

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TREND MICRO, INC.

Petitioner

v.

OPEN TEXT INC.

Patent Owner

Case IPR2023-00655

Patent No. 8,418,250

**DECLARATION OF PROFESSOR NENAD MEDVIDOVIC, Ph.D.
IN SUPPORT OF PATENT OWNER'S PRELIMINARY RESPONSE**

TABLE OF CONTENTS

I.	QUALIFICATIONS	2
II.	UNDERSTANDING OF RELEVANT LEGAL PRINCIPLES	7
III.	STATE OF THE PRIOR ART	9
IV.	The '250 Patent.....	11
	A. Summary Of The '250 Patent.....	11
	B. Claim Construction.....	18
	C. Level Of Ordinary Skill In The Art.....	19
V.	OVERVIEW OF PRIOR ART	21
	A. Overview Of Halperin	21
	B. Overview Of Mankins	25
	C. Overview Of Inoue.....	27
VI.	HALPERIN IN VIEW OF MANKINS DOES NOT DISCLOSE THE CHALLENGED CLAIMS	30
	A. There Is No Motivation To Combine Halperin With Mankins.....	31
	1. Halperin discourages systems like Mankins's.....	31
	2. Halperin and Mankins are not complementary systems	34
	3. Combining Halperin with Mankins would diminish accuracy instead of improving upon false positives.	36
	B. There Is No Motivation To Combine Halperin With Inoue.....	40
	C. Independent Claims 1, 13, and 25	43
	1. Halperin Does Not Disclose Limitations 1[b] to 1[d], 13[b] to 13[d], or 25[b] and 25[c].....	44

2.	The Petition’s Combinations of Halperin in View of Mankins and Halperin in View of Inoue Do Not Render Obvious Limitations 1[f] and 24[e]	55
VII.	CONCLUSION.....	65

EXHIBIT LIST

Exhibit	Description
2001	Declaration of Professor Nenad Medvidovic, Ph.D
2002	<i>Curriculum Vitae</i> of Professor Nenad Medvidovic, Ph.D.
2003	<i>Sonrai Memory Ltd. v. Kingston Tech. Co. and Kingston Tech. Corp.</i> , Case No. 6:21-cv-1284-ADA, Order Denying Kingston Tech. Co. and Kingston Tech. Corp.’s Opposed Motion to Stay Pending Resolution of <i>Inter Partes</i> Review, Dkt. 94 (W.D. Tex., Oct. 18, 2022)
2004	<i>mCom IP, LLC v. Cisco Systems, Inc.</i> , Case No. 6:22-cv-00261-ADA, Order Denying Defendant’s Motion to Stay Pending <i>Inter Partes</i> Review, Dkt. 42 (W.D. Tex., Oct. 20, 2022)
2005	<i>Intellectual Ventures I LLC et al. v. Hewlett Packard Enterprise Co.</i> , No. 6:21-cv-00596-ADA, Order Denying Hewlett Packard Enterprise Company’s Opposed Motion to Stay Pending Resolution of <i>Inter Partes</i> Review, Dkt. 104 (W.D. Tex., Dec. 22, 2022)
2006	Dani Kass, “Catching Up On Patent Litigation With Judge Albright” (Law360 Mar. 14, 2023); https://www.law360.com/pulse/articles/1582438/print?section=pulse/courts
2007	U.S. District Courts—Federal Court Management Statistics—Profiles—During the 12-Month Periods Ending March 31, 2018 Through 2023, “Table N/A—U.S. District Courts—Combined Civil and Criminal Federal Court Management Statistics (March 31, 2023)”, <i>available</i> https://www.uscourts.gov/sites/default/files/data_tables/fcms_na_distprofile0331.2023.pdf

Exhibit	Description
2008	<i>Webroot, Inc., Open Text Inc., v. AO Kaspersky Lab et al.</i> , Case No. 6:22-cv-00243-ADA-DTG, Order Granting Third Amended Scheduling Order, Dkt. 142 (W.D. Tex., Dec. 27, 2022)
2009	<i>Webroot, Inc. and Open Text Inc., v. AO Kaspersky Lab. et al.</i> , Case No. 6:22-cv-243-ADA-DTG, Claim Construction Order, Dkt. 236 (W.D. Tex. Mar. 16, 2023)
2010	<i>Webroot, Inc. and Open Text Inc., v. Trend Micro, Inc.</i> , Case No. 6:22-cv-239-ADA-DTG, First Amended Complaint For Patent Infringement, Dkt. 42 (W.D. Tex. June 23, 2023)
2011	<i>Webroot, Inc. and Open Text Inc., v. AO Kaspersky Lab. et al.</i> , Case No. 6:22-cv-243-ADA-DTG, Scheduling Order, Dkt. 53 (W.D. Tex. Sept. 7, 2022)
2012	<i>Webroot, Inc. and Open Text Inc., v. Trend Micro, Inc.</i> , Case No. 6:22-cv-00239-ADA-DTG, Defendant’s Preliminary Invalidity Contentions (W.D. Tex., Sept. 13, 2022) (excerpted)
2013	<i>Webroot, Inc. and Open Text Inc., v. CrowdStrike, Inc. and CrowdStrike Holdings, Inc.</i> , Case No. 6:22-cv-00241-ADA-DTG, Defendant’s Preliminary Invalidity Contentions (W.D. Tex., Sept. 13, 2022) (excerpted)
2014	<i>Webroot, Inc. and Open Text Inc., v. AO Kaspersky Lab</i> , Case No. 6:22-cv-00243-ADA-DTG, Defendant’s Preliminary Invalidity Contentions Appendix A (W.D. Tex., Sept. 14, 2022) (excerpted)

I, Professor Nenad Medvidovic, Ph.D., declare as follows:

1. I have been engaged by King & Spalding LLP on behalf of OpenText Inc. (“Patent Owner”), to provide this Declaration concerning the technical subject matter relevant to Petitioner Trend Micro’s petition for Inter Partes Review (“Petition”) of U.S. Patent 8,418,250 (“the ’250 patent”).

2. I am over 18 years of age. I have personal knowledge of the facts stated in this Declaration and could testify competently to them if asked to do so.

3. I have reviewed and am familiar with the specification and the claims of the ’250 patent. In general, I will cite to the specification of a United States patent using the following formats: (Exhibit No. Col.:Line Number(s)) or (Exhibit No. Paragraph Number(s)). For example, the citation (EX-1001 1:1–10) points to the ’250 patent specification at column 1, lines 1–10. Also, for convenience, I use italics to denote limitations from the challenged claims.

4. All the opinions in this Declaration are based on the documents I reviewed and my knowledge and professional judgment. In forming the opinions expressed in this Declaration, I reviewed the Petition for *Inter Partes* Review of U.S. Patent No. 8,418,250 and supporting exhibits.

5. I have been asked to provide my technical opinions regarding how a person of ordinary skill in the art would have understood the claims of the ’250 patent at the time of the alleged invention. I have also been asked to provide my

technical opinions on how concepts in the '250 patent specification relate to claim limitations of the '250 patent. In reaching the opinions provided herein, I have considered the '250 patent, its prosecution history, and the references cited above and have drawn as appropriate on my own education, training, research, knowledge, and personal and professional experience.

I. QUALIFICATIONS

6. In formulating my opinions, I have relied on my knowledge, training, and experience in the relevant filed, which I will summarize briefly. A more detailed summary of my background, experience, and publications is set forth in my curriculum vitae (CV), which is submitted as Exhibit 2002.

7. I received a Bachelor of Science degree, *summa cum laude*, from Arizona State University's Computer Science and Engineering department.

8. I received a Master of Science degree from the University of California at Irvine's Information and Computer Science department.

9. I received a Doctor of Philosophy degree from the University of California at Irvine's Information and Computer Science department. My dissertation was entitled "Architecture-Based Specification-Time Software Evolution."

10. For nearly thirty years, my research has focused on the building blocks of software systems: software components (computational and data elements) and

connectors (interaction-facilitating elements), and mechanisms for their implementation. In particular, I have focused on middleware facilities and software frameworks. In my Ph.D. dissertation—whose first research results were published in 1995 while the dissertation was completed in 1999—yielded a new design style for the development and evolution of highly distributed software systems built out of independent components that exchange different types of information encoded in messages in order to accomplish specific goals. This research focused on the design principles behind those systems, the appropriate support for modeling the systems (both at the level of individual agents as well as their integrations into subsystems and entire systems), and the infrastructure for their implementation, deployment, and optimization. This work was applied across a wide range of domains, from traditional distributed systems to mobile devices and autonomous robots.

11. I am employed by the University of Southern California (“USC”) as a faculty member in the Computer Science Department and have been since January 1999. I am a tenured professor. Since July 2022, I have served as my Department’s Chair. Between January 2009 and January 2013, I served as the Director of USC’s Center for Systems and Software Engineering. Between July 2011 and July 2015, I served as my Department’s Associate Chair for Ph.D. Affairs. Between July 2021 and June 2022, I served as my Department’s Vice Chair for Tenure Affairs.

12. I teach graduate and undergraduate courses in software architecture, software engineering, and embedded systems. and I advise Ph.D. students. I have graduated nineteen Ph.D. students and advise seven students currently pursuing a Ph.D.

13. I served as Program Co-Chair for the flagship conference in my field—International Conference on Software Engineering (“ICSE”)—held in May 2011. I have served as Chair or Co-Chair for various other conferences in the Software Engineering field, including: the Fifth Working Institute of Electrical and Electronics Engineers (“IEEE”)/IFIP Conference on Software Architecture, the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems, the Fifteenth International Association for Computing Machinery’s (“ACM”) Special Interest Group on Software Engineering (“SIGSOFT”) Symposium on Component Based Software Engineering, the IEEE/CSSE/ISE Workshop on Software Architecture Challenges for the 21st Century, and the Doctoral Symposia at the Sixteenth ACM SIGSOFT International Symposium on the Foundations of Software Engineering and at the 44th International Conference on Software Engineering.

14. I serve or have served as an editor of several peer-reviewed journals, including “IEEE Transactions on Software Engineering,” “ACM Transactions on Software Engineering and Methodology,” “Journal of Software Engineering for

Robotics,” “Elsevier Information and Software Technology Journal,” “Journal of Systems and Software,” “Journal of Software Engineering Research and Development,” and “Springer Computing Journal.” Additionally, I have served as a guest editor of several special issues for other journals. Since January 2018, I have served as Editor-in-Chief of “IEEE Transactions on Software Engineering.”

