0018]; *Supra*, Section VIII.A.7. [6.3.a]; *supra*, Section VIII.B.2. My opinion in Ground 1 regarding [6.3.a] is incorporated by reference.

[6.3.b] and wherein the research portion is configured to access a second set of recorded information on the computer that relates the at least one file to an externally networked source from which the file was received; and

257. As explained in Ground 1, Piccard teaches this element. *Supra*, Section VIII.A.7. [6.3.b]; *supra*, Section VIII.B.2. My opinion in Ground 1 regarding [6.3.b] is incorporated by reference.

[6.4] a reporting portion configured to generate a report that identifies the externally networked source of the file and to report an identity of the externally networked source to an externally networked pestware research entity so as to enable the externally networked pestware research entity to research whether the externally networked source is a source of pestware.

258. As explained in Ground 1, Piccard teaches this element. *Supra*, Section VIII.A.7. [6.4]; *supra*, Section VIII.B.2. My opinion in Ground 1 regarding [6.4] is incorporated by reference.

6. Claim 8

[8.0] The system of claim 6,

259. See Ground 2, [6.0]-[6.4].

[8.1] wherein the heuristics module is configured to receive information about the API calls from the activity monitor and determine whether the API calls are indicative of pestware.

260. As I have explained above, Kidron's modified system discloses a pattern recognition processor (*heuristics module*) configured to receive information about operating system calls (*API calls*) from signal monitor (*activity monitor*) as a log of operating system calls. *Supra*, Section VIII.B.5. [6.1]-[6.2]. Kidron further teaches "pattern recognition processor 12a periodically analyzes the log of operating system calls to detect any irregular patterns." Ex. 1007, 6:24-25. Here Kidron's operating system calls (i.e., API calls) are monitored by the signal monitor, which shares this information with the pattern recognition processor (i.e., heuristics module).

261. Irregular behavior patterns may correspond to, for example, known or unknown viruses. Ex. 1007, 6:28-29. The pattern recognition processor analyzes the API calls for irregular patterns to determine if the calls indicate the behavior of pestware.

7. Claim 12

[12.0] A non-transitory computer-readable medium including processor-executable instructions for identifying an origin of activity on a computer that is indicative of pestware, the instructions including instructions for:

262. I understand that a preamble typically does not limit a claim. However, to the extent that the preamble is limiting, it is my opinion that it is disclosed. Kidron-Williamson-Piccard discloses a method of identifying an origin of activity on a computer that is indicative of pestware. *Supra*, Section VIII.B.3. [1.0]. To the extent the preamble is limiting, Kidron teaches instructions for the method of identifying stored on a non-transitory computer-readable medium. A POSITA would have understood that a system “detecting and countering malicious code in an enterprise network system” would consist of executable instructions for execution on a “plurality of local machines” and that such instructions would need to be stored. In one example, Kidron discusses “tools may be embodied in one or more computer programs stored on a computer readable medium” for “detection of malicious code.” Ex. 1007, 4:18-24, 5:43-45, 7:31-35, Claim 24).

[12.1]-[12.5]

263. See Ground 2, [1.1]-[1.5]. My opinion in Ground 2 regarding claim 1 is incorporated by reference.

8. Claims 2, 4-5, 9-11, 13-14

264. My opinion from Ground 1 regarding these claims is incorporated by reference, and in particular a method, system, and non-transitory computer-readable medium of Kidron modified by Williamson and Piccard. *Supra*, Section VIII.A.

IX. NO OBJECTIVE INDICIA OF NONOBVIOUSNESS

265. I am not aware of any objective indicia of non-obviousness related specifically to the '243 patent. Indeed, I am not aware of any evidence of commercial success, long felt but unsolved need, failure of others, industry praise, or unexpected results specifically related to the '243 patent.

X. CONCLUSION

266. For the reasons set forth above, it is my opinion that claims 1-14 of the '243 patent are invalid in view of the prior art. In signing this declaration, I understand that the declaration will be filed as evidence in a contested case before the Patent Trial and Appeal Board of the United States Patent and Trademark Office. I acknowledge that I may be subject to cross-examination in this case and that cross-examination will take place within the United States. If cross-examination is

required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

267. I declare that all statements made herein of my knowledge are true, that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1007 of Title 18 of the United States Code.

Executed: December 8, 2023

By: Michael J. Donahoo

Michael J. Donahoo