

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

TREND MICRO, INC.,

Petitioner

v.

OPEN TEXT INC.

Patent Owner

Case IPR2023-01389

Patent No. 8,856,505

**DECLARATION OF RON SCHNELL
IN SUPPORT OF PATENT OWNER'S PRELIMINARY RESPONSE**

TABLE OF CONTENTS

I.	QUALIFICATIONS.....	2
II.	UNDERSTANDING OF RELEVANT LEGAL PRINCIPLES	5
III.	STATE OF THE PRIOR ART	7
IV.	THE '505 PATENT.....	9
	A. Summary Of The '505 Patent	9
	B. Claim Construction.....	13
	C. Level Of Ordinary Skill In The Art.....	14
V.	OVERVIEW OF PRIOR ART	16
	A. Overview Of Garfinkel.....	16
	B. Overview Of Hوجلund.....	21
VI.	GARFINKEL IN VIEW OF HOGLUND DOES NOT RENDER OBVIOUS THE CHALLENGED CLAIMS	25
	A. Independent Claims 1, 6, & 11	25
	1. The cited combination of prior art does not render obvious managing pestware-related events during a second period of a boot sequence.	26
	2. The cited combination of prior art does not render obvious scanning data in a registry during a second period of a boot sequence.	31
	3. The cited combination of prior art does not render obvious generating, in response to monitoring, a record of events.	37
	B. Dependent Claims	39
VII.	CLAIMS AND/OR LIMITATIONS NOT ADDRESSED IN THIS DECLARATION	40

VIII. SECONDARY OBJECTIVE INDICIA OF NONOBVIOUSNESS	40
IX. CONCLUSION	41

EX.HIBIT LIST

Exhibit	Description
2001	Declaration of Ron Schnell
2002	<i>Curriculum Vitae</i> of Ron Schnell
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	<i>Webroot, Inc. and Open Text Inc., v. AO Kaspersky Lab. et al.</i> , Case No. 6:22-cv-243-ADA-DTG, Dec. 9, 2022 Transcript of Motions Hearing, Dkt. 133 (W.D. Tex. Dec. 15, 2022) (excerpted)
2007	Dani Kass, “Catching Up On Patent Litigation With Judge Albright” (Law360 Mar. 14, 2023)
2008	<i>Webroot, Inc. and Open Text Inc., v. Trend Micro, Inc. et al.</i> , Case No. 6:22-cv-243-ADA-DTG, Amend the Schedule Order, Dkt. 521 (W.D. Tex. Dec. 7, 2023)
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. 391 (W.D. Tex. July 25, 2023)
2010	<i>Webroot, Inc. and Open Text Inc., v. Sophos Ltd.</i> , Case No. 6:22-cv-240-ADA-DTG, Defendant Sophos Ltd.’s Final Invalidity Contentions

Exhibit	Description
	– Claim Chart 505-A-21 (“Garfinkel”) (W.D. Tex. Nov. 1, 2023) (excerpted)
2011	<i>Webroot, Inc. and Open Text Inc., v. Sophos Ltd.</i> , Case No. 6:22-cv-240-ADA-DTG, Defendant Sophos Ltd.’s Final Invalidity Contentions – Claim Chart 505-A-22 (“Hoglund”) (W.D. Tex. Nov. 1, 2023) (excerpted)
2012	Wayback Machine re symposium website address, https://web.archive.org/web/202300000000000*/https://www.ndss-symposium.org/ndss2003/
2013	Network and Distributed System Security Symposium conference proceedings--2003 [electronic resource]: NDSS '03 Symposium; https://science-catalogue.canada.ca/record=b2011094~S6

I, Ron Schnell, 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 8,856,505 (“the ’505 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 ’505 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 ’505 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. 8,856,505 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 ’505 patent at the time of the alleged invention. I have also been asked to provide my

technical opinions on how concepts in the '505 patent specification relate to claim limitations of the '505 patent. In reaching the opinions provided herein, I have considered the '505 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 field, 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 am a Managing Director at Berkeley Research Group, LLC ("BRG").

8. I have over 40 years of experience in the fields of computer science and software.

9. I have a master's degree in computer science from Syracuse University and have a working knowledge of over 40 computer languages.

10. Over the course of my career, I have worked at many of the world's largest computing companies, including (but not limited to) Bell Laboratories, IBM Corporation, and Sun Microsystems. I was also an early researcher on the Arpanet, which later became the Internet. In addition to working at large corporations, I have founded three start-up companies to date, including a backup and restore software

company and a .COM company that I sold to a public corporation in 2000.

11. Early in my career, I worked at the Artificial Intelligence Lab at the Massachusetts Institute of Technology.

12. Later, in approximately 1986, I worked at Bell Laboratories on the UNIX Operating System, writing many thousands of lines of source code representing portions of the core kernel of the UNIX operating system.

13. From 1988-1990, I worked at IBM Corporation on the kernel of their operating system called AIX (for the IBM 370 mainframe, as well as the IBM PS/2), as a programmer, and eventually a development manager.

14. From 1991-1992, I worked as a Principal Member of the Technical Staff at Encore Computer Corporation, on the kernel of their operating system called UMAX.

15. From 1992-1995, I worked at Sun Microsystems, on the kernel of their operating system called Solaris.

16. From 2002-2005, I was a Vice President at Equifax Corporation, one of the three major credit bureaus in the United States, leading software development in their Internet Marketing division. I was also the company expert on e-mail delivery, responsible for the delivery of over one billion e-mail messages per month.

17. I have also been on staff and consulted at numerous other companies and institutions, including (but not limited to) Syracuse University, Encore

Computing Corporation, Harris Computer Corporation, and ITT.

18. During my career I founded several startup companies, including two kernel software companies. I sold my last startup company to a public company in 2000.

19. I am currently an adjunct professor of computer science and a former advisory board member at the College of Engineering and Information Science at Nova Southeastern University.

20. From 2005 through 2011, I was responsible for monitoring Microsoft Corporation in connection with the antitrust litigations brought by the U.S. Department of Justice and the Attorneys General of 19 states in *United States v. Microsoft* and *New York v. Microsoft*. In this role, I had responsibilities akin to those of a Special Master and provided periodic compliance reports to Hon. Colleen Kollar-Kotelly of the United States District Court for the District of Columbia. I was responsible for directing the day-to-day enforcement of the final judgments against Microsoft and supervised a team of 93 people in performing these duties. In this role, I spent time auditing the Windows kernel, and used computer simulation models to detect noncompliance with the Final Judgments.