15. Between September 2013 and September 2015, I served as Chair of the ICSE Steering Committee. Both prior to that (2008-2013) and subsequent to that (2015-present) I have been a member of the ICSE Steering committee. I am currently a member of the Steering Committee of the joint European Conference on Software Engineering and the Symposium on the Foundations of Software Engineering (“ESEC/FSE”). From July 2015 to September 2017, I chaired the Steering Committee of FSE, while it was a standalone symposium. I previously served as a member of the Steering Committees of the Working IEEE/IFIP Conference on Software Architecture.

16. Between July 2015 and June 2018, I served as Chair of the ACM SIGSOFT, the largest professional organization in my field. During the past year, I have chaired the committee in charge of ACM’s Software System Award, one of the most important technical awards given in my field.

17. I co-authored *Software Architecture: Foundations, Theory, and Practice*, a widely used textbook in the field of software systems architecture.

18. I have served as editor of various books in the Software Engineering field, including *Proceedings of the 3rd International Conference on Self-Adaptive and Self-Organizing Systems*, *Proceedings of the Warm-Up Workshop for the 32nd International Conference on Software Engineering*, *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, and *40 Editions of ICSE—the Ruby Anniversary Celebration*.”

19. I have authored or co-authored nearly 300 papers in the Software Engineering field. My most cited paper has been cited well over 3,000 times. A paper I co-authored in the 1998 International Conference on Software Engineering, my field’s flagship conference, was given ten years later, in 2008, that conference’s Most Influential Paper Award. Another of my publications received a Most Influential Paper Award in 2020. Several of my other publications have received “best paper” or “most cited” awards.”

20. Several of my publications have focused on different aspects of computer security. A paper describing a security-vulnerability detection tool for Android devices, built in my lab, won the Best Tool Paper award at the 2017 Automated Software Engineering Conference.

21. I have co-invented three patents in the area of cybersecurity.

22. I have been named a Distinguished Scientist of the ACM. I have been elected a Fellow of the IEEE. Fellow of the IEEE is IEEE's highest grade; less than 0.1% of its membership earns this grade each year.

II. UNDERSTANDING OF RELEVANT LEGAL PRINCIPLES

23. I am not a lawyer, and I will not provide any legal opinions. Although I am not a lawyer, I have been advised that technical experts apply certain legal standards when they form opinions about the meaning and validity of patent claims.

24. I understand that a patent claim is invalid if it is anticipated or obvious in view of the prior art, and that a claim can be unpatentable even if all of the requirements of the claim cannot be found in a single prior-art reference. I further understand that invalidity of a claim requires that the claim be anticipated or obvious from the perspective of a person of ordinary skill in the art at the time the invention was made.

25. I have been informed that a patent claim is invalid if it would have been obvious to a person of ordinary skill in the art. In analyzing the obviousness of a claim, I understand the following factors may be taken into account: (1) the scope and content of the prior art; (2) the differences between the prior art and the claims; (3) the level of ordinary skill in the art; and (4) any so called "secondary considerations" of nonobviousness if they are present.

26. I have been informed that a claim can be obvious in light of a single prior art reference or multiple prior art references. I understand that to prove that prior art or a combination of prior art renders a patent obvious, it is necessary to:

- 1) Identify the particular references that, singly or in combination, make the patent obvious;
- 2) Specifically identify which elements of the patent claim appear in each of the asserted references; and
- 3) Explain why a person of ordinary skill in the art would have combined the references, and how they would have done so, to create the inventions claimed in the patent. I further understand that exemplary rationales that may support a conclusion of obviousness include:
 - Combining prior art elements according to known methods to yield predictable results;
 - Simple substitution of a known element for another to obtain predictable results;
 - Use of known technique(s) to improve similar devices (methods or products) in the same way;
 - Applying a known technique to a known device (method or product) ready for improvement to yield predictable results;

- Choosing from a finite number of identified, predictable solutions with a reasonable expectation of success, which I am informed is the “obvious to try” standard;
- Known work in one field of endeavor may prompt variations of the work for use in either the same field or a different field based on design incentives or other market forces if the variations are predicable to a person of ordinary skill in the art; and
- Some teaching, suggestion, or motivation in the prior art that would have led a person of ordinary skill in the art to modify the prior art reference or to combine the prior art reference teachings to arrive at the claimed invention.

27. I have been informed that, in considering obviousness, hindsight reasoning derived from the patent at issue may not be used.

III. STATE OF THE PRIOR ART

28. “Malware” generally refers to an executable computer file or an object that consists of or contains malicious code. Viruses, trojans, worms, and similar programs are all types of malware. (EX-1001 1:14-17.) Different types of malware work differently, and so what works to detect one type of malware (*e.g.*, a virus) may be less effective against another version (*e.g.*, a worm). (*See generally* EX-1001 1:14-2:56.)

29. At the time the invention in the '250 patent was invented, many systems designed to address the threat of malware focused on looking for previously known malware. For example, antivirus software might examine executable computer files for known signatures indicative of malware. (*E.g.*, EX-1001 1:18-29.) “Generally, the method of dealing with malware is that when new types of malware are released, for example via the Internet, these are eventually detected.” (EX-1001 1:22-24.) Providers of antivirus software push an update to their software intended to recognize and handle known malware. (EX-1001 1:24-28.) However, these types of systems suffered from inherent delay. (EX-1001 1:30-37.) A system would be vulnerable after release of a new piece of malware until such malware was detected and a solution generated and released to users. (*Id.*)

30. Other anti-malware systems sought to protect from previously unknown malware. For example, local area networks were known to implement a centralized computer responsible for authorizing the running of files on remote computers within the network. (EX-1001 1:60-67.) When a remote computer encountered a file that was unknown to it, it would forward a request to the central computer. (EX-1001 1:64-2:4.) But when the file was unknown even to the central computer, the central computer would obtain the whole file from the remote computer so it could to determine whether it should be regarded as safe or malware. (EX-1001 2:4-7.) Such an analysis had to be “carried out manually or semi-

manually”—that is, not by the computer but by a human—by subjecting the file to a detailed analysis. (EX-1001 2:8-11.)

31. These analysis methods included, for example, emulation or interpretation, “which can still take days given the human involvement that is typically required.” (EX-1001 2:4-11.) Other prior art systems similarly relied on a human system administrator to “make a better informed though still ‘manual’ decision as to whether or not the file is safe to run.” (*E.g.*, EX-1001 2:18-27.) “There is therefore still a considerable period of time before a new file is classified as safe or as malware.” (EX-1001 2:12-13.)

32. Thus, prior art systems “rel[ied] on deep analysis of a new object in order to determine whether or not the object is malicious, which introduces delay and therefore risk to users during the period that the file is analysed and new anti-malware signatures distributed[.]” (EX-1001 2:57-61.)

33. Accordingly, there was a long-felt need for an improved technique to efficiently determine whether objects were safe or malware.

IV. THE ’250 PATENT

A. Summary Of The ’250 Patent

34. The ’250 Patent is entitled “Methods and Apparatus for Dealing with Malware” and describes and claims a method and system for classifying a computer object as malware. (*E.g.*, EX-1001 Claims 1, 13, 25.) The ’250 patent describes a

computer “object” as including a computer file, partial file, sub-program, macro, web page, any piece of code to be operated by or on the computer. (EX-1001 7:63-66.)

35. Unlike the methods described in the prior art, the '250 Patent describes a rapid determination of whether an object (according to the definition in ¶ 34) should be classified as safe or malware, without requiring a preliminary detailed analysis of the object. (EX-1001 3:8-18.)

36. The '250 patent teaches a method that may be implemented in a distributed network, such as a local area network, having a “central” or “base” computer connected to multiple “remote” computers. (EX-1001 7:54-59, Fig. 1.)

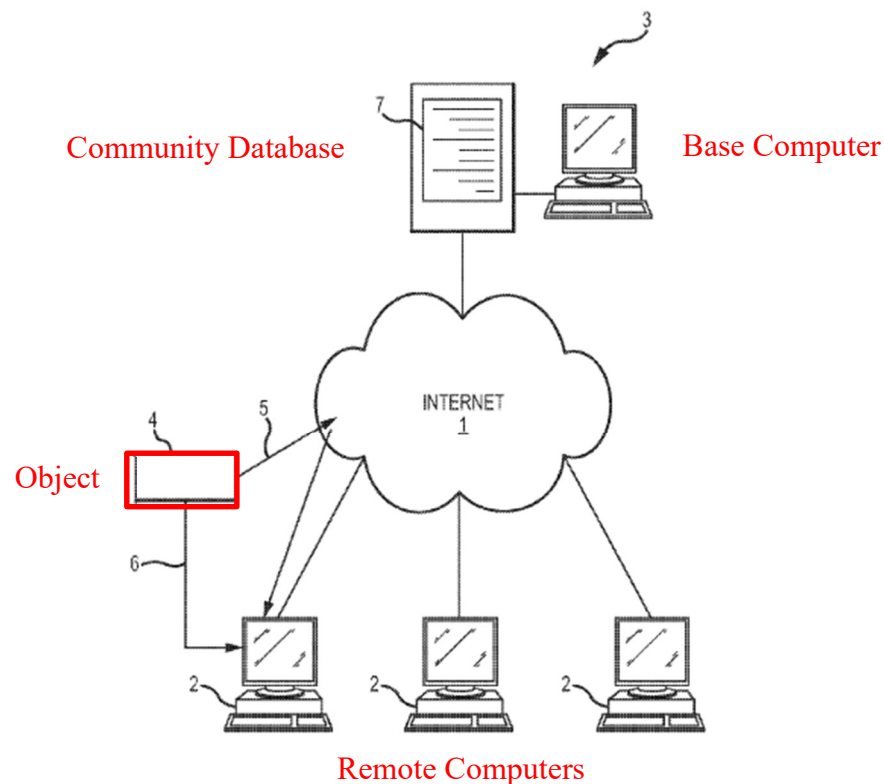


FIG.1

37. (EX-1001, Fig. 1, 7:54-8:16 (annotated).)

38. The '250 patent teaches that the remote computers provide data about an object to the base computer to facilitate a comparison of data at the base computer. (EX-1001 4:43-47.) Such data may include executable instructions within the object; the size of the object; the current name of the object; the physical and folder location of the object; the original name of the object; the creation and modification dates of the object; vendor, product and version and any other information stored within the object; the object header or header held by the remote computer; and events initiated by or involving the object. (EX-1001 4:52-61.)