21. Two publications I have written in the field are: Jay L. Himes, Jason Nieh & Ron Schnell, “Antitrust Enforcement and Big Tech: After the Remedy Is Ordered,” 1 Stanford Computational Antitrust 64 (2021), and Ron Schnell, “Hacking

101: Using Social Engineering Increases Security Attack Effectiveness,” Coast Guard Journal of Safety & Security at Sea, Proceedings of the Marine Safety & Security Council 71, no. 4 (2014) 25–27.

II. UNDERSTANDING OF RELEVANT LEGAL PRINCIPLES

22. 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.

23. 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.

24. 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.

25. 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.

26. 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

27. “Personal computers and business computers are continually attacked by trojans, spyware, and adware, collectively referred to as ‘malware’ or ‘pestware.’ These types of programs generally act to gather information about a person or organization—often without the person or organizations’ knowledge.” (EX-1001 1:32-37.) While pestware may be non-malicious, it still “may cause issues with

privacy or system performance.” (EX-1001 1:37-39.)

28. Pestware techniques evolved to include “cloaking techniques” that made such pestware difficult to identify, remove, or even detect. (EX-1001 1:46-48, 1:57-60.) For example, prior art detection tools relied on the dependability of operating system calls to return lists of running programs, file system and registry contents, etc. (EX-1001 1:51-56.) But evolved pestware techniques included “intercepting and altering the results of operating system queries,” thereby “modifying the behavior of a computer operating system itself.” (EX-1001 1:48-56.) That is, where existing anti-pestware applications relied on real-time shields, those shields were not executed until after pestware had already been able to execute and modify the operating system. (EX-1001 1:57-61.)

29. Figure 3 of the ’505 patent, below, illustrates a traditional boot and operating system sequence in which pestware (e.g., a rootkit) is loaded prior to a conventional anti-virus/anti-spyware application in the boot sequence:

Conventional Monitoring

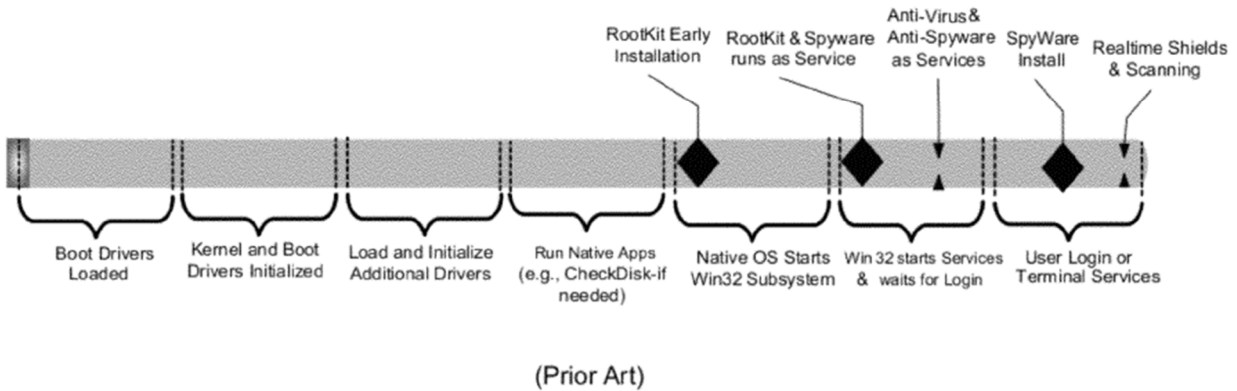


FIG. 3

(EX-1001 Fig. 3.) Because the rootkit is loaded earlier, user-mode services or applications like the conventional anti-virus/anti-spyware application are unable to detect and prevent the pestware. (EX-1001 1:57-65.)

30. Accordingly, conventional systems were still unable to detect rootkits effectively and accurately, and an improved technology for managing malware during a boot sequence was needed.

IV. THE '505 PATENT

A. Summary Of The '505 Patent

31. The '505 patent discloses a system and method for managing “pestware,” which it describes as including “any program that collects and/or reports information about a person or an organization and any ‘watcher processes’ related

to the pestware.” (EX-1001 1:41-45; *see id.* 1:32-37; Section V(B), *infra.*) Specifically, the ’505 patent recognized that due to pestware cloaking techniques, existing pestware prevention that relied on “real-time shields” were ineffective, “because these real-time shields begin execution after pestware has been able to execute and modify the operating system.” (EX-1001 1:57-61, Fig. 3.) The ’505 patent also explains that “[p]eriodic scanning for pestware is particularly ineffective because it leaves a window of time between scans in which pestware can execute and cloak itself.” (EX-1001 1:65-67.)

32. Accordingly, the ’505 patent discloses and claims a system and method for kernel-level detection of pestware during a boot sequence. (EX-1001 Title, Abstract, cls. 1, 6, 11.)

33. Figure 4 illustrates this kernel-level monitoring that occurs earlier in the boot sequence than conventional anti-virus/anti-spyware applications:

Kernel-Level Monitoring

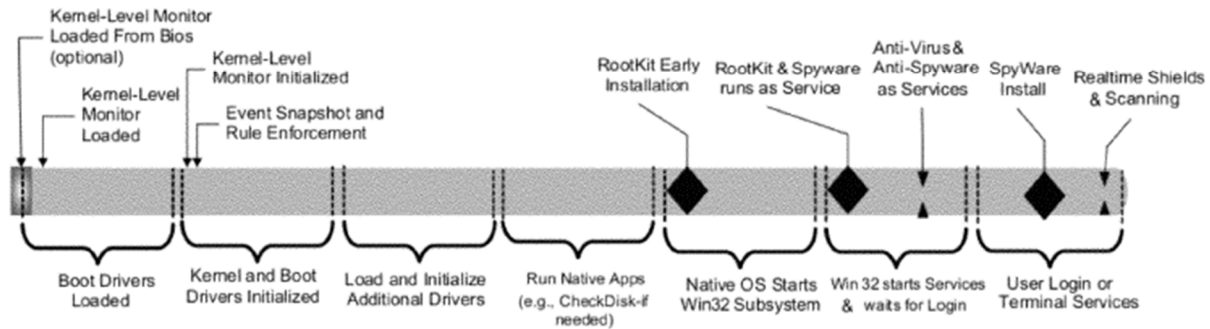


FIG. 4

(EX-1001 Fig. 4.)