39. One of the ways that this data may be provided to the base computer is in the form of a “key,” which the '250 patent explains allows for a quick comparison of objects and faster analysis at the base computer. (EX-1001 3:51-4:6.) “The key preferably has at least one component that represents executable instructions contained within or constituted by the object.” (EX-1001 3:51-53.) This “important preferred feature” allows a comparison to be made at the base computer of only the executable instructions of the object. (EX-1001 3:53-55.) The '250 patent explains that when objects with different names have the same executable instructions, this often indicates the objects are malware. (EX-1001 3:55-59.) The use of a key representing the object’s executable instructions means that the invention according to the '250 patent treats these differently named but substantively similar objects as

the same object for the purpose of determining whether they are safe or malware.
(*Id.*)

40. Figure 2 is described as “a flowchart showing schematically the operation of an example of” the ’250 patent’s method. (EX-1001 7:46-48.)

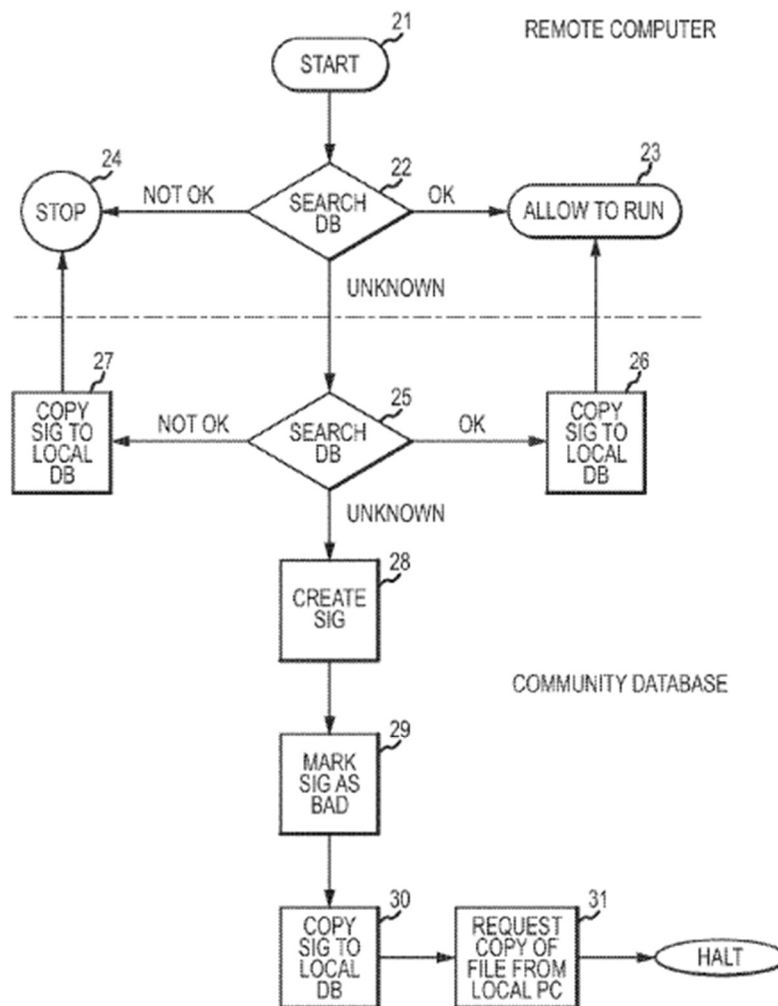


FIG.2

41. (EX-1001 Fig. 2, 8:16-15:5 (annotated). I describe the steps illustrated in Figure 2, below.

42. Starting at step 21, a computer object runs on a remote computer. (EX-1001 8:17-18.) At step 22, the remote computer searches its local database for a key representing that particular process, along with any related objects or events. (EX-1001 8:18-24, Fig. 2 (step 22).) If the process is known and a local key exists, the remote computer allows or stops the process based on whether the key indicates that the process is safe or unsafe. (EX-1001 8:24-34, Fig. 2 (steps 23-24).)

43. When a search of the local database does not reveal the identity of the object, data about the object is passed along to the base computer. (EX-1001 8:45-47.) (This is illustrated between steps 22 and 25, as the process bridges the divide between the remote computer and the base computer. EX-1001 Fig. 2.) At step 25, the base computer searches the community database for the object's signature. (EX-1001 8:47-50, Fig. 2 (step 25).) If the base computer finds the object's signature in the community database, it may indicate to the remote computer that the object is safe (EX-1001 8:60-9:30, Fig. 2 (step 26)) or that it is unsafe (EX 1001 9:4-7, Fig. 2 (step 27)). This indication may determine whether the process is stopped or permitted to run. (EX-1001 Fig. 2, (steps 23-24, 26-27).) In either instance, the remote computer's local database may be updated with this additional information, so that the remote computer has this information immediately at hand the next time that computer encounters the object. (EX-1001 8:60-9:10.)

44. If the object remains unknown even after searching the community database, the '250 patent teaches creating a signature for the new object, and the object is initially marked as bad or unsafe. (EX-1001 9:11-17, Fig. 2 (steps 28-29).) Once the signature has been created, the '250 patent's invention provides it to the remote computer to copy to its local database, along with a message that might instruct the remote computer not to run the object or might inform the user of the risk and offer the choice of running it; additionally, the community database may request a copy of the object itself, in case further analysis is needed. (EX-1001 9:17-25, Fig. 2 (steps 30-31).)

45. If the unknown object is allowed to run on the remote computer, the '250 patent teaches creating a "mask" to monitor the object's actions. (EX-1001 9:26-39, 14:55-61 ("The signature of an object may comprise or be associated with a mask.")) For example, the mask parameters might indicate "the expected behaviour of the object; any external requests or Internet connections that that object might legitimately have to make or call upon the remote computer 2 to make, including details of any ports or interfaces that might be required to be opened to allow such communication, any databases, either local or over a local area network or wide area network or Internet, that may be expected to be interrogated by that object; and so on." (EX-1001 14:61-15:4.)

46. In addition to monitoring processes that are not pre-authorized, the '250 Patent discloses an embodiment in which even pre-authorized objects are monitored while running to determine whether those objects exceed the parameters of the existing mask. (EX-1001 15:42-45, Fig 3 (steps 35 and 36).)

47. If the object's actions remain within the boundaries set by the mask, then the object is allowed to run. (EX-1001 15:45-47, Fig. 3 (step 34).) But if the object's actions extend outside the mask—if, for example, the object attempts to initiate connections that have not been pre-authorized—then the behavior is flagged at an alert step and the '250 patent teaches that subsequent action may be taken. (EX-1001 15:47-51, Fig. 3 (step 37).)

48. The object undergoes continuous monitoring, which allows the mask to be updated when current or future object behavior varies from prior object behavior. (EX-1001 15:64-16:3.) For example, when an object exceeds the mask's initial parameters but no ill effects follow, or if the system recognizes the behavior from other objects and knows that it is safe, then the '250 Patent teaches modifying the mask to include the new behavior as a safe or allowed behavior. (EX-1001 15:9-21.) Continuous monitoring allows for similar reclassification of the object as unsafe if the object violates the parameters of the generated mask by attempting to connect to a known unsafe website or take an action known to be suspicious (*i.e.*, an action only attempted by malware). (EX-1001 15:22-25.)

49. Thus, the '250 Patent describes and claims systems and methods that provide more efficient and accurate determinations of whether an object is safe or malware. The '250 patent's "sophisticated pattern analysis" allows for "rapid determination of the nature of the object to be made, without requiring detailed analysis of the object itself." (EX-1001 3:8-18.) Its methods and system decreased reliance on signature verification and negated the need for manual, time-consuming "deep analysis" of an object. (*Id.*) This improved system performance and reduced the number of resources required, allowing for a new method of computer object classification that is rapid, efficient, and accurate.

B. Claim Construction

50. Petitioner does not construe any claim terms in its Petition. Accordingly, I don't take a position on claim construction at this point in the proceeding.

51. I understand that in the District Court's March 16, 2023 claim construction order, the Court determined that no terms of the '250 patent were indefinite and that all terms should be given their plain and ordinary meaning. (EX-2012 3-5, 8.) Consistent with that order, I apply the plain and ordinary meaning of all claim terms.

C. Level Of Ordinary Skill In The Art

52. I understand that certain issues relating to the validity of the '250 patent must be judged from the perspective of a person of ordinary skill in the relevant art, as I discuss below. I have been asked to define the level of a “person having ordinary skill in the art” or “POSITA” that someone would have had at the time the alleged invention as claimed was made. In deciding the level of ordinary skill, I have considered the following factors, which I have been informed are relevant in determining the level of ordinary skill in the art:

- The levels of education and experience of persons working in the field;
- The types of problems encountered in the art;
- The prior art solutions to these problems;
- The rapidity with which innovations are made; and
- The sophistication of the technology.

53. A POSITA at the time of the invention of the '250 patent (which claims priority to an earlier application (GB 0513375.6) filed in Great Britain on June 30, 2005) would have had a Bachelor's degree in an accredited program of electrical engineering, computer engineering, computer science, or a similar discipline, and 2–3 years of practical work or research experience in the general fields of electrical engineering, computer science, networking, communications, and device and

network security. More advanced degrees or training in a related discipline or both can compensate for shorter work experience.

54. I understand that Petitioner has defined the level of ordinary skill in the art as:

[A] person having ordinary skill in the art (PHOSITA) at the time of the earliest claimed priority date for the '250 patent would have had a bachelor's degree in computer science, or a related field, or an equivalent technical degree or equivalent work experience, and an additional one to two years of education or experience in the field of computer security topics such as malware/exploit detection and/or network forensics. More education can supplement practical experience and vice versa.

(Petition, 26.)

55. My opinions as expressed in this Declaration would be the same when viewed under either party's definition of the level of ordinary skill in the art. The minor variations between them are immaterial.

56. Based on either standard, I was at least a person of ordinary skill in the art at the time of the invention of the '250 patent.

V. OVERVIEW OF PRIOR ART

A. Overview Of Halperin

57. Halperin is directed to detecting patterns of collective computer behavior on a network that suggest the presence of viral activity and taking containment actions to halt further spread. (EX-1003 Abstract, [0036]-[0057].).

58. Halperin observed that some prior art antivirus software identified virus behavior by characterizing a program's legitimate behavior and identifying any deviations from such behavior. (EX-1003 [0006].) Halperin criticized this type of object-specific technique as often resulting in false alarms due to the fact that computer user behavior tends to vary over time and between users. (EX-1003 [0006].) Halperin found that this technique's false-positive shortcomings led users to either disable this type of antivirus software entirely or set its sensitivity so low that it failed to find existing viruses. (*Id.*)

59. Halperin further recognized that “[i]n most cases, viruses are spread when computer users send infected files to other computer users,” for example, via e-mail, data storage media (*e.g.*, disk or CD), or by sharing infected files from one computer to another. (EX-1003 [0003].) Some viruses were even capable of spreading on their own “with little or no intervention on the part of the computer user.” (EX-1003 [0004].) For example:

A virus that spreads via e-mail messages will typically access an e-mail program's address book or sent/received mail folders and automatically

send itself to one or more of these addresses. Alternatively, the virus may attach itself to otherwise innocuous e-mail messages that are sent by a computer user to unsuspecting recipients.

(EX-1003 [0004].)

60. Thus, Halperin sought to provide a network-level method to detect symptoms of malware and take containment actions to prevent the possibility of malware spreading, while overcoming the “false alarm” problem it perceived in the prior art. (EX-1003 [0037].) Rather than monitor or analyze a particular object—like the ’250 patent—Halperin’s proposed solution detects suspicious patterns of “target behavior” across a group of computers on a network (indicating the presence of a potential virus in the network) and takes one or more of what Halperin calls “virus containment” actions. (EX-1003 Abstract, [0038]-[0057].) Figure 6 illustrates this overall process:

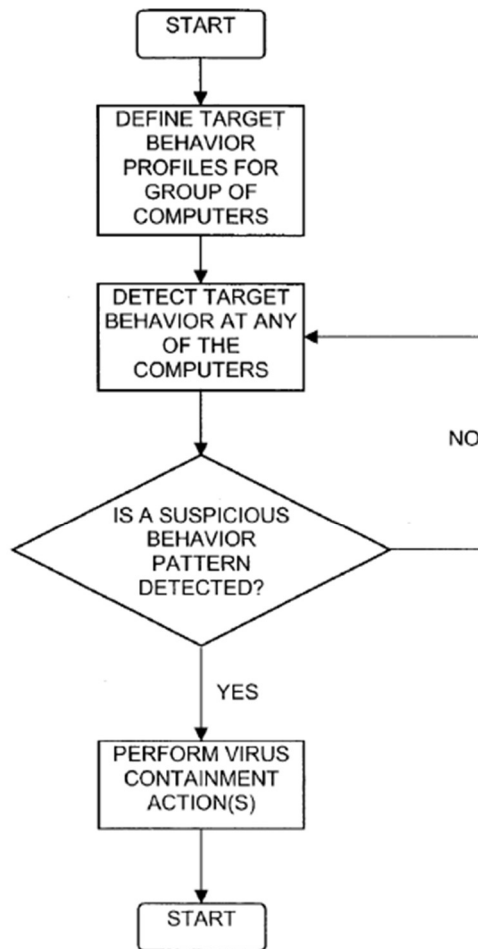


Fig. 6

(EX-1003, Fig. 6.)

61. Halperin starts by defining one or more “target behavior profiles” for a group of computers on a network. (EX-1003 [0038], Fig. 6.) Examples of target behavior profiles include, among others, “[s]ending messages to more than a predefined number of users during a predefined period of time;...[m]odifying operating system file such as the Microsoft Windows registry; [and] [d]eleting more than a predefined number of files on the computer’s hard disk during a predefined

period of time.” (EX-1003 [0039]-[0046].) The ’250 patent examines the *same individual object*—for example, an individual computer program—across multiple computers on a network. By contrast, Halperin seeks the *same computer behavior* being exhibited by multiple computers on a network. (EX-1003 [0037]-[0038].)

62. Halperin then employs a server on the network to correlate the presence of target computer behavior on multiple computers, *e.g.*, by “identifying the same *activity* in different computers at about the same time.” (EX-1003 [0047].) Halperin then teaches determining whether this collective computer behavior corresponds to a “predefined suspicious behavior pattern,” which would indicate that a virus may have infected the network. (EX-1003 [0047].)

63. The exemplary patterns of suspicious behavior Halperin discloses all relate to a “certain percentage of the computers in the network” all doing the same thing, most often within a relatively short amount of time. (EX-1003 [0048]-[0056].) For example, Halperin states that when a “certain percentage of computers in the network send[] more than 10 messages per minute in the last 5 minutes,” a virus may be present on the network. (EX-1003 [0048].) That is, whereas the ’250 patent relies on data that is specific to a particular computer object to determine whether that object is malware (Section IV.A, *supra*), Halperin relies on data that is not particular to any object, because it seeks to determine whether a certain percentage of computers on the network are behaving unexpectedly.

64. Halperin teaches that if the server detects a suspicious behavior pattern, it may initiate “virus containment actions.” (EX-1003 [0057], [0023]-[0030].) The system proffers no means of identifying a particular object as the malware culprit, much less classifying any object as malware or not malware. Instead, Halperin’s proposed solutions amount to either telling a human that the system has discovered symptoms indicating that malware might be present (*id.* [0025]–[0027]) or halting the flow of information across the network (*id.* at [0024], [0028]–[0030].) Halperin does not teach removing the offending software—and could not, because it does not teach how to identify the offending software. (*See* EX-1001 [0024]-[0030].)

65. Thus, whereas the ’250 patent examines information about *the same object* across multiple computers on a network to determine whether the *object* should be classified as malware, Halperin looks for *the same behavior or activity* across multiple computers on a network to determine whether the *network* shows signs of a malware infection, causing the Halperin system to take generalized containment actions. (EX-1003 [0037]-[0038].)

B. Overview Of Mankins

66. Mankins is directed to a kernel-based intrusion detection system using Bloom filters. (EX-1004 Abstract.) The Bloom filter is a data object, embedded in the operating system, that contains data representative of expected system-call behavior associated with a computer program. (EX-1004 2:41-52.)

67. A Bloom filter is generally known as a mathematical construct directed at determining whether a value is within a specified set of values. The Bloom filter is generated by running a specified hash function on each value that is in the set. Using the same hash function on the value being assessed for set membership yields a result that can be quickly compared to the Bloom filter value to return a result. The result indicates either that the value is *definitely not* within the set or that the value *may be* within the set. That is, Bloom filter results do not include false negatives (in this context, mistakenly identifying recognized system calls as unrecognized, which, if it could happen, would result in a false intrusion alert), but may include some number of false positives (in this context, mistakenly identifying unrecognized system calls as recognized, and thus missing a potential sign of intrusion). The rate of false positives is calculable based on the size of the Bloom filter, and that rate may be reduced by increasing the size of the Bloom filter.

68. In a training phase, Mankins creates a Bloom filter object by monitoring normal operation of the computer program when it is not subject to intrusion attacks. (EX-1004 3:37-40.) The Bloom filter tracks individual system calls or system-call sequences, such as “open, read, mmap, mmap, open.” (EX-1004 3:31-33, 6:20-23.) By compacting the data representing normal system-call behavior of an object using hash functions to construct a Bloom filter representing the collection of typical system calls, Mankins is able to quickly determine if a program operating in “real

(i.e., actual) circumstances” is executing anomalous system calls that may suggest “intrusion attempts.” (EX-1004 3:46-67.)

69. Notably, Mankins makes no determination of what any given sequence of system-calls achieves. That is, Mankins focuses entirely on the system calls themselves and is completely agnostic to the behavior of the computer as a result of the system calls.

C. Overview Of Inoue

70. “Anomaly Intrusion Detection in Dynamic Execution Environments” by Hajime Inoue and Stephanie Forrest (“Inoue,” EX-1005) is a paper published in Proceedings of the 2002 Workshop on New Security Paradigms on September 23, 2002. (See EX-1005-10, Declaration of Ingrid Hsieh-Yee, Ph.D., ¶ 27.)

71. Inoue describes “dynamic sandboxing,” which it claims is an improvement for Dynamic Execution Environments, such as Java Virtual Machines. (EX-1005-1.) Inoue teaches “two activities: sandbox generation and sandbox execution.” (EX-1005-3.) Inoue discusses this improvement in the context of the Java Virtual Machine. (EX-1005-1.)

72. A POSITA would have known that a Java Virtual Machine (“JVM”) is a type of virtual-machine environment that enables computers to run programs that are compiled into Java bytecode. The JVM allows programmers to create programs that can run on many different hardware platforms without concern for the

idiosyncrasies of the underlying hardware platform. In this way, the JVM operates as a middleman between the running application, which could be malware, and the host computer.

73. Because of the mobility of its code, Java quickly became a popular programming language, and many programs (both malware and not) were written to operate in the JVM. Thus, researchers began to investigate ways to leverage the JVM's middleman status to improve security for the end-user. And by capitalizing on the JVM's middleman status, Researchers discovered it possible to insulate underlying hardware from malicious attacks by using integrated security mechanisms, such as sandboxes, type checking, and verification mechanisms. (EX-1005-1, -2.)

74. A sandbox is a program-specific application context designed to allow the JVM to safely execute untrusted code from remote sources. Innoue explains that "each sandbox is customized to a given program and context in which it is executed." (EX-1005-3.)

75. Inoue describes a new approach to the standard JVM sandbox called "dynamic sandboxing." (EX-1005-1.) In this approach, there is a learning phase called "sandbox generation" and a runtime phase called "sandbox execution." (EX-1005-3.) The sandbox generation phase involves "running [a] program with an instrumented [Java Virtual Machine]." (EX-1005-3.) During sandbox generation,

“profiling information is recorded to the sandbox profile” by “accumulating records for each unique behavior” of a program. (*Id.*) “Each time a new method is invoked, its signature is written to a log.” (*Id.*) “This log constitutes the sandbox profile.” (*Id.*) Then, during sandbox execution phase, “behavior that isn’t in the profile is considered anomalous.” (*Id.*)

76. Like Mankins’s Bloom filters, Inoue’s dynamic sandbox offers a means for automatically distinguishing routine (and thus allowable) program behaviors from novel and anomalous (and thus disallowed) program behaviors. (Compare *id.* with EX-1004 10:66-67.) Because Inoue works heuristically, it works best when the expected behavior of the program is relatively stable and thus is “best suited to server or middleware applications, where security is most needed and behavior usually limited.” (EX-1005-7.) As such, like Mankins, Inoue is equally susceptible to false positives as a result of user activity. For example, Inoue cautions that “[i]nteractive applications, which are closer to the user, might have functionality that is invoked at infrequent intervals, creating larger numbers of false positives.” (*Id.*)

77. Inoue suggests that its “response” to detecting “anomalous” behavior is “potentially very flexible,” and “could range from logging the anomaly to the application termination.” (EX-1005-3.) But the sole preexisting response within the standard Java security model is “to disallow the attempted behavior.” (*Id.*)

VI. HALPERIN IN VIEW OF MANKINS DOES NOT DISCLOSE THE CHALLENGED CLAIMS

78. The Petitioner contends that the combination of Halperin and Mankins “discloses” every limitation of claims 1-5, 12-17, and 24 (including independent claims 1 and 13). The Petitioner further contends that the combination of Halperin and Inoue “discloses” every limitation of claims 1-5, 12-17, and 24 (again, including independent claims 1 and 13). And the Petitioner contends in Ground 3 that key limitations in the sole remaining independent claim, claim 25, are “disclosed” by Halperin alone or Halperin in view of Mankins for the same reasons as Claim 1 in Ground 1. Petitioner’s other ground, Ground 2, addresses only dependent claims—6 and 18.