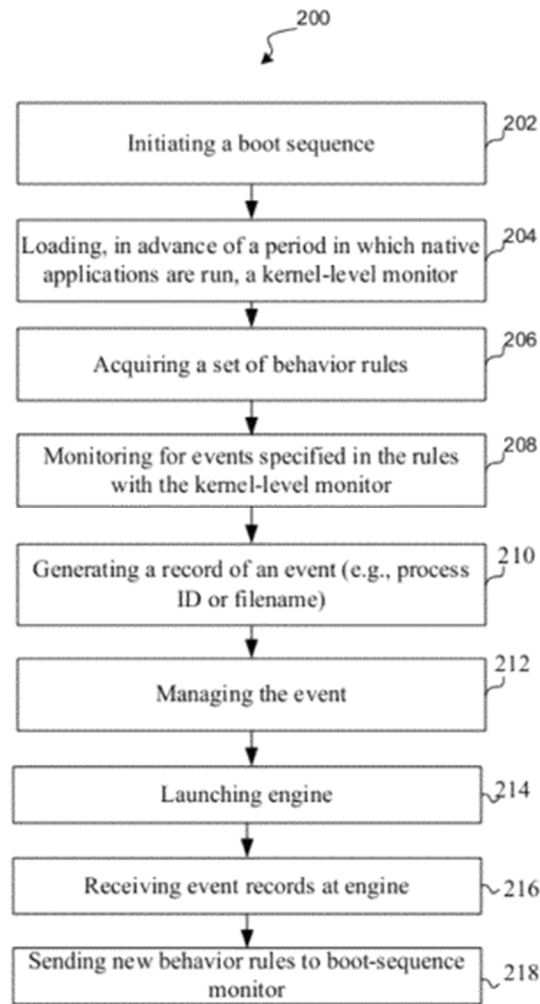
34. The '505 patent identifies a “first period” in a boot sequence, which occurs after the kernel is loaded but before the computer is configured to run native applications and before a sub-system of an operating system is loaded. (EX-1001 2:17-29, cl. 1.) During this first period of the boot sequence, a kernel-level monitor observes and manages events, such as those events that might be associated with the loading, execution, and behavior of pestware. (EX-1001 3:66-4:1, 2:17-29, cl. 1.)

35. The '505 patent identifies a “second period” in the boot sequence, which occurs when the computer is configured to run native applications. (EX-1001 2:17-29, cl. 1.) During this second period of the boot sequence, any pestware-related

events may be managed according to a set of behavior rules that were previously compiled by the pestware management engine. (EX-1001 3:24-46, cl. 1.) Data in the computer's registry may also be scanned during this second period for pestware. (EX-1001 2:28-29, 5:60-6:17, cl. 1, Fig. 5.)

36. In order to ensure the most effective monitoring and management, a record of the monitor-observed events is generated and analyzed to identify the pestware-related events. (EX-1001 cl. 1, 3:53-57.) For example, the event log created by the kernel-level monitor may be examined by a pestware management engine for instances of known pestware, suspicious events by unknown software, and other patterns of events that may be useful in identifying pestware. (EX-1001 3:53-57.) The pestware management engine modifies the set of behavior rules in order to prevent pestware-related events, such that each time the kernel-level monitoring occurs, it has the most up-to-date and most effective set of behavior rules. (EX-1001 3:58-4:5, cl. 1.)

37. Figure 2 illustrates this an embodiment of this process:



(EX-1001 Fig. 2.)

B. Claim Construction

38. I understand that in the parallel district court litigation, the court construed the following terms of the '505 patent:

Claim Term	Court's Construction
------------	----------------------

<i>pestware</i> (cls. 1, 2, 4, 6, 7, 9, 11, 12, 14, 16-20)	“any program that collects and/or reports information about a person or an organization and any ‘watcher processes’ related to the pestware” (EX-2009 3.)
[Order of the method steps in claims 1, 6, 11.]	No order recited except for the “generating” step must happen before the “analyzing” step. (EX-2009 17.)

39. I further understand that the court found all other terms to have their plain-and-ordinary meaning. (EX-2009 15-18.)

40. Petitioner states that the Board should construe the claims of the '505 patent consistent with the court's district court claim construction order. (Petition 9.)

41. For purposes of this declaration, I have applied the plain-and-ordinary meaning of all claim terms and the district court's construction of “*pestware*” and method steps, restated above.

C. Level Of Ordinary Skill In The Art

42. I understand that certain issues relating to the validity of the '505 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 of 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.

43. In my opinion, a POSITA at the time of the invention of the '505 patent (around Aug. 7, 2006) 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. More education can supplement practical experience and vice versa.

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

[A] person having ordinary skill in the art (PHOSITA) as of the earliest priority date for the '505 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. More education can supplement practical experience and vice versa.

(Petition 8.)

45. Despite minor variations in these two positions, these variations would not alter my opinions expressed in this Declaration.

46. Based on either standard, I was at least a person of ordinary skill in the art at the time of the invention of the '505 patent.

V. OVERVIEW OF PRIOR ART

A. Overview Of Garfinkel

47. Garfinkel seeks to disclose a new architecture for an intrusion detection system (IDS). (EX-1006 Title, Abstract, 2 col. 1.)

48. Garfinkel explains that prior art IDSs were typically defeated through one of two ways. First, an IDS could be defeated through attack, e.g., tampering with the IDS or trusted components. (EX-1006 1 col. 1.) Or an IDS could be defeated through evasion, e.g., disguising malicious activity so it goes unrecognized. (EX-1006 1 col. 1.) Garfinkel posits that countering these two approaches produced conflicting requirements—specifically, the need for providing the IDS with better visibility into the state of monitored systems conflicted with the need to isolate the

IDS from the monitored system and attacker. (EX-1006 1 cols. 1-2.) The existing IDS architectures described by Garfinkel included two types: (1) network-based intrusion detection systems (NIDS) that offered high attack resistance at the cost of visibility; and (2) host-based intrusion detection systems (HIDS) that offered high visibility but sacrificed attack resistance. (EX-1006 1 col. 2.)

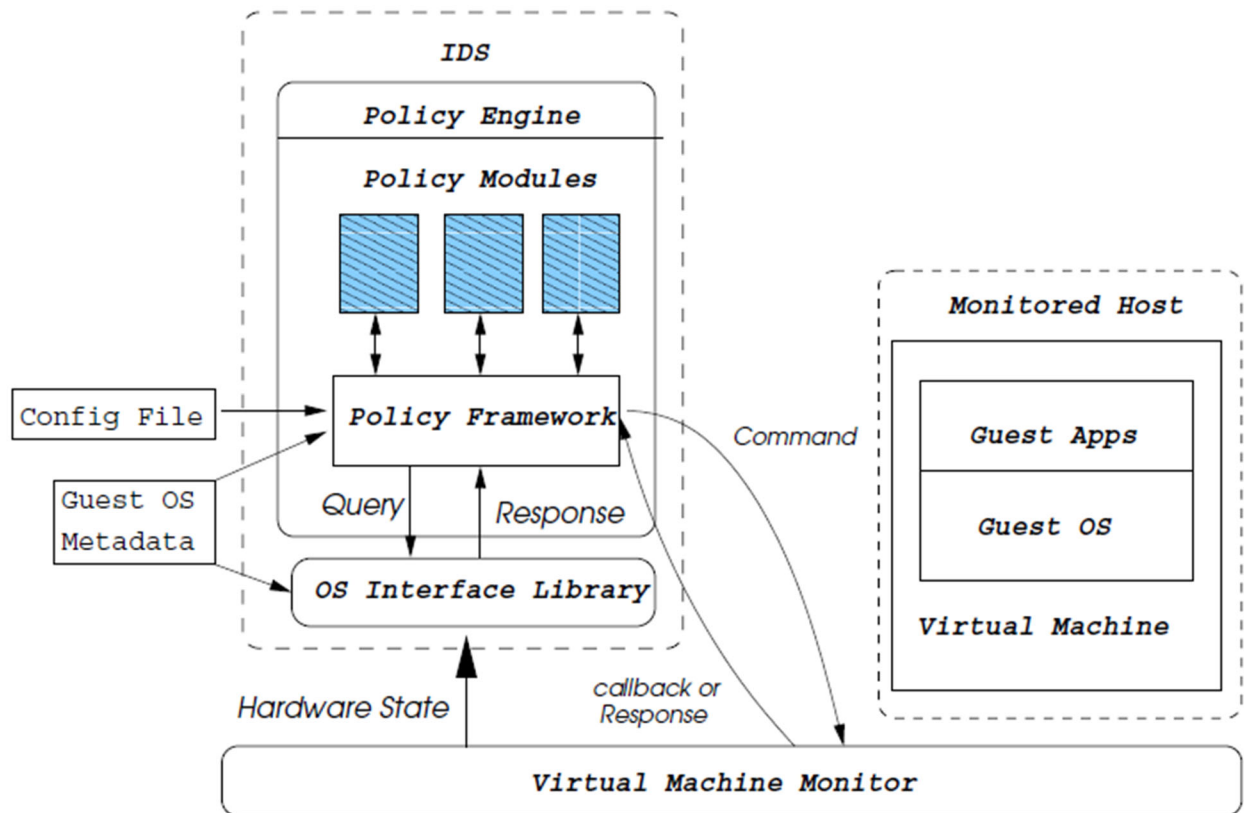
49. Garfinkel acknowledged that some systems might move the IDS into the kernel; however, Garfinkel criticizes that approach, stating that “kernel code is no safer from tampering by a privileged user than normal user-level code.” (EX-1006 2 col. 2.) Garfinkel asserts that “[a] critical fault in a kernel-based IDS will often [] fail open” and “cause the entire system to crash or allow the attacker to compromise the kernel.” (EX-1006 3 col. 1.)

50. Thus, Garfinkel sought to present a new IDS architecture to resist both evasion and attack by providing visibility into the state of the monitored host, while still providing strong isolation for the IDS. (EX-1006 1 col. 2.)

51. Garfinkel proposes an architecture based on “virtual machine introspection.” (EX-1006 Title, Abstract, 2 col. 1.) Garfinkel discloses an architecture which places the IDS on the same machine as the host it is monitoring while still being “outside” of the host system by being in a different hardware protection domain. (EX-1006 1 col. 2, §11.) The IDS relies on a virtual machine monitor (VMM)—i.e., software—to virtualize the hardware of the host machine into

a virtual machine. (EX-1006 §§3-3.1.) The virtual machine runs a “guest” operating system (OS) and guest applications. (*Id.*) In this way, the IDS is isolated from the monitored host, while still having visibility into the virtual machine via the VMM. (EX-1006 §3.4.) Garfinkel refers to this approach of “inspecting a virtual machine from the outside for the purpose of analyzing the software running inside” as “virtual machine introspection (VMI).” (EX-1006 2 col. 1.)

52. Garfinkel’s proposed VMI-based IDS architecture is illustrated below in Figure 1. On the right, the figure illustrates a virtual machine being run on the host that is to be monitored. (EX-1006 Fig. 1.) On the left, the figure illustrates the IDS, which gains visibility into the virtual machine through the virtual machine monitor:



(EX-1006 Fig. 1.)

53. Garfinkel's IDS comprises two parts. (EX-1006 §4.3.) The first, an operating system (OS) interface library, provides “an OS-level view of the virtual machine's state in order to facilitate easy policy development and implementation.” (*Id.*) The second, a policy engine, relies on one or more policy modules to “execute IDS policies by using the OS interface library and the VMM interface.” (*Id.*)

54. Garfinkel discloses six different exemplary policy modules—four “polling modules” that are run periodically and two “event-driven modules” that are triggered by specific events. (EX-1006 §5.4.2; *see also generally id.* §6.)

55. For example, one of the polling policy modules is a “lie detector”

module. (EX-1006 §6.1.1.) “Attackers often achieve stealth by modifying the OS kernel, shared libraries, or user-level services,” which can lead to inconsistencies between the kernel (or hardware) view of the system and the view provided by user-level programs. (*Id.*) Garfinkel therefore discloses a module which periodically inspects the hardware state both directly and by querying the host system through user-level programs. (*Id.*) If it detects a conflict between those views, then it knows the system is lying and reports the presence of tampering. (*Id.*)

56. Depending on the type of policy modules implemented in the IDS, the policy engine uses a variety of commands to inspect, monitor, and/or control what happens in the virtual machine. (*See, e.g.*, EX-1006 §§4.2.1, 5.4.1.) Where polling policy modules are implemented, the IDS uses “inspection commands,” which cause the VMM to provide the hardware state of the virtual machine to the IDS. (EX-1006 6 col. 1; *see also id.* §6.1.)