Ground 1	Section 103(a)	Obvious in view of Halperin and Mankins	Claims 1-5, 12-17, and 24
Ground 2	Section 103(a)	Obvious in view of Halperin, Mankins, and Arnold	Claims 6 and 18
Ground 3	Section 103(a)	Obvious in view of Halperin, Mankins, Kirby, and Carpentier	Claims 7-11, 19-23, and 25-30
Ground 4	Section 103(a)	Obvious in view of Halperin and Inoue	Claims 1-5, 12-17, and 24

79. I disagree with the arguments in the Petition and the opinions set forth by Dr. Nielson for at least the reasons that I discuss in further detail below.

A. There Is No Motivation To Combine Halperin With Mankins

80. It is my opinion that a POSITA would have had no motivation to combine Halperin with Mankins, and I disagree with Dr. Nielson's opinion to the contrary. First, I read Halperin to discourage detection methodologies like Mankins's. Second, a POSITA would not have expected success in the combination because Halperin and Mankins are not complementary systems. Third, a POSITA would have understood that running distributed Bloom filters provides no benefit over either system because it is in fact simply running each system separately. Moreover, the two systems operate to fundamentally different ends: where Halperin detects malware symptoms by correlating high-level computer actions across a network, Mankins's focus is far narrower: detecting an individual program's intrusion attempts by monitoring that program's system calls within a single computer's OS kernel.

1. Halperin discourages systems like Mankins's.

81. I read Halperin to expressly discourage detecting malware by employing heuristic techniques that are based on an object's normal behavior. Halperin criticizes some previously known malware detections techniques. One example Halperin criticizes is signature matching. Halperin explains that signature

matching is less desirable because it is not effective against new viruses. (Ex. 1003 [0005]). A second method Halperin criticizes is heuristic techniques that “identify typical virus behavior by characterizing legitimate software behavior and then identifying any deviation from such behavior.” (Ex. 1003, [0006]). Halperin cautions that because “computer user behavior is quite dynamic and tends to vary over time and between different users[,] [t]he application of heuristic techniques ... often results in a false alarm whenever a user does anything unusual.” (*Id.*) Halperin explains that as a result of this, “computer users ... disable such software or set the sensitivity of such software so low to the point where new viruses are often not identified.” (*Id.*) Accordingly, it is my opinion that a POSITA reading Halperin would have been discouraged from relying on either signature matching or normal-behavior-based heuristics for viral detections.

82. I read Mankins to teach exactly what Halperin discourages. Mankins teaches implementing Bloom filters in a computer’s kernel. Mankins’s Bloom filters are a record of an object’s expected system-call sequence. Using Bloom filters, Mankins’s monitoring subsystem, which I understand runs at the kernel, “monitor[s] system-call sequences and compare the monitored sequences with predefined normal system-call sequences to determine whether an anomalous sequence of system calls has occurred in the OS kernel.” (EX-1004 6:30-34.) In accordance with

my understanding of Bloom filter data objects, Mankins explains that the monitoring happens “during operation of a particular computer program.” (EX-1004 6:35-39.)

83. Mankins produces the very false positives against which Halperin cautions. I even read Mankins to acknowledge this fact. For example, Mankins recognizes that the complexity of the Bloom filter’s hash function and the size of the bit vector (*i.e.*, the amount of data each Bloom filter can store) determine the likelihood of a false positive. (EX-1004 8:38-42.) Said another way, I understand that at some point Mankins’s Bloom filters will trigger a false negative according to Halperin. Incorporating Mankins’s Bloom filter into Halperin’s system would not improve this reality. What is more, incorporating Mankins’s Bloom filter into Halperin’s system would actually increase the number of false negatives that either system would have encountered individually. For example, because Halperin only triggers an alarm upon a threshold number of computers performing the same suspicious behavior and because Mankins’s system may miss an otherwise abnormal behavior, it is even more likely that Halperin’s system will not trigger an alarm when it should.

84. Based on my understanding of Halperin and Mankins, it is my opinion that a POSITA would have been discouraged from combining the two references by any means. Halperin discourages employing normal-behavior-based heuristics and, instead, relies on detecting a pattern of suspicious behaviors by a threshold number

of computers. It is, therefore, my opinion that a POSITA would not have considered modifying Halperin to focus on normal behaviors of an object—at any stage of Halperin’s detection process.

2. Halperin and Mankins are not complementary systems

85. Halperin and Mankins are directed to different detection methodologies, so a POSITA would have had no motivation to combine them. Where Halperin detects malware symptoms by correlating high-level computer actions across a network, Mankins’s focus is far narrower: detecting an individual program’s intrusion attempts by monitoring that program’s system calls within a single computer’s OS kernel. Nevertheless, both programs have two distinct phases: a learning phase and a detection phase.

86. In my opinion, the salient distinction between Halperin and Mankins is that they operate on different levels—Halperin at the network level and Mankins at the object level—not that they are “reactive” or “proactive.” Accordingly, Dr. Nielson’s suggestion that a POSITA would have been motivated to combine the two references “[b]ecause of these synergies” is further unfounded.

87. I further disagree with Dr. Nielson that a POSITA would have had a reasonable expectation of success in combining Halperin with Mankins such that “Mankins can be used as an active quarantine system.” (EX-1002 ¶ 112). Dr. Nielson alleges that Mankins teaches appending Bloom filters to “a program believed to be

benign.” Dr. Nielson suggests that because of this and because “Halperin recognizes that suspicious activity can be a ‘false positive,’ it would have been obvious to wrap the suspect software into a system that can continue to monitor it along with a guard of acceptable behavior.” (*Id.*) I disagree.

88. Mankins describes Bloom filters and discusses running them at a computer’s kernel. But Mankins never teaches to what programs to append Bloom filters. I do not understand—and Dr. Nielson does not explain—how a POSITA would determine *which* object or objects are suspicious and should be “wrapped” according to the system in Mankins. In a situation where a Halperin system detects symptoms of malware infection, a POSITA would not even have an indication of which computers to focus on, much less any way to determine which objects on those computers should be run first to train a Mankins Bloom filter and then to monitor those objects’ system calls. Because of the large number of objects running on an indeterminate number of computers distributed on a network, I disagree with Dr. Nielson’s contention that “it would have been obvious to wrap the suspect software.” (EX-1002 ¶112.)

89. To the extent Halperin discusses quarantining viral emails, it would not work to append Mankins’s Bloom filters to these or any other objects because Halperin’s quarantine means that the programs are no longer running on any computer. And Mankins’s Bloom filters monitor objects running at endpoints in the

ordinary course of operation. Moreover, to the extent Dr. Nielson is suggesting that the Halperin-Mankins combination would quarantine objects and then run them against a Bloom filter, he is seemingly describing a sandbox exactly like that taught by Dozortsev, which I understand the applicants to have distinguished during the '250 patent's prosecution history. (EX-1007 at 307-18.)

90. Dr. Nielson's opinion here would have required a POSITA to fill in gaps not taught by either reference and would only arrive at exactly what Applicants explained the '250 is not.

3. Combining Halperin with Mankins would diminish accuracy instead of improving upon false positives.

91. Dr. Nielson suggests that the proposed combination of Halperin with Mankins where Mankins is implemented "as an active quarantine system" to Halperin would decrease false positives. (EX-1002 ¶112.) Dr. Nielson contends that when Halperin recognizes suspicious activity, "it would have been obvious to wrap the suspect software into a system that can continue to monitor it." (*Id.*) Dr. Nielson contends that this would offer "another alternative to a simple false-alarm/less-detection trade-off" of Halperin. (EX-1002 ¶111.)

92. Instead, a POSITA would not have been motivated to combine Halperin and Mankins because a POSITA would recognize that the combination could lead to decreased accuracy overall. The combination of Mankins with Halperin would not improve Halperin because it would fail to detect errors in the operation of

Halperin and could introduce increased inaccuracies from which Halperin would not suffer if uncombined. Thus, a POSITA would not have been motivated to combine Halperin with Mankins in any way, regardless of how the combination was actually implemented.

93. The initial flaw with this proposition is that it is not a combination of the two systems, but is instead a “defense-in-depth” strategy involving two distinct pieces of software. In Petitioner’s proposal, Halperin and Mankins would operate entirely independently of each other. Defense-in-depth refers to the cybersecurity strategy of layering multiple different defense strategies in succession to achieve greater protection; however, defense-in-depth requires careful coordination between the layers of defense, such that the layers operate with synergy to increase defense. When care is not taken and multiple security strategies are ineffectively combined, the result is not added security but instead redundancy, waste of resources, and frequently an increase in false detections or decreased protection. Here, a POSITA would not have considered running Halperin and then Mankins as two separate malware detection layers to be a useful defense-in-depth combination because the two layers would be unable to share or benefit from each other’s data and might even contradict each other. Mankins would be an arbitrary addition to Halperin, providing no discernable benefit to its operation.

94. For example, when Halperin operates as designed, a single sensor triggering—or even multiple sensors triggering, as long as the number is below the threshold—will not prompt an alert. A POSITA would not then choose to perform the process of Mankins, which triggers upon a single Bloom filter violation, as an extra test in Halperin. Doing so would contradict Halperin’s teaching that behavior must be evaluated in the aggregate. A POSITA would not find it obvious to connect two cybersecurity solutions with conflicting methodologies, because it would lead to difficulty in interpreting threat risk in the computer system or network. More specifically, if Halperin’s predefined suspicious behavior patterns do not detect viral activity, then it is because Halperin is working as intended and there is no viral outbreak to detect. A POSITA would not view this negative result from Halperin as a prompt to change strategies altogether to now move into an object-focused intrusion detection routine as disclosed in Mankins—much less to *add* such a contrasting methodology to the system according to Halperin.

95. Further, Mankins would not reduce false alarms in Halperin. At the outset, Halperin warns that when “characterizing legitimate software behavior and ... identifying any deviation from such behavior[,] false positives arise” because “computer user behavior is quite dynamic and tends to vary over time and between different users.” (EX-1003 [0006].) To rectify this issue, Halperin employs a central server to detect a threshold number of computers performing the same suspicious

behavior before it signals that suspicious activity is detected. In this way, the Halperin threshold can be increased to decrease false positives (and detection sensitivity as a result), or vice versa. Because Mankins operates at an object level, Mankins can reduce false positives in a Mankins-*only* system by monitoring Bloom filter data objects together. (EX-1004 10:53-64.) But the use of Mankins would not affect the threshold mechanism of Halperin, which would remain at whatever predefined level to which it was originally set. In this way, the risk of false positives in Halperin remains unchanged by the use of Mankins.