57. Unlike the polling modules, which run periodically on a schedule, the event-driven policy modules are triggered by an event. (EX-1006 §5.4.2.) The IDS uses “monitor commands” to request notification (e.g., by registering callbacks) for events taking place in the virtual machine. (EX-1006 § 6 col.1; *see also id.* §5.4.1.) “[E]ach event-driven checker registers all of the events it would like to be notified of with the policy framework.” (EX-1006 §6.2.) “At runtime, when one of these events occurs, the VMM relays a message to the policy framework.” (*Id.*) The

virtual machine may continue to run, or the VMM can be directed to suspend the machine at each event. (*Id.*)

58. For example, one of the event-driven policy modules is a “memory access enforcer,” which marks certain portions of the kernel (e.g., `sys_call_table`) as read-only through the VMM. (EX-1006 §6.2.1.) If a malicious program tries to modify these sections of kernel memory, the VM will be halted and the enforcer notified. (*Id.*)

59. That is, upon obtaining notification of an event (for an event-driven module) in the virtual machine, the IDS’s policy engine determines whether the system has been compromised based on the relevant policy modules. (EX-1006 §§4.3.2, 5.4-5.4.2.) The IDS then uses “administrative commands” to control execution (e.g., suspend or resume) in the virtual machine. (EX-1006 §4.2.1.) For example, in Garfinkel’s prototype, Livewire, Garfinkel teaches that its prototype, Livewire, suspends the virtual machine’s execution every time an event occurred until the IDS responded with a command to resume. (EX-1006 §5.2.) Garfinkel’s prototype was implemented via Linux-based software. (EX-1006 §§5-5.1.)

B. Overview Of Hoglund

60. Hoglund is a book titled “Rootkits: Subverting the Windows Kernel.” (EX-1006 1.) Hoglund explains that:

Rootkits can be generic to different versions of an OS, but will still be limited

to a given OS family. For example, some rootkits in the public domain affect all flavors of Windows NT, 2000, and XP. This is possible only when all the flavors of the OS have similar data structures and behaviors. It would be far less feasible to create a generic rootkit that can infect both Windows and Solaris, for example.

(EX-1007 41.) Hoglund specifically discusses rootkit “threats to the Windows operating system.” (EX-1007 2.)

61. Hoglund describes a wide variety of techniques that may be used by Windows-based rootkits (EX-1007 38), but only a few discrete disclosures are cited in Petitioner’s assertions. In particular, Hoglund discloses a few rootkit methods which *may* rely on registry keys.

62. For example, Hoglund teaches methods used by attackers to load a rootkit as a driver into kernel memory, which can be done with or without creating a registry key. (EX-1007 61-65 (“There are many ways to load a driver into the kernel....[Y]ou can load a driver into the kernel without having to create any registry keys.”).) Rootkits are “typically installed using a loader program,” which runs as a “Ring Three application” with less privileges than code running in “Ring Zero.” (EX-1007 71.) “In order to load rootkit into the kernel, these loader programs use special function calls that allow them to access Ring Zero.” (EX-1007 71.)

63. As another example, Hoglund teaches that an attacker may try to

modify “the hardware mechanisms that work behind the scenes to enforce security and memory access in the Windows operating system.” (EX-1007 70.) For example, the CPU relies on a variety of tables stored in memory in order to know what software routine should deal with every important event. (EX-1007 73.) One of these tables Hoglund discloses is an “Interrupt Descriptor Table (IDT),” which the CPU uses to find interrupt handlers. (EX-1007 73.) “Interrupts are used for a variety of low-level functions in a computer,” e.g., whenever a keystroke is typed on the keyboard. (EX-1007 82.)

64. On Windows XP and greater, the IDT is set to read-only; one way around this is altering two registry keys. (EX-1007 78-79.) Hoglund notes that there are other means for rootkits subverting these tables that do not involve registry keys. (EX-1007 79 (“[T]hese registry keys are no protection against rootkits, since a rootkit can modify the page tables directly or use the CR0 trick to enable or disable access restrictions on-the-fly.”); *see also id.* 61 (“[Y]ou can load a driver into the kernel without having to create any registry keys.”).) Rootkits may create a new interrupt table to hide modifications made to the original interrupt table. (EX-1007 82.) While a virus scanner may check the integrity of the original IDT, the rootkit can make a copy of the IDT and redirect the CPU. (EX-1007 82.)

65. As another example, Hoglund teaches that a rootkit may attempt to inject a shared library using the Windows registry. (EX-1007 96-97.) Hoglund

discloses a registry key specific to some of the Windows operating systems, which a rootkit may alter to one of its own dynamic link libraries (DLLs) (i.e., a shared library). (EX-1007 96.) When a user application is loaded that uses “User32.dll,” User32.dll will load the DLL listed in this key and load them into the application’s address space. (EX-1007 96-97.)

66. Hoglund discusses some of the techniques that can be used to detect the presence of a rootkit, based on their potential for altering registry keys. (EX-1007 346.) For example, Hoglund teaches using detection software to hook specific software functions but acknowledges that this method requires knowing which associated registry keys to guard. (EX-1007 347. While Hoglund discloses a few likely registry keys, it “assume[s] you have discovered all the locations to watch in order to prevent rootkits from loading, and...further assume[s] you have resolved all the possible names of critical resources to protect.” (EX-1007 347.) Even then, Hoglund criticizes this detection method, because it would still require a signature for comparison and, therefore, be limited to known attacks. (EX-1007 347.)

67. As an alternative, Hoglund discloses a method for detecting hidden registry keys. (EX-1007 359.) Hoglund explains that the goal of such a technique is to catch the operating system in a “lie”—i.e., where an API returns values known to be false. (EX-1007 359.) Specifically, Hoglund discloses a known tool called “rootkit-revealer.” (*Id.*) First, the tool determines that the “truth” is by parsing files

that correspond to the registry without the aid of the standard Win32 API calls and parsing the file system at a very low level while avoiding typical API calls. (*Id.*) The tool then calls the highest-level APIs to compare the result with what it knows to be true. (*Id.*) A discrepancy indicates the presence of a rootkit. (*Id.*)

VI. GARFINKEL IN VIEW OF HOGLUND DOES NOT RENDER OBVIOUS THE CHALLENGED CLAIMS

68. The petition analyzes a single ground, asserting that Garfinkel in view of Hoglund renders obvious every limitation of independent claims 1, 6, and 11. This assertion underlies Petitioner’s challenge of all remaining dependent claims. (Petition 4.)

Ground 1	Section 103(a)	Obvious over Garfinkel and Hoglund	Claims 1-20
-----------------	----------------	------------------------------------	-------------

69. As I discussed above, I apply the plain and ordinary meaning of all claim terms and apply the district court’s claim construction of “*pestware*” and the order of the method steps. Section IV.B, *supra*.

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

A. Independent Claims 1, 6, & 11

71. The cited combination of prior art does not disclose or suggest the following limitations of the independent claims: (1) managing pestware-related

events during a second period of a boot sequence; (2) scanning data in a registry during a second period of a boot sequence; or (3) generating, in response to monitoring events, a record of events.

1. The cited combination of prior art does not render obvious managing pestware-related events during a second period of a boot sequence.

72. As I discussed previously, the '505 patent identifies two periods in a computer's boot sequence. (Section IV.A, *supra*.) Limitation [1d] recites “*managing pestware-related events in accordance with a set of behavior rules during a second period in the boot sequence occurring when the computer is configured to run native applications.*” (EX-1001 cl. [1d]; *see also id.* cls. [6f], [11d].) That is, the '505 patent requires managing pestware-related events during the boot sequence and, more specifically, during a period in the boot sequence when the computer is configured to run native applications.

73. Petitioner relies on two alternative disclosures in Garfinkel as allegedly disclosing this limitation: (1) Garfinkel's disclosure of a lie detector policy module; or (2) Garfinkel's disclosure of a memory access enforcer policy module. (Petition 16-17.) But application of neither the lie detector policy module nor the memory access enforcer policy module discloses or suggests this limitation.

74. Garfinkel discloses six different exemplary policy modules—four “polling modules” that are run periodically and two “event-driven modules” that are

triggered by specific events. (EX-1006 §5.4.2; *see also generally id.* §6.)

75. The lie detector policy module, which Petitioner references, is one of the exemplary polling policy modules. (EX-1006 §6.1.1.) That is, the IDS may use “inspection commands” to obtain visibility into the virtual machine to periodically check on things. (*Id.* 6 col. 1; *see also id.* §6.1.) Thus, Garfinkel teaches that the lie detector policy module is only run periodically, at various points in time, to check whether any conflict exists between the hardware-view of the system and the user-level program-view of the system. (*Id.*; *see also id.* §5.4.2.)

76. Garfinkel’s reliance on a lie detector policy module does not “*manag[e] pestware-related events.*” Because the module is a polling module that is run periodically, it is limited to “report[ing] the presence of malicious tampering,” when it detects a lie. (EX-1006 §6.1.1.) But Garfinkel does not disclose any event associated with the application of or analysis by this polling module. For example, Garfinkel does not disclose a means of identifying what event might have resulted in the “lie”—only that a lie exists. (EX-1006 §6.1.1 (“This technique has the nice property that it does not matter what part of the system the intruder modified in order to elicit the malicious behavior.”).)

77. Garfinkel also does not disclose or suggest application of the lie detector policy module during “*a second period in the boot sequence.*” Garfinkel does not explicitly disclose or reference a boot sequence at all. (*See generally* EX-

1006.) And the nature of the lie detector module—i.e., the fact that it is a polling module that is run periodically—indicates to a POSITA that the module’s inspection and query of the hardware view and user-level program view may take place at any time in the system’s lifespan—long after the boot sequence has completed. (*See* EX-1006 §§5.4.2, 6.1; *see also* Section V.A, *supra*.) That is, nothing in Garfinkel’s disclosure suggests applying the lie detector module during the boot sequence at all.

78. While Garfinkel’s memory access enforcer module is an event-driven module, Garfinkel also does not disclose or suggest application of this module “*during a second period in the boot sequence.*” Again, Garfinkel does not mention or reference the boot sequence at all or dictate any particular timing of applying event-driven policy modules. (*See generally* EX-1006.) Indeed, Garfinkel identifies a number of events that occur long after the boot sequence is completed. For example, Garfinkel explains that monitor commands can be used to register “callbacks for events that a policy module wants to be notified of, e.g., a write to a range of memory,” as well as “the VM rebooting or powering down.” (EX-1006 §5.4.1.) Writing to a range of memory may occur in many instances, such as editing a document, which occurs long after a boot sequence has completed. A virtual machine need only be “reboot[ed]” or “power[ed] down” where it is already powered up and has completed the boot sequence.

79. Garfinkel’s memory access enforcer module, specifically, is triggered

by modifications to specific areas of memory that have been pre-marked as read-only. (EX-1006 §6.2.1.) But Garfinkel says nothing about the timing of such events, and a POSITA would understand that malware or pestware might attempt a modification of memory at any time during the system’s lifespan—including long after the boot sequence has completed. (*See id.*)

80. The petition does not contain any explanation as to why a POSITA would have allegedly “known that rootkit attacks attempt to modify memory and perform malicious system calls in the period of the boot sequence after /sbin/init/ is completed and after OS subsystems are initiated.” (Petition 17.) Dr. Nielson’s declaration does not support this conclusion, as he concedes that Garfinkel’s memory check “is used during any time period to detect unauthorized memory changes.” (EX-1003 ¶ 193.) For example, in limitation [1c], Dr. Nielson asserts that Garfinkel’s monitoring and blocking the SucKIT rootkit must mean that the memory access enforcer module is applied during the *first* period of the boot sequence, before the computer is configured to run native applications. (EX-1003 ¶¶181-183.) But in limitation [1d], Dr. Nielson asserts that a POSITA would have understood that a rootkit might modify memory during the user-level boot sequence that begins after /sbin/init and that, therefore, Garfinkel discloses applying the memory access enforcer during the second period of the boot sequence. (EX-1003 ¶193.)

81. But Garfinkel does not disclose any specific timing at which the policy modules are applied. Thus, even if Dr. Nielson were correct that a POSITA would have understood that Linux rootkits *could* modify memory during the user-level boot sequence that begins after `/sbin/init` (*id.*), nothing in Garfinkel suggests checking for rootkits during the boot sequence at all or that Garfinkel's prototype was ever used to check for rootkits during the boot sequence.

82. The most that Garfinkel discloses is that its prototype was capable—using the memory enforcer—to block the SucKIT rootkit's attempt to modify the interrupt dispatch table. (EX-1006 §7.2.) And while—for this limitation--Dr. Nielson asserts that the SucKIT rootkit might have attempted to make such a modification after the `/sbin/init` program was launched, Dr. Nielson fails to acknowledge that—while the `/sbin/init` program is launched during the boot sequence—it runs during the lifetime of the system, i.e., until it is shut down. That is, Garfinkel's disclosure of blocking the SucKIT rootkit does not disclose or suggest application of its memory access enforcer module at any particular time, does not disclose or suggest application of this module during the boot sequence, and certainly does not disclose or suggest application of this module at any particular time during the boot sequence.

83. Petitioner does not rely on Hoglund as allegedly teaching this limitation and does not rely on any argument that such a limitation would have been obvious.

(Petition 17-18.)

2. The cited combination of prior art does not render obvious scanning data in a registry during a second period of a boot sequence.

84. As I discussed previously, the '505 patent identifies two periods in a computer's boot sequence. (Section IV.A, *supra*.) Limitation [1h] recites "*scanning data in a registry of the computer for pestware during the second period in the boot sequence.*" (EX-1001 cl. 1; *see also id.* cls. [6j], [11h].) That is, the '505 patent requires scanning data in a registry of the computer for pestware during the boot sequence and, more specifically, during a period in the boot sequence when the computer is configured to run native applications. (*See* EX-1001 cl. 1.)