96. Besides the inability for Mankins to reduce false positives in Halperin, a POSITA would not have turned to Mankins to correct false negatives in Halperin, either. Mankins's Bloom filter data objects are specific to an object and uniquely created at each remote computer. (EX-1004 10:57-64.) Because of the nature of Bloom filters' compression and Mankins's implementation of them as collections of recognized (and thus allowable) system calls, sometimes a system call passes a Mankins Bloom filter test—*i.e.*, the computer thinks it's okay—when the action should instead have triggered an alert.¹ (EX-1004 8:61-64.) Because of this false-negative issue with Mankins, if combined with Halperin, Halperin risks detecting

¹ Although Mankins defines this as a false positive (because it is looking for Bloom filter *matches*), in the context of Halperin (in which the object is detecting viral activity) this would be a false negative.

fewer behaviors at end-point computers than it should, causing Halperin to incorrectly believe there is no viral outbreak. Accordingly, Mankins would not provide a benefit to Halperin, whether to detect false positives or false negatives in Halperin and a POSITA would not have been motivated to make the combination.

97. Even more so, a POSITA would recognize that combining Halperin with Mankins would be problematic even when Halperin is *not* making faulty detections. For example, if Halperin accurately detects that there is no viral outbreak on the network, but a Mankins Bloom filter data object mistakenly detects a suspicious behavior on an individual remote computer, the system would be forced to resolve this discrepancy between the two sources of data. This is a fundamental incoherency between the object-focused approach of Mankins and the aggregated network-level behavioral approach of Halperin. Dr. Nielson offers no way of discerning which system's results should prevail. A POSITA would lose confidence in both systems.

B. There Is No Motivation To Combine Halperin With Inoue

98. For many of the same reasons discussed above with regards to the Halperin-Mankins combination, a POSITA would not have been motivated to combine Halperin with Inoue. Dr. Nielson sets forth the same arguments—nearly verbatim—in support of combining Halperin with Inoue that he did for the combination of Halperin and Mankins. But, just as Halperin teaches away from the

heuristic behavioral approach of Mankins, it teaches away from Inoue, which functions using exactly the same technique. And just as with Mankins, a POSITA would not have been motivated to combine Halperin with Inoue because the systems do not operate in a complementary manner and would be ill-fitting in their suggested combination. Finally, just as with Mankins, combining Halperin with Inoue would diminish accuracy instead of reducing the number of false positives. I will address each of these points briefly to avoid repeating much of section VI.A.

99. First, Inoue’s dynamic sandbox technique utilizes a similar strategy to Mankins’s Bloom filters and thus would be discouraged by Halperin for the same reason as Halperin teaches away from Mankins. During training, Inoue first profiles a program’s typical behavior by logging records of each behavior observed, and then uses that profile to classify “behavior that isn’t in the profile [as] ... anomalous.” (EX-1005-3). Thus, Inoue’s dynamic sandboxing—like Mankins’s Bloom filters—teaches “heuristic techniques to identify typical virus behavior by characterizing legitimate software behavior and then identifying any deviation from such behavior.” (EX-1003 [0006].) Moreover, Inoue recognizes that it suffers from “larger numbers of false positives” when operating on “[i]nteractive applications, which are closer to the user” because of the user’s propensity “to invoke[]” certain “functionality . . . at infrequent intervals.” (EX-1005-7.) Halperin expressly teaches

away from heuristic behavior detection because of its unreliability when user interaction is taken into account. (EX-1003 [0006].)

100. Second, as discussed above with regards to Mankins, using Inoue as “an active quarantine system” is untenable because Halperin does not teach how to identify “suspect software” to be quarantined. (EX-1002 ¶426.) Thus, Dr. Nielson’s proposed implementation of the combination is not technically feasible.

101. Third, Dr. Nielson’s supposed motivation to combine Halperin with Inoue is not a proper combination and would simply diminish Halperin’s accuracy instead of improving false positives. Using Inoue as an “active quarantine” or a false-positive detector alongside Halperin is not a combination, but simply a defense-in-depth strategy that layers two disjointed solutions in sequence. Dr. Nielson never explains how Halperin would receive data or interact at all with the Inoue system. In fact, as discussed above with regard to Mankins, if Halperin accurately detects that there is no viral outbreak on the network, but Inoue’s dynamic sandbox mistakenly detects a suspicious behavior of an individual program, the system would be forced to resolve this discrepancy between the two sources of data. This is a fundamental incoherency between the object-focused approach of Inoue and the network-level behavioral approach of Halperin. Dr. Nielson offers no way of discerning which system’s results should prevail and a POSITA would lose confidence in both systems.

C. Independent Claims 1, 13, and 25

102. Petitioner relies on the same analysis of Halperin to support Independent Limitations 1[b] to 1[d] and 13[b] to 13[d], and 25[b] in Grounds 1, 3, and 4. (Petition 29-31, 46, 63.) Petitioner further relies on the analysis of dependent claim 2, which in turn relies on the analysis of limitations 1[b] to 1[d], in support of its challenge to limitation 25[c]. (Petition 38-39, 63.) Accordingly, my arguments addressing limitations 1[b] to 1[d] apply with equal force to at least Petitioner's arguments in support of limitations 13[b] to 13[d], 25[b], 25[c], and dependent claim 2.

103. Petitioner's Ground 2 addresses only dependent claims.

104. The Petitioner also relies on the same analysis of Halperin in view of Mankins to support Independent Limitation 1[f] in Ground 1 and Independent Limitation 24[e] in Ground 2. (Petition 32, 64-65). Accordingly, I address limitation 1[f]. The arguments I raise in response to Petitioner's contentions regarding that limitation apply with equal force to limitation 24[e] in Ground 2.

105. Additionally, I address the Petitioner's analysis of Halperin in view of Inoue to support Independent Limitation 1[f] in Ground 4.

106. As I discussed above, I disagree with Dr. Nielson that there is a motivation to combine Halperin with Mankins or Halperin with Inoue. In either purported combination, the two references employ disparate detection techniques to

target different types of malware problems resulting in different ameliorative steps. Where Halperin is designed to detect viral outbreaks in a network by analyzing and correlating computer behavior, Mankins and Inoue are designed to prevent system intrusions by detecting anomalous program behavior at the object level. As a result of their differences, neither would benefit the other, their combination would require significant modification to Halperin's system, and their combination would suffer from inefficiencies.

107. I further disagree with Dr. Nielson that Halperin alone teaches each of 1[b] to 1[d]. *First*, I disagree that Halperin receives data *about an object* from remote computers. It is my opinion that Halperin, instead, receives only behavior data about actions generically performed by a computer—without regards to any specific object. *Second*, I do not agree that Halperin utilizes received data to determine that a computer object is malware. *Third*, I do not agree that Halperin *classifies objects* as malware.

1. Halperin Does Not Disclose Limitations 1[b] to 1[d], 13[b] to 13[d], or 25[b] and 25[c]

108. It is my opinion that Halperin does not teach or render obvious the limitations directed at receiving information about an object from remote computers and using that information to determine whether the object is malware and classifying the object.

109. Halperin is directed to determining whether there is a viral outbreak in a network by examining when a percentage of computers perform the same suspicious behavior, such as, for example, sending emails to too many users within a predetermined amount of time, sending emails without interaction from a human user, or deleting too many files within a predefined period of time. (EX-1003 [0048]-[0056].) These behaviors are not necessarily performed by any specific object and so Halperin's data collection has no need to, and indeed does not, collect information about any particular object. As such, Halperin also does not rely on information about a particular object to reach a determination and classify that object as malware.

- a. **1[b] at a base computer, receiving data about a computer object from each of plural remote computers on which the object or similar objects are stored, the data including information about the behaviour of the object running on one or more remote computers.**

110. I disagree with Dr. Nielson that Halperin discloses this limitation because I do not read Halperin to teach that the data that is communicated to Halperin's base computer is object specific or even associated with any particular object. In other words, I do not understand Halperin's detections to be focused on any particular object and so I do not understand Halperin to communicate any data that is "about a computer object."

111. In the overview of the '250 patent, it is explained that the base computer "may gather data on multiple instances of a particular object across different remote

computers and correlate those different data points (regarding, e.g., name, location, vendor, product, and version) to create a score.” (EX-1001 10:41-47). In contrast, Halperin does not gather any of this type of data. Instead, Halperin gathers computer-related data that is not necessarily tied to any object. Halperin operates using sensors that monitor computer behavior for certain patterns of suspicious actions, that “trip” or “trigger” an alert to the central computer when the suspicious behavior is detected. (EX-1003 [0046].) During Halperin’s detection phase, “the system logs every event of a sensor identifying a suspicious behavior on a specific computer into a central database ... preferably includ[ing] the type of event, the time it occurred and on which computer it occurred (if relevant).” (EX-1003 [0090].) Halperin does not disclose that the sensors collect any object-specific information and specifically identifies the type of information stored in the central database, none of which is object-specific.

112. Dr. Nielson argues that because Halperin “expresses that the same behavior must be detected at two or more computers” that “the data received is about the same ... ‘object.’” (EX-1002 ¶127.) As explained above, it is possible to correlate behavior while entirely ignorant of the computer object responsible for that behavior. Namely, Halperin can recognize that several sensors of the same type have “tripped” without depending on the knowledge of which object tripped those

sensors. In fact, Dr. Nielson concedes that “the data received by the central computer is behavioral.” (EX-1002 ¶128.)

113. Then, to justify his statement that Halperin can identify specific virus software, Dr. Nielson jumps to an entirely different embodiment of Halperin that relies on the use of “decoy” or “bait” email elements and not on target behavior patterns. (EX-1002 ¶128.) Halperin’s decoy emails are a technically and fundamentally distinct embodiment, which relies on detecting emails sent to decoy addresses and quarantining the objects attached thereto. (EX-1003 [0035]; EX-1003 [0086] (separately identifying “**detectors** (e.g. file deletion and Windows Registry access) and **decoys** (e.g. e-mail decoy).”))