85. Petitioner relies on Garfinkel's disclosure of a signature detector policy module as allegedly disclosing "*scanning*" for pestware during boot and asserts that a POSITA would have been motivated, based on Hoglund, to scan "*data in a registry of the computer.*" (Petition 21-22; *see also id.* 33, 39.) But once again, Garfinkel does not disclose or suggest scanning during a boot sequence or at any particular time during the boot sequence, and Hoglund only discloses scanning after the boot sequence has completed. Furthermore, a POSITA would not have been motivated to combine the teachings of Garfinkel and Hoglund to "*scan[] data in a registry.*"

a. Neither Garfinkel nor Hoglund disclose or render obvious scanning during the second period in the boot sequence.

86. While Petitioner relies on Garfinkel’s disclosure of a signature scanner for this limitation, Petitioner does not allege that either Garfinkel or Hoglund teach scanning a registry “*during the second period in the boot sequence*,” specifically. (Petition 21-22.) Petitioner merely asserts that Garfinkel discloses a system with the *capability* “at any time during boot to scan a registry.” (Petition 21.) And the cited portion of Dr. Nielson’s declaration does not explain the expert’s conclusory assumption that Garfinkel allegedly discloses running the scanner during the second period of a boot sequence. (EX-1003 ¶¶231-232.)

87. Garfinkel does not teach or suggest running its signature detector policy module during the boot sequence and certainly does not teach or suggest running its signature detector policy module at any particular time in the boot sequence. Rather, Garfinkel’s signature detector policy module is another type of polling module—i.e., one that “periodically check[s] the system for signs of malicious activity.” (EX-1006 §§6.1.3, 6.1.) Based on Garfinkel’s teaching of performing periodic checks, Garfinkel indicates to a POSITA that such a detector should be applied throughout the life of the system, long after the boot sequence is complete. (*See id.*)

88. Hoglund does not supply the missing disclosure, because Hoglund explicitly teaches the application of its “Rootkit-Revealer” tool *after* the boot

sequence has completed. As I explained above, Hoglund discusses rootkits targeted at Windows-based systems. (Section V.B, *supra*.) In discussing potential tools for handling such rootkits, Hoglund references a known tool called “Rootkit-Revealer,” which detects hidden registry entries and determines the “truth.” (EX-1007 359.) But to do so, Hoglund explains that the tool “parses the file system at a very low level, *avoiding the typical API calls*.” (*Id.*) Rootkit revealer then “calls the *highest level APIs* to compare the result with what it knows to be true.” (*Id.*) In doing so, the tool detects discrepancies or changes to the registry. (*Id.*)

89. Critically, based on Hoglund’s teaching that this tool relies on a comparison to the “highest level APIs,” a POSITA would understand that the rootkit revealer is not capable of calling “the highest level APIs” until the system is fully up and running—i.e., *after* the boot sequence is complete. (EX-1007 359.) That is, Hoglund discloses a tool that is not used during the boot sequence at all, because the method implemented by the rootkit revealer could not be implemented during the boot sequence. (*See id.*)

90. Accordingly, Petitioner fails to allege that the prior art teaches scanning during a second period in the boot sequence; Garfinkel does not disclose or suggest scanning during a second period in the boot sequence; and Hoglund discloses a tool that is only capable of operating after the boot sequence has already completed. Thus, the cited prior art does not disclose or render obvious scanning a registry of

the computer “*during the second period of the boot sequence.*”

b. Garfinkel and Hoglund do not render obvious scanning data in a registry of the computer for pestware

91. Petitioner fails to show that a POSITA would have been motivated to combine Garfinkel’s and Hoglund’s teachings to render obvious scanning data in a registry of the computer for pestware.

92. Petitioner—at most—asserts that Garfinkel only discloses a system with the *capability* of scanning memory with access to a registry, and its expert Dr. Nielson concedes that “Garfinkel does not explicitly disclose scanning the registry.” (Petition 21; EX-1003 ¶231.) Consequently, Petitioner argues that Hoglund discloses rootkit revealer, which Petitioner characterizes as a “scanning tool,” and argues that a POSITA would have been motivated to combine Garfinkel’s signature detector with Hoglund’s registry scanning where Garfinkel’s IDS hosts a Windows-based operating system. (Petition 21-22.)

93. Petitioner and Dr. Nielson fail to acknowledge the critical distinctions between Garfinkel’s signature detector and Hoglund’s rootkit revealer. Garfinkel discloses a polling-type policy module called a “signature detector,” which scans memory for known attack signatures—e.g., known signature substrings of the program. (EX-1006 §6.1.3.) Garfinkel teaches that such an approach “requires a more careful selection of signatures to avoid false positives.” (EX-1006 §6.1.3.)

And while Garfinkel references a list of “[p]opular” attack tools that includes “back-orifice,” this is not one of the attack tools that Garfinkel identifies as having been captured or prevented by any of its policy modules, including the signature detector. (EX-1006 Table 1.)

94. Furthermore, Petitioner states, “Garfinkel discloses a signature scanner that scans *files*, buffer cache, and memory with access to memory registers.” (Petition 21 (emphasis added).) However, Garfinkel explicitly states, “Our signature detector performs a scan of all of host memory for attack signatures.” (EX-1006 §§6.1.3). Garfinkel does not teach scanning the file system. Thus, Garfinkel’s signature scanner would not scan any portion of the registry stored on disk and could not form the basis of a registry scanner.

95. Hoglund does not disclose a signature detector and does not teach or suggest applying a signature scanner to a Windows registry. (*See* EX-1007 359.) Hoglund teaches that the rootkit revealer may be used to detect specific changes to the registry by pulling data out of the registry through two different methods and comparing the results to detect discrepancies. (EX-1007 359.) But Petitioner’s expert recites this disclosure and then makes the unsupported conclusion that Hoglund discloses “Windows scanning rootkit signature detectors.” (EX-1003 ¶¶234-236.) Hoglund does not teach scanning data in the Windows registry for specific values, e.g., a signature. (*See* EX-1007 359.) Rather, Hoglund merely

discloses a means for detecting changes to the registry. (*Id.*)

96. Petitioner does not acknowledge or discuss these critical distinctions between Garfinkel’s and Hoglund’s methods, nor explain how a POSITA might have modified Garfinkel’s signature detector in view of Hoglund’s teachings—which do not relate to signature scanning. Accordingly, a POSITA would not have been motivated to combine Garfinkel’s and Hoglund’s distinct teachings to “*scan[] data in a registry of the computer for pestware during the second period in the boot sequence.*”

97. Finally, Petitioner also makes a general argument regarding motivation to combine Garfinkel’s and Hoglund’s teachings, asserting that because Garfinkel’s virtual machine can host a Linux-based or Windows-based operating system as the “guest OS,” a POSITA would understand Garfinkel’s teachings to be “interchangeable for equivalent stages in either boot process.” (Petition 10-12.) The accuracy of this generalized statement would be context-dependent and require analysis of the specific teachings, given the unique characteristics of Linux and Windows operating systems. But even if this were true, as I’ve just explained, Garfinkel discloses a signature detector, whereas Hoglund discloses a means for detecting changes to the registry—two distinct methods that are not interchangeable. Furthermore, as I discussed immediately above in Section VI.A.2.a, neither Garfinkel nor Hoglund teach or suggest scanning a registry during a boot sequence

and certainly not “*during the second period of the boot sequence.*”

3. The cited combination of prior art does not render obvious generating, in response to monitoring, a record of events.

98. Limitation [1e] recites “generating, in response to the monitoring, a record of events, the record of events including the pestware-related events.” (EX-1001 cl. 1; see also *id.* cls. [6g], [11e].)

99. Petitioner’s argument that Garfinkel discloses basing event-driven inspection “on a log of events” relies on a proposed modification of Garfinkel’s disclosed system—a modification from which Garfinkel teaches away. (Petition 18.) Furthermore, Petitioner does not demonstrate that the resulting, modified system would continue to disclose or render obvious any of the previously-addressed limitations.

100. Garfinkel teaches that the authors’ prototype takes “an extremely conservative approach to introspection by primarily engaging in passive checks that incur no visible impact on system performance.” (EX-1006 13 col.2.) Garfinkel acknowledges that where a system designer wishes to monitor “frequent asynchronous events, e.g., all system calls,” such an endeavor would be “quite performance intensive.” (*Id.*) But Garfinkel discloses, under “Future Work,” that this could be supported by its VMI IDS by “hard-coding the functionality to log these events directly into the monitor, then offloading the processing of these logs

to the policy engine.” (*Id.*) “Asynchronous” events refer to events that may occur simultaneously—i.e., execution of each operating is not dependent on completing the one before it. That is, rather than registering callbacks for specific events and monitoring those events in real-time, Garfinkel’s IDS would be modified such that the VMM logs asynchronous events and passes the analysis of those logs to the IDS after-the-fact. (*See* EX-1006 13 col.2.)

101. But critically, Garfinkel teaches away from making such a modification. After discussing the possibility of hard-coding the system, such that the VMM logs events, Garfinkel explains:

However, this approach seems somewhat inflexible. We believe a more promising approach would involve support for providing a small, safe and extensible mechanism for efficiently filtering architecture events in the VMM, in much the same fashion that current OSes provide this functionality for filtering packets via BPF.

(EX-1006 13 col.2.) Thus, Garfinkel criticizes the proposed logging of events and guides a POSITA to an alternative approach.

102. Additionally, Petitioner does not address how such a drastic modification of Garfinkel’s IDS would affect its operation with respect to the previously addressed limitations. For example, if a POSITA were to hard code the IDS to log events directly into the monitor and offload processing of that log to the policy engine, the policy modules would be applied to these events after-the-fact.

(See EX-1006 13 col.2.) While Petitioner's assertions as to whether Garfinkel teaches managing pestware-related events during a boot sequence are already incorrect, for the reasons I addressed above, they are certainly incorrect in the modified IDS that Petitioner proposes, because the logs (and the events they record) will be analyzed after the events have occurred. Where the policy engine is retroactively analyzing a log of events recorded by the VMM, there is no disclosure of applying policy modules during a boot sequence or at any particular time in a boot sequence. Even if Petitioner could show that events being logged were limited to the boot sequence (which it has not done), there would be no need to apply the policy modules during the boot sequence since the VMM is not interrupting and managing events in real-time.

103. Accordingly, a POSITA would not have been motivated to modify Garfinkel's disclosed IDS to generate a record of events as alleged, because Garfinkel teaches away from such a modification, and even if a POSITA were to modify the system, then Petitioner has not shown how the modified system discloses or renders obvious all limitations of the independent claims.

B. Dependent Claims

104. Because Garfinkel in view of Hoglund does not render obvious all limitations of the independent claims, I understand that Ground 1 fails to render obvious the challenged dependent claims as well.

VII. CLAIMS AND/OR LIMITATIONS NOT ADDRESSED IN THIS DECLARATION

105. It is my understanding that this declaration is being provided in support of Patent Owner's Preliminary Response. I have provided analysis of key limitations not included in the prior art. The fact that I have not analyzed particular limitations or claims in no way indicates or supports that these limitations or claims are disclosed or rendered obvious by the prior art, and I reserve the right to supplement my analysis if asked to do so.

VIII. SECONDARY OBJECTIVE INDICIA OF NONOBVIOUSNESS

106. I understand that in determining obviousness of the claimed invention, objective evidence of secondary considerations pertaining to nonobviousness must be considered, including long-felt but unsolved needs, failure of others, commercial success, etc. (*In re Huang*, 100 F.3d 135, 139 (Fed. Cir. 1996) (citing *Graham v. John Deere Co.*, 383 U.S. 1, 17-18 (1966).) I further understand that these indicia constitute independent evidence of nonobviousness that can be the most probative evidence in that regard "in avoiding the trap of hindsight when reviewing, what otherwise seems like, a combination of known elements." (*Leo Pharm. Prods., Ltd. v. Rea*, 726 F.3d 1346, 1358 (Fed. Cir. 2013).)

107. As demonstrated by my summary of the prior art, above, there was at least a long-felt and persistent need prior to the '505 patent to improve upon the inadequate prior art methods for managing pestware. (*See* Section III, *supra*.) Prior

attempts to develop such methods were limited in their ability to manage and prevent pestware that was installed during the boot sequence, before conventional anti-virus and anti-pestware systems were configured and running. (*See* Section III, *supra*.) Where others failed, the '505 Patent succeeded. (*See* Section IV(A), *supra*.)

108. Thus, these secondary indicia only further support my opinion that the '505 Patent was non-obvious.

IX. CONCLUSION

109. For at least the foregoing reasons, it is my opinion that Petitioner's ground does not render obvious any of the challenged claims of the '505 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 22nd day of December, 2023 in Sunrise, Florida.

Respectfully submitted,

A handwritten signature in black ink, appearing to read 'Ron Schnell', with a stylized, sweeping flourish at the end.

Ron Schnell