114. Moreover, the e-mail decoys themselves are not “data about a virus” (as Dr. Nielson suggests) nor “data about a computer object” (as the limitation requires). In much the same way that “target behavior” is not tied to a specific computer object, e-mails received at the server are not tied to any computer object operating at the remote computers, because there is no way to tell from the e-mail itself which computer object was responsible for sending that e-mail. Halperin confirms that this is true because Halperin explains that a system administrator is required “to determine whether or not the messages were indeed sent by a computer virus.” (EX-1003 [0058].) If Halperin’s system were capable of determining which computer object was responsible for sending the infected e-mail, it would not be

necessary to utilize a human to make that determination instead. Thus, Dr. Nielson’s bare assertion that a “POSITA would have understood ‘infected’ messages to typically include the virus itself” is contradictory to the disclosure of Halperin. (EX-1002 ¶ 130.)

115. To elucidate the fundamental distinction between Halperin’s behavior detection and receipt of data *about an object*, I offer a practical example that would have been obvious to a POSITA. Halperin describes one suspicious behavior pattern as “a certain percentage of computers in the network deleting more than 10 files in the last 1 minute.” (EX-1003 [0050].) Consider several users on different computers all cleaning their hard drives around the same time—for example, after the expiration of a litigation hold. This would trigger Halperin’s predefined suspicious behavior pattern. But a report of that human-triggered, non-malware behavior is not “data about [any one] computer object,” much less evidence that a given “object is malware.” Accordingly, even though the false alarm is, according to Halperin, “an indication that a computer virus may have infected those computers,” it is not an indication that any computer object is malware.

116. Halperin’s target behavior detectors are not “data about a computer object” and Halperin’s use of e-mail decoys does not make it so. Accordingly, Halperin does not disclose, either expressly or inherently, receiving data about a computer object at a base computer.

- b. 1[c] determining in the base computer whether the data about the computer object received from the plural computers indicates that the computer object is malware;**

117. For reasons similar to those I set forth for 1[b], and which I incorporate by reference here, I disagree with Dr. Nielson that Halperin teaches “determining in the base computer whether the data about the computer object received from the plural computers indicates that the computer object is malware.” Again, I understand Halperin to perform an object-agnostic determination of whether there are symptoms of malware activity across a computer network. Halperin does not suggest the identity of the virus, or even identify an object that *might* be a virus.

118. Dr. Nielson contends that “Halperin discloses determining at the centralized server if behavior data received from computers on the network indicates the presence of malicious software.” Based on this, Petitioner asserts that Halperin “discloses identifying portions of emails and software ... that are suspicious or malicious.” (Petition 30.) There is an unjustifiable leap from claiming that Halperin discloses correlating behavior data to identify symptoms of malware to claiming that Halperin discloses identifying the responsible software. As discussed above in response to limitation 1[b], Halperin correlates behavior and in no way relies on an identification of the object responsible for that behavior. Moreover, if Halperin were capable of determining that particular computer objects are malware, it would not

require the extra step of using a system administrator “to determine whether or not the messages were indeed sent by a computer virus.” (EX-1003 [0058].)

119. Dr. Nielson makes the bare allegation that “[t]he detecting of malware is necessary step in order to mitigate or stop the malicious software from causing harm.” (EX-1002 ¶135.) Dr. Nielson points out that Halperin initiates “virus containment actions” when it detects suspicious behavior, and thus Halperin must have “detected malware.” (*Id.*) I disagree on both counts.

120. First, a POSITA would understand that it is possible to mitigate or stop the operation and spread of malicious software without ever detecting or identifying the malicious program. The most obvious solution is simply to turn off the computer—if the program is not running it cannot cause damage. This solution does not require identifying the malicious program. Other, more complex solutions, could be to turn off network traffic to the computer (mitigating the spread of the malware to other computers) or restoring the computer to an earlier state—neither of which depends on identifying the malicious program. Dr. Nielson’s first conclusion that a POSITA *must* identify the malware to stop it from causing harm is patently false.

121. Dr. Nielson’s second conclusion, that Halperin’s “virus containment actions” indicate that Halperin has identified the object responsible for the suspicious behavior on the network, is similarly flawed. Halperin explains that it initiates those responses because a certain threshold of predefined suspicious actions

in the *entire network* was met. (EX-1003 [0090].) This is in no way explicitly correlated to a specific computer object. First, this paragraph explains that the alarms are determined based on a “predefined threshold.” (*Id.*) It is illogical to say that an individual computer object has a threshold. Instead, what does have a threshold is a probability that a collection of numerous suspicious actions indicates that there is in fact a “virus in the system.” (*Id.*) Second, as discussed previously, Halperin does not collect object-specific behavior and thus cannot initiate a virus alarm based on object-specific classifications.

122. To reinforce this point, I offer another analogy. When a patient calls his doctor and says he is experiencing coughing, sneezing, and a runny nose, the doctor may recommend, among other things, that the patient stay home from work or school to avoid passing on the presumed infection to others. But the fact that the doctor detected symptoms and took containment actions does not mean that he also necessarily diagnosed (or classified) the patient as having any virus—much less that the doctor identified the particular virus responsible for the patient’s symptoms.

123. Similarly, here, when Halperin detects malware symptoms, it initiates actions to prevent further spread from potentially infected systems. But Halperin’s disclosure of actions taken to contain a possible virus on a network does not inherently disclose or indicate to a POSITA that Halperin has classified any particular object as malware—especially where none of the remedial containment

actions Halperin discloses requires knowledge that a particular object might be or is in fact malware. (See EX-1003 [0024]-[0030].)

124. Accordingly, I do not read Halperin to communicate data about objects to the central computer. Therefore, I further do not read Halperin to make a determination about whether a computer object is malware based on “data about the computer object.”

c. 1[d] classifying the computer object as malware when the data indicates that the computer object is malware

125. For reasons similar to those I set forth for 1[b] and 1[c], and which I incorporate by reference here, I disagree with Dr. Nielson that Halperin teaches “classifying the computer object as malware when the data indicates that the computer object is malware.” I disagree that Halperin classifies *any* malware, let alone that it classifies a particular computer object as malware.

126. To begin with, Halperin does not classify any malware; it simply detects symptoms of viral outbreaks. Halperin explains that it correlates “target behavior ... to a predefined suspicious behavior pattern of target behavior as an indication that a computer virus may have infected those computers.” (EX-1003 [0047].) Identifying indications that a computer virus may have infected several computers is not classifying malware, because it merely identifies symptoms of malware and deduces that malware may be present. As discussed above, Halperin does not disclose how the object responsible for the behavior is to be identified. For

example, Halperin does not disclose how to trace “[a] certain percentage of computers in the network deleting a file with the same name within the last 1 hour” back to the objects (or even the computers) that deleted the file. (EX-1003 [0051].) Observing this behavior is merely observing a symptom of what may be several different objects’ actions, not the objects themselves. As another example, Halperin does not disclose how to trace the infected e-mail message in quarantine back to the object (or even the computer) that sent it. (EX-1003 [0058].) In sum, quarantining the e-mail is merely quarantining the symptom of an object’s actions, not the object itself.

127. By way of analogy, Halperin’s disclosure is akin to suspecting that a pandemic exists because doctors have found that millions of people all over the world are simultaneously exhibiting signs of illness (shortness of breath, fevers, etc.). Classifying malware is akin to a determination by one doctor that one individual has COVID-19. The two scenarios are not dependent on each other: a doctor can diagnose an individual patient regardless of the existence of a pandemic and it is possible to suspect that a pandemic exists without knowing the specific virus responsible for it. Similarly, Halperin deduces that a virus is likely responsible for the suspicious symptoms, but does not disclose detecting the specific computer object responsible for creating those suspicious symptoms (actually classifying the computer object).

128. Again relying on the “virus containment actions” disclosed in Halperin, Dr. Nielson contends that all of them “identify potentially malicious computer objects such as software.” (EX-1002 ¶140.) This conclusion is easily dismissed. For example, “suspending any or all messages sent by computer 100” does not, in any way, require identifying a potentially malicious computer object such as software. It merely requires blocking computer 100’s access to the e-mail server. (EX-1003 [0024].) Even “forwarding messages that are addressed to a decoy address to a third party for analysis,” which suggests that the forwarded message contains some infected software, requires that the analysis—the “identify[ing] [of] potentially malicious computer objects”—must be done by a remote human being. (EX-1003 [0025].) In this way, there is no equivalence between Halperin’s “virus containment actions” and any form of classification.

129. Lastly, Dr. Nielson contends that a POSITA would have recognized that the “infected” messages typically include the virus itself. As I discussed in 1[b], this argument contradicts the teaching of Halperin because if the messages typically included the virus itself then no further analysis by a system administrator would be necessary.

130. Halperin’s failure to disclose classifying any computer objects is by design, because Halperin’s system does not rely on the classification or identity of any individual computer objects. None of Halperin’s disclosed containment actions

require classification of a computer object. Accordingly, I do not read Halperin to teach classifying any computer objects as malware.

2. The Petition's Combinations of Halperin in View of Mankins and Halperin in View of Inoue Do Not Render Obvious Limitations 1[f] and 24[e]

131. As I discussed above in sections VI.A and VI.B, I disagree with Dr. Nielson that there is a motivation to combine Halperin with either Mankins or Inoue to perform these limitations. The two combinations of references employ disparate detection mechanisms; where Halperin is designed to detect symptoms of viral outbreaks in a network by analyzing and correlating computer behavior and to contain those potential outbreaks, Mankins and Inoue are designed to prevent system intrusions by detecting anomalous behavior at the level of individual programs. As a result of their divergent approaches to solving essentially different problems, Halperin would not benefit either Mankins or Inoue (or vice versa); their combinations would suffer from inefficiencies; and adding Mankins or Inoue would at impair Halperin's ability to perform its own disclosed detection methodology.

- a. (Mankins) 1[f] automatically generating a mask for the computer object that defines acceptable behavior for the computer object, wherein the mask is generated in accordance with normal behavior of the object determined from said received data;**

132. Dr. Nielson recognizes that Halperin's central server identifies acceptable behavior of the monitored computers. (EX-1002 ¶ 148.) From this, the Petitioner leaps to the conclusion that the behavioral information aggregated at the

central computer corresponds to “normal behavior *of the object*.” This is incorrect. The type of information that Halperin logs in the central database is object-agnostic. Halperin explains “[t]hat information is logged in a central database that logs for each event the **type of event**, **time of event** and **on which computer** it took place.” (EX-1003 [0086] (emphasis added).) In fact, Halperin further enumerates the information collected: “Virus behavior may be monitored ... in terms of numbers of message per unit of time, shapes of utilization graphs, such as for disk storage access or CPU usage, ... histograms of e-mail length, ... recipients per message, etc.” (EX-1003 [0076].) None of this information is tied to a specific computer object or software. Dr. Nielson’s leap to the conclusion that “monitoring system events” discloses “generating ... acceptable behavior for the computer object” is thus unsupported. (EX-1002 ¶149.) The Petitioner does not include Mankins for this portion of the limitation, nor would Mankins cure this deficiency.

133. The Petitioner and Dr. Nielson rely on Mankins in this limitation to teach the “mask.” Dr. Nielson suggests that the obvious implementation of the combination would be to “automatically generate Mankins’ [B]loom filter of ‘normal program behavior’ ... on the central system after receiving information about the object from the remote computers disclosed by Halperin.” (EX-1002 ¶153.) There are three flaws with this proposed “combination.” *First*, it is not a proper combination, but rather two systems acting independently, not sharing

information, and perhaps even acting antagonistically. *Second*, Halperin does not receive “information about the object from the remote computers” (and especially not “information about a software’s system calls,” as the Petitioner puts it). *Third*, Mankins explicitly teaches away from centralizing its Bloom filter data objects on the server, because this approach would reduce false positives. (See EX-1004, 2:14-22).

134. *In the first case*, as discussed in section VI.A.2 above, running Mankins’s Bloom filters on Halperin’s remote computers after Halperin’s network-level behavior-detection system is not a combination of the two systems. Instead, it is a “defense-in-depth” strategy involving two distinct pieces of software. The result is that the two systems do not communicate useful information between them: Halperin provides no identification of the object responsible for the suspicious network behavior and thus Mankins must run on every software on every computer.

135. Further, the two systems are likely to contradict each other. Halperin warns that activity observed on just one computer “cannot be confidently attributed to a virus.” (EX-1003 [0037].) For this reason, a POSITA would understand that Halperin aggregates sensors from different remote computers until the threshold number of computers is reached before suspecting a virus in the network. When Halperin operates as designed, a single sensor triggering—or even multiple sensors triggering, as long as the number is below the threshold—will not prompt an alert.

A POSITA would not then choose to perform the process of Mankins, which triggers upon a single Bloom filter violation, as an extra test in Halperin. Doing so would contradict Halperin's teaching that behavior must be evaluated in the aggregate. Moreover, if Halperin accurately detects that there is no viral outbreak on the network, but a Mankins Bloom filter data object mistakenly detects a suspicious behavior on an individual remote computer, the system would be forced to resolve this discrepancy between the two sources of data. This is a fundamental incoherency between the object-focused approach of Mankins and the aggregated network-level behavioral approach of Halperin. Dr. Nielson offers no way of discerning which system's results should prevail. A POSITA would lose confidence in both systems.

136. *In the second case*, Halperin does not receive "information about a software's system calls," as the Petitioner puts it, or "information about the object from the remote computers" as Dr. Nielson puts it. This mischaracterization of Halperin means that Dr. Nielson's proposed combination is based on unsupported functionality and would not be motivated.

137. *In the third case*, combining Halperin and Mankins in the way suggested by Dr. Nielson would eliminate Mankins's ability to detect false positives. Mankins explicitly teaches away from a singular Bloom filter generated at a central server out of the system-call behaviors observed on every remote computer. At the

outset, Mankins explains that generating the Bloom filter data object anywhere except the computer kernel is ill-advised:

Unfortunately, several shortcomings are apparent in existing system-call-based intrusion detection techniques. For example, significant delays are inherent in these techniques and may make them impracticable for use with complex computer programs that are processed at high speeds. In particular, it takes time to compare system calls tracked in the OS kernel with data of a profile database stored outside of the OS because communications must be sent back and forth between the OS kernel and the profile database.

(EX-1004 2:14-26.) Additionally, in order to reduce false positives, Mankins *requires* that there are distinct Bloom filters generated at *each* of the multiple remote computers so that their evaluations can be compared to one another:

[W]hen several network devices generate separate Bloom filter data objects 180, are monitored together, and are subject only to attacks that affect each of the network devices within a small amount of time, it is likely that at least one of the network devices would detect a false positive that is produced by another device.

(EX-1004 10:57-62.) The differences in the Bloom filters generated at each remote computer operates as a sanity check for false positives; if a system-call sequence is anomalous in one Bloom filter it may not be in another, and this comparison allows Mankins to determine that the sequence is less likely to be a false positive. Combining all of Mankins's system calls by generating "the [B]loom filter once on

the central computer and then transmit[ting]” it out to all the remote computers entirely deprives Mankins of its ability to check for false positives and eviscerates one of Dr. Nielson’s stated motivations for combining Halperin and Mankins. (EX-1002 ¶153.) Accordingly, a POSITA would not have sacrificed Mankins’s efficiency by adapting the Bloom filter data objects to Halperin’s central computer 502 as advocated by Petitioners.

138. Lastly, as I described above, *supra* ¶88, Halperin does not provide Mankins with the requisite information to efficiently perform its detection mechanism. A POSITA would not understand for which computer object, running on which computer Mankins should “automatically generate Mankins’ [B]loom filter.” (EX-1002 ¶153.) Halperin only identifies that suspicious behavior is present on the network, not the context necessary to identify any suspicious computer objects.

139. Accordingly, I do not read Halperin in view of Mankins to teach this limitation nor would this limitation be obvious over Halperin in view of Mankins.

- b. (Inoue) 1[f] automatically generating a mask for the computer object that defines acceptable behavior for the computer object, wherein the mask is generated in accordance with normal behavior of the object determined from said received data;**

140. As discussed *supra* ¶132, the type of information that Halperin logs in the central database is object-agnostic and thus cannot correspond to “normal behavior *of the object*” as the Petitioner suggests. (Petition 71.)

141. A POSITA would have recognized that Mankins and Inoue provide fundamentally similar solutions to the problem of detecting intrusion at the program level. Each of the references describes heuristically training a model—in Mankins, a Bloom filter; in Inoue, a dynamic sandbox—with the expected behavior of a program. (EX-1004 7:8-11; EX-1005-3.) Each of the references then describes using the model to filter the program’s behavior. (*Compare* EX-1004 12:1-14 *with* EX-1005-3.) In each system, behaviors (in Mankins, system calls; in Inoue, method invocations) are classified based on whether they fall within the set of behaviors the program engaged in during the training phase. Behaviors that were previously observed during training are assumed to be normal behaviors. Behaviors that are novel—unrecognized when compared against the model—are assumed to be aberrant behaviors, and those aberrant behaviors trigger a system response.

142. The few differences between the Mankins and Inoue systems are due largely to Inoue’s Java-specific implementation. For example, Inoue describes

filtering a program's method invocations in a Java Virtual Machine sandbox instead of system calls in an operating system kernel. (EX-1005-3.) A POSITA would have understood that such differences are largely immaterial for the purposes of understanding how either system would function alongside a system according to Halperin. Thus, a POSITA would have understood that the same considerations that counsel against employing Mankins alongside Halperin similarly counsel against employing Inoue alongside Halperin.

143. As I discussed above, Halperin expressly teaches away from using a heuristic behavior-detection strategy like the one Inoue employs. (EX-1003 [0006].) Halperin teaches that such strategies are prone to errors especially in the context of interactive programs where users may introduce unexpected behaviors into a system. (*Id.*) For its part, Inoue confirms that its system is more prone to errors in those contexts: "Interactive applications, which are closer to the user, might have functionality that is invoked at infrequent intervals, creating larger numbers of false positives. (EX-1005-7.) Thus, a POSITA taking into account these teachings in both systems would have been disinclined to employ Inoue in combination with Halperin.

144. Running Inoue's dynamic sandboxes on Halperin's remote computers after Halperin's network-level behavior-detection system is not a combination of the two systems. Instead, it is a "defense-in-depth" strategy involving two distinct pieces

of software. The result is that the two systems do not communicate useful information between them: Halperin provides no identification of the object responsible for the suspicious network behavior and thus Inoue must run every piece of software through the rigors of its sandbox on every computer² on the Halperin network.

145. Further, the two systems are likely to return contradictory reports. Halperin warns that activity observed on just one computer “cannot be confidently attributed to a virus.” (EX-1003 [0037].) A POSITA would understand that for this reason Halperin aggregates sensors from different remote computers until the threshold number of suspicious-behavior reports is reached before suspecting a virus in the network. When Halperin operates as designed, a single sensor triggering—or even multiple sensors triggering, as long as the number is below the threshold—will not prompt an alert. A POSITA would not then choose to perform the process of Inoue, which acts whenever an individual program’s single behavior exceeds the sandbox limits, as an extra test in Halperin. Doing so would contradict Halperin’s teaching that behavior must be evaluated in the aggregate. Moreover, if Halperin accurately detects that there is no viral outbreak on the network, but an Inoue

² Notably, although Halperin is directed to monitoring computers in a network and Inoue is directed to monitoring Java Virtual Machines, Dr. Nielson never addresses these technical differences.

dynamic sandbox mistakenly detects a suspicious behavior on an individual remote computer, the system would be forced to resolve this discrepancy between the two sources of data. This is a fundamental incoherency between the object-focused approach of Inoue and the aggregated network-level behavioral approach of Halperin. Dr. Nielson offers no way of discerning which system's results should prevail. A POSITA would lose confidence in both systems.

146. Lastly, Halperin does not receive “information about a software’s system calls,” as the Petitioner puts it, or “information about the object from the remote computers” as Dr. Nielson puts it. (EX-1002 ¶468.) This mischaracterization of Halperin means that Dr. Nielson’s proposed combination with Inoue is based on unsupported functionality and would not be motivated. For that reason, Halperin does not provide Inoue with the requisite information to efficiently perform its detection mechanism. A POSITA would not understand from Halperin around which computer object, running on which computer, Inoue should “wrap” a “sandbox” so that its behavior may be “stopped dynamically” if it “gets worse.” (EX-1002 ¶472.) Halperin only identifies that suspicious behavior is present on the network, not the context necessary to identify any suspicious computer objects.

147. Accordingly, I do not read Halperin in view of Inoue to teach this limitation nor would this limitation be obvious over Halperin in view of Inoue.

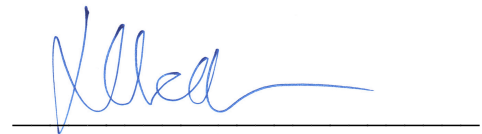
VII. CONCLUSION

148. For at least the foregoing reasons, it is my opinion that Petitioner's grounds do not render obvious any of the challenged claims of the '250 Patent.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further 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 1001 of Title 18 of the United States Code.

Executed this 17th day of July, 2023 in Los Angeles.

Respectfully submitted,

A handwritten signature in blue ink, appearing to read "N. Medvidovic", is written over a horizontal line.

Nenad Medvidovic, Ph.D